LLM for Mobile: An Initial Roadmap

DAIHANG CHEN, Beihang University, China
YONGHUI LIU, Monash University, Australia
MINGYI ZHOU, Beihang University, China
YANJIE ZHAO, Huazhong University of Science and Technology, China
HAOYU WANG, Huazhong University of Science and Technology, China
SHUAI WANG, Hong Kong University of Science and Technology, Hong Kong
XIAO CHEN, The University of Newcastle, Australia
TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg
JACQUES KLEIN, University of Luxembourg, Luxembourg
LI LI*, Beihang University, China

When mobile meets LLMs, mobile app users deserve to have more intelligent usage experiences. For this to happen, we argue that there is a strong need to apply LLMs for the mobile ecosystem. We therefore provide a research roadmap for guiding our fellow researchers to achieve that as a whole. In this roadmap, we sum up six directions that we believe are urgently required for research to enable native intelligence in mobile devices. In each direction, we further summarize the current research progress and the gaps that still need to be filled by our fellow researchers.

CCS Concepts: • **Software and its engineering → Software safety**; **Software reliability**.

1 INTRODUCTION

Large Language Model (LLM) has been the emergent buzzword in the SE community since the successful release of ChatGPT, a conversation-based AI system powered by OpenAI's GPT-3.5 model, and the successful release of Copilot, GitHub's AI developer tool supported by OpenAI's Codex model. It quickly becomes the most popular research topic in software engineering (if not in computer science). The research efforts mainly focus on exploring two directions. The first direction is related to applying SE methods to improve LLMs. Indeed, as a new technique, LLM also comes with limitations that need to be resolved in order to apply LLMs in practice, as has happened with other emerging technologies. The other direction is to apply LLMs to resolve traditional SE tasks (e.g., code generation, unit test generation, etc.). Our fellow researchers have experimentally shown that LLMs can achieve better results, compared to approaches that do not use AI or only adopt pre-LLM AI techniques.

*Corresponding author (lilicoding@ieee.org).

Authors' addresses: Daihang Chen, Beihang University, China; Yonghui Liu, Monash University, Australia; Mingyi Zhou, Beihang University, China; Yanjie Zhao, Huazhong University of Science and Technology, China; Haoyu Wang, Huazhong University of Science and Technology, China; Shuai Wang, Hong Kong University of Science and Technology, Hong Kong; Xiao Chen, The University of Newcastle, Australia; Tegawendé F. Bissyandé, University of Luxembourg, Luxembourg; Jacques Klein, University of Luxembourg, Luxembourg; Li Li, Beihang University, China.

ACM 1557-7392/2024/12-ART
https://doi.org/10.1145/3708528

Mobile Software Engineering (MSE) has been a hot research area in Software Engineering (SE). It generally involves applying traditional software engineering methodologies (concepts, methods, tools, models, programming styles) to mobile software systems (such as Android or iOS) and apps, which are often distributed through app stores [88, 99, 101, 119, 215]. So far, this hot research topic has attracted lots of attention from software engineering researchers who have subsequently made significant contributions to the MSE community from various aspects, such as Security and Privacy Analysis [41, 66, 98, 100, 104, 154, 166], App Quality Assurance [15, 102, 103, 167, 226], App Store Analysis [140, 183, 184], etc.

With the great results achieved by applying LLMs for SE and the flourishing mobile ecosystem, we believe it is time to apply LLMs for mobile. The smart devices will only be "smart" if LLMs are embedded. Actually, many smartphone producers have already started to deploy LLM directly on devices. For example, OPPO claims that it is the first smartphone company to deploy an LLM with 7 billion parameters directly on the device.[1] As highlighted on the official Android website, On-device AI is a great solution for use cases where low latency, low cost, and privacy safeguards are primary concerns. Indeed, there are many scenarios where rich AI experiences are more suitable to be done on the device instead of on the cloud. For example, latency-sensitive tasks such as phone call translation need to be achieved nearly at the same time. Data-intensive tasks such as automated online meeting summarization need to be achieved on the device as the meeting records are usually very big and thereby will require huge data transmission costs if processed on the cloud. Other AI tasks that have to be implemented based on on-device models include features that need to be always on no matter the phone is connected to the internet or not, or features that require sensitive user data that is not allowed to be sent outside the devices.

At the moment, our fellow researchers have also seen the opportunities to apply LLMs for mobile and hence conducted several studies in this field. However, the research roadmap for applying LLM for mobile has not yet been sketched. To fill this research gap, in this position paper, we commit to summarizing the initial roadmap of applying LLM for mobile. Figure 1 provides an overview of the roadmap. In general, we divide the LLM for mobile tasks into two phases: LLM Supply and LLM Use. The former phase involves preparing the right LLMs for solving downstream tasks, while the latter phase concerns the usages (or inference) of LLMs in mobile devices through local models (i.e., deployed in the device as part of the operating system) or online models (i.e., deployed in the cloud). In these two phases, we further summarize six research directions that need to be further researched in order to seamlessly integrate LLMs into the mobile ecosystem. The six directions are depicted below.

- **Preparing datasets for fine-tuning LLMs dedicated to mobile.** In particular, we advocate that the datasets should include User Experience (UX) scenarios that enable better ways for apps to interact with LLMs, SE scenarios that allow more efficient app development and analyses, and other multi-modal data processing scenarios (e.g., sensor data, app logs) that enable LLMs to process various types of data and tasks on mobile devices.
- **Applying LLMs for mobile app development and analysis.** For app development, the whole lifecycle (i.e., requirement, design, coding, testing and debugging, maintenance, etc.) should be considered. For app analysis, both static code analysis and dynamic app testing need to be covered.
- **Serving LLM on mobile.** LLMs are powerful in processing our personal or work data. Deploying LLMs on cloud servers must share the user's sensitive data with remote servers. In addition, accessing remote LLMs is impractical where internet connectivity is unavailable. These problems are non-trivial to solve in cloud-based LLMs. So, deployment of the LLMs directly on mobile devices is a better option. Yet deploying these "large" LLMs on resource-constrained mobile devices poses a challenge. Thus, innovative full-stack solutions are necessary to efficiently serve LLMs on mobile devices.
- **Defending against security exploits targeting on-device LLMs.** Because of concerns such as internet connection, privacy, and data transmission, in certain circumstances, LLMs need to be deployed directly

[1] https://www.oppo.com/en/newsroom/press/oppo-make-ai-phones-accessible/

on devices. In such a situation, it increases the attacking surface by opening new security problems to end users. Indeed, the attack surface of LLMs deployed on devices is much larger than those deployed over the cloud, as the physical on-device LLMs are stored in mobile devices that are easily accessible to attackers. Unlike the privacy leakage and internet connectivity issues of deploying LLMs on cloud servers, on-device deployment poses new challenges that require dedicated approaches to tackling. Our fellow researchers have taken the initiative to achieve this objective. Representative works include the integration of trustworthy execution environment (TEE) [153, 180], obfuscation [230], and customization [229] to reduce the attacking surface. Therefore, better-defending approaches will be very important in LLM deployment.

- **Providing LLM-powered framework APIs.** Expectedly, mobile apps are interested in accessing LLMs to enable intelligent features. However, it would be challenging for app developers to directly interact with LLMs, especially if they lack the necessary AI knowledge. We therefore argue that there is a need to provide well-designed framework APIs to facilitate intelligent app development.
- **Providing LLM-powered runtime app monitoring.** Recent studies have presented various runtime monitoring techniques for mobile apps where provenances are collected and analyzed by remote app vendors to facilitate runtime profiling, performance optimization, and even mitigating security exploitations. We anticipate LLMs can offer highly intelligent runtime monitoring techniques to reason about the provenances and provide insights into the runtime behavior of mobile apps. Moreover, while recent studies have shown the potential privacy risks when uploading app logs to remote servers, we note that LLMs on mobile can be used to analyze these sensitive logs locally without leaking sensitive information to remote servers.

We elaborate on these directions in the following sections.

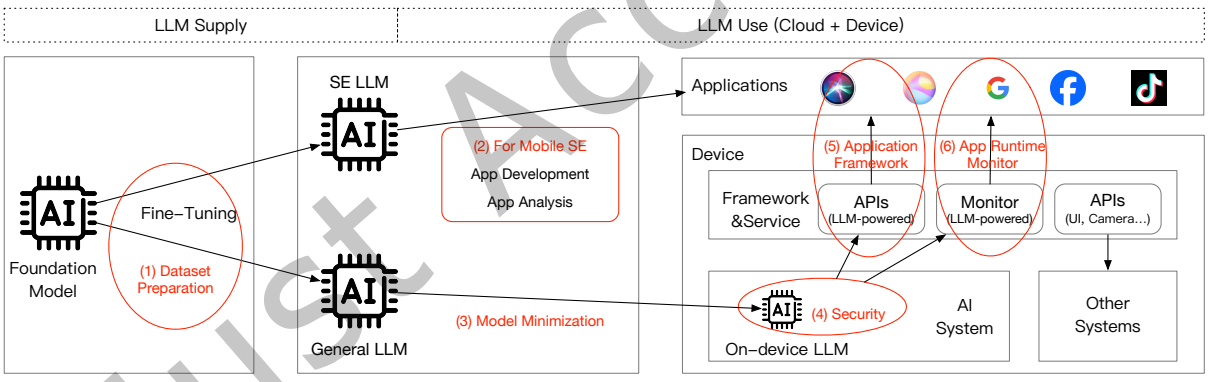

Fig. 1. Roadmap in Applying LLMs for Mobile.

2 PREPARING DATASET FOR FINE-TUNING LLMS

In the domain of software engineering (SE), the preparation of datasets is crucial for the effective training and fine-tuning of LLMs [168]. Accurate, high-quality, and diverse datasets not only enhance the model's generalization capabilities but also optimize its performance, ensuring reliability in validation and testing. When preparing datasets for fine-tuning LLMs, especially within SE, User Experience (UX), and other multi-modal data processing scenarios, researchers must focus on the collection, classification, preprocessing, and representation of data to ensure its richness and diversity.

2.1 Application Scenarios

SE Datasets. The primary application domain would be SE scenarios, for which dataset preparation needs to center around specific SE tasks such as code comprehension, bug fixing, code generation, and more. Data sources can be divided into four main categories [64]: open-source datasets, collected datasets, constructed datasets, and industrial datasets. *Open-source Datasets* [16, 83, 186, 213]: Publicly accessible datasets distributed via open-source platforms or repositories. For example, the HumanEval dataset [17] contains 164 manually created Python problems with their unit tests. *Collected Datasets* [67, 128, 157, 174]: Datasets compiled by researchers from various sources such as websites, forums, blogs, and social media. Data is often extracted from Stack Overflow threads or GitHub issue comments to tailor datasets for specific research queries. *Constructed Datasets* [32, 80, 86, 219]: Datasets specifically designed by researchers by altering or enriching collected data to closely match particular research goals. This includes manually annotating code snippet datasets to study automated program repair technologies, among others. *Industrial Datasets* [4, 134, 189]: Comprise proprietary business information, user behavior logs, and other sensitive data from commercial or industrial firms. These datasets are crucial for research targeting real-world business situations but usually require navigating legal barriers to protect commercial interests.

The current research landscape reveals a significant reliance on open-source and collected datasets due to their accessibility and reliability. However, there's a notable gap in the use of constructed datasets (mainly on how are the dataset pre-processed for LLMs) and industrial datasets, indicating a potential disconnect between academic research datasets and those encountered in real-world industrial contexts. Future research directions should aim to bridge this gap by exploring the use of industrial datasets, ensuring that LLMs are applicable and robust across both academic and industrial scenarios.

UX Datasets. In the mobile domain, UX scenarios are also important. Towards improving the user experience of using mobile devices, one imperative task is to identify the list of scenarios that can be powered by LLMs. To achieve this, it requires to prepare suitable datasets (e.g., diverse user-system interaction data) to train and fine-tune LLMs. Key data sources include the following. *User Interaction Logs*: Records of user actions within software, websites, or apps, which provide insights into behavior patterns, task workflows, and interface pain points. For example, the new user-apps interactions dataset for behavioral profiling using smartphones [6] presents over 3 million actions derived from user interactions, offering valuable data to improve smartphone security. *User Feedback and Reviews*: Comments from app stores, social media, forums, and review systems, providing insights into user satisfaction and expectations. For example, the Google Play Store Reviews dataset [152] includes over 12,000 reviews, allowing classification into positive or negative sentiments based on ratings. Similarly, app_reviews dataset [48] contains approximately 280,000 user reviews from 395 different open-source Android applications. It offers comprehensive feedback on software maintenance and evolution, helping to understand user preferences and app effectiveness. *User Surveys and Interviews*: These direct sources reveal user needs and preferences. The challenge lies in converting responses into a structured format for LLM learning, necessitating careful coding and categorization. For example, the Worldwide Mobile App User Behavior Dataset [112] provides extensive insights into global mobile app usage behaviors and demographics, which can be instrumental in understanding diverse user outlooks across different regions. *User Testing and Experiments*: Conducted in controlled settings, this data shows how design choices affect user behavior and satisfaction. It's crucial to understand the impact of different interface designs and functionalities.

In streamlining the discussion on personalization and adaptation in UX scenarios for LLMs, we focus on the essence of crafting user-centric software solutions. The process hinges on analyzing User Interaction Logs, Feedback and Reviews, and insights from Surveys and Interviews to tailor experiences that resonate with individual preferences. By dynamically adjusting content and interactions based on a deep understanding of user behaviors and patterns, the software can offer a more personalized journey, enhancing user engagement and

satisfaction. The challenge lies in balancing personalized experiences with privacy and security, ensuring data is handled with care. Moreover, adaptation goes beyond customization to evolve with user feedback and subtle cues, like device type or location, to anticipate and meet unexpressed needs, thereby fostering a deeper connection with the user. Despite the hurdles of privacy concerns, bias mitigation, and technological limitations, the goal is to develop LLM-powered applications that are not just functional but intuitive and engaging. This condensed narrative underscores the importance of personalization and adaptation in moving towards more human-centric, responsive, and ultimately more effective software solutions.

Multi-modal Datasets. Further to the above directions, it is also essential to consider handling multi-modal data available on mobile devices. To date, large models have demonstrated emerging capabilities in handling tasks over multi-modal data, such as text, image, audio, etc. Importantly, deploying LLMs in mobile devices offers multi-modal data exposures, as modern mobile platforms can face various types of domain-specific data from users (e.g., text, photos, audio), sensors (e.g. accelerometer, gyroscope, GPS), wearable devices (e.g., heart rate, sleep quality), and network. Recent studies have shown that LLMs can be fine-tuned to comprehend textualized signal collected from sensors [202]. Nevertheless, the integration of LLMs with multi-modal data processing in mobile devices remains largely unexplored. We envision key challenges coming from numerous data sources, data formats, and data types, which require innovative approaches to process and analyze.

More importantly, we envision the possibility of instructing LLMs to process various logs and traces generated by mobile apps and even the mobile operating system (OS) itself. We aim to leverage LLMs to analyze those logs and traces to facilitate runtime profiling, debugging, and performance optimization (see further technical details and discussions in Sec. 7). Moreover, we anticipate the technical solutions for runtime detection of security exploitations, penetrations, and other anomalies of apps and OS using LLMs. Supporting this vision, we advocate for the community to provide datasets that include logs, traces, and other dimensions of provenances to enable proper fine-tuning and calibration of LLMs.

2.2 Fine-tuning Techniques

This technique involves updating all or most of the model's parameters during the fine-tuning process [126]. In mobile development, this could be used when adapting a general-purpose LLM to become a specialized mobile development assistant. For example, fine-tuning the entire model on a large corpus of Swift and iOS development documentation and code examples to create an iOS-specific coding assistant.

Parameter-Efficient Fine-tuning (PEFT). PEFT techniques [29, 59, 116] aim to adapt models with high performance while updating only a small subset of parameters, which is particularly useful when computational resources are limited.

LoRA (Low-Rank Adaptation). LoRA [65] adds trainable rank decomposition matrices to each layer of the model. In mobile development, this could be used to efficiently adapt a model to a specific framework like React Native, adjusting its understanding of JavaScript and React paradigms in the mobile context without needing to retrain the entire model.

Prefix Tuning. This method prepends trainable prefixes to the input of each transformer layer [109]. For mobile development, this could be used to adapt a model to understand company-specific coding practices or internal libraries, by adding context about the company's development style at each layer of the model.

Prompt Tuning. Similar to prefix tuning, but only prepends trainable vectors to the input layer [96]. This could be useful for task-specific tuning in mobile development, such as optimizing the model for generating UI layout code or handling platform-specific APIs.

Instruction Fine-tuning. This technique involves fine-tuning the model on a dataset of instruction-output pairs [221]. In mobile development, this could be particularly useful for enhancing the model's ability to follow specific development instructions or coding standards. For example, fine-tuning the model to understand and

generate code based on platform-specific design guidelines like Material Design for Android or Human Interface Guidelines for iOS.

Multi-task Fine-tuning. This approach involves fine-tuning the model on multiple related tasks simultaneously [127]. In mobile development, this could be used to create a versatile assistant capable of handling various aspects of app development, such as simultaneously training on code generation, debugging, and performance optimization tasks.

Continual Learning. This technique allows the model to learn new information over time without forgetting previously learned knowledge [187]. In the rapidly evolving field of mobile development, this could be used to keep the model updated with the latest SDK changes, new programming paradigms, or emerging best practices without losing its understanding of fundamental concepts.

Few-shot Fine-tuning. This method fine-tunes the model using only a small number of examples [116]. In mobile development, this could be particularly useful for quickly adapting a model to a new framework or language feature without requiring a large dataset. For instance, updating a model's understanding of new SwiftUI features with just a handful of code examples.

Domain-Adaptive Pretraining. This two-stage approach first pre-trains the model on domain-specific data before fine-tuning it on the target task [54]. For mobile development, this could involve pre-training on a large corpus of mobile development documentation and open-source code across multiple platforms, followed by fine-tuning for specific tasks or company needs.

3 APPLYING LLMS FOR MOBILE APP DEVELOPMENT AND ANALYSIS

This section proposes a holistic framework that utilizes the advanced capabilities of LLMs to address critical aspects of mobile app development and analysis. By seamlessly integrating LLMs into processes such as app development, code analysis, app testing, privacy evaluations, and app market analysis, we aim to ensure a secure, user-centric, and optimized digital ecosystem. We now detail a vision where LLMs empower stakeholders across the mobile app landscape, enhancing every facet from code integrity to market dynamics.

3.1 Objectives

Requirements Engineering for Mobile Apps. In the specific context of mobile app development, LLMs have the potential to significantly enhance requirements engineering by assisting in the translation of user needs into clear, actionable requirements tailored for mobile platforms. They can enhance communication among stakeholders, which is crucial for capturing the unique demands of mobile users. LLMs can be instrumental in crafting precise documentation and use cases that reflect the mobile user experience, taking into account the constraints and capabilities of mobile devices. However, it's crucial to acknowledge the challenges associated with LLM use in this critical phase, particularly the risk of hallucinations that could lead to incorrect or misleading requirements. To mitigate risks, a hybrid approach combining LLM assistance with human expertise and validation is essential. While LLMs can facilitate the validation process by checking for completeness and consistency, human oversight remains crucial to ensure the accuracy and relevance of requirements. This approach aims to minimize the risk of expensive modifications during the critical stages of mobile app development and align the project closely with mobile user expectations and project objectives. The decision to use LLMs in requirements engineering for mobile apps should be made carefully, weighing the potential benefits of efficiency and comprehensiveness against the risks of inaccuracies, and implementing robust validation processes to leverage LLM capabilities while safeguarding against their limitations.

App Development. LLMs could revolutionize the way developers conceive, design, and implement mobile apps. While LLMs are already being used for code development, their potential in mobile app development extends beyond current applications. By providing real-time coding assistance, generating code snippets based

on developer prompts, and offering optimization suggestions, LLMs can significantly reduce development time and elevate code quality [10, 45, 74, 111, 117, 145]. Specifically for mobile apps, LLMs can address the unique challenges posed by the dynamic nature of mobile devices and varying operating systems. In addition to high-level assistance, LLMs can delve into the intricacies of algorithm optimization, suggesting efficient data structures and algorithms tailored to the app's specific needs and constraints. For example, a study by Feng Lin et al. introduces LCG [113], a code generation framework inspired by software engineering practices. LCG leverages various software process models such as LCGWaterfall, LCGTDD, and LCGScrum, with LLM agents assuming roles like requirement engineer, architect, developer, tester, and Scrum master. These agents use chain-of-thought and prompt composition techniques to continuously refine and enhance code quality. This is particularly crucial for mobile apps where resource management and performance optimization are critical due to device limitations.

Moreover, through code interpretation, LLMs can elucidate complex code segments, offering clarifications and detailed explanations that enhance developers' understanding of their own and others' code. This leads to improved debugging and maintenance efficiency. For mobile apps, LLMs can be trained to understand platform-specific APIs and frameworks, providing more targeted assistance. The integration of code refactoring capabilities could allow LLMs to suggest structural improvements that increase the readability and performance of the codebase, promoting best practices and design patterns. In the context of mobile apps, LLMs can be utilized for dynamic code refactoring, adapting to changing device capabilities and user interactions in real time.

App Code Analysis. The core functionality of an app hinges on its complex code, requiring detailed analysis to ensure performance and security. LLMs provide powerful, comprehensive analysis beyond traditional methods [52, 105, 120, 125, 157, 177, 220]. While there are many non-LLM-based static analysis tools available, such as pylint, LLMs offer unique advantages in mobile app code analysis. For example, LLMs can improve static code analysis to thoroughly inspect code without running it, identifying complexities, compliance with coding standards, and risky API uses [60, 165]. Unlike traditional tools, LLMs can understand context and nuances in code, potentially identifying subtle issues that rule-based systems might miss. They can also adapt to new coding patterns and languages more easily, which is crucial in the rapidly evolving mobile app ecosystem. This proactive analysis is pivotal in identifying security vulnerabilities, code smells, and performance bottlenecks, effectively preempting issues before they escalate into more significant problems.

LLMs can also enhance code clone detection by analyzing code's syntax and semantics to identify duplicates across apps [23, 30, 73, 82]. This could help prevent app cloning, protect originality, and avoid licensing issues, preserving the app ecosystem's integrity. Furthermore, LLMs can be leveraged for dynamic code analysis in mobile apps, understanding how code behaves during runtime across different devices and operating system versions. This capability is particularly valuable for identifying issues that only manifest under specific runtime conditions.

App Testing and Optimization. Achieving a seamless and faultless app experience necessitates a relentless pursuit of perfection through rigorous testing and constant optimization. LLMs are revolutionizing this process by automating various facets of testing and optimization [107, 108, 162, 185, 199, 206, 212]. In GUI testing [121, 209], for instance, LLMs can automate the generation of test cases, predict potential user interactions, and validate UI elements for accessibility and usability standards. For mobile apps specifically, LLMs can generate test scenarios that account for various screen sizes, orientations, and input methods unique to mobile devices. This automation extends to bug replay and fixing [69, 78, 79], where LLMs can intelligently suggest corrections and optimizations for identified issues, reducing the manual effort required from developers. In the mobile context, LLMs can be trained to understand and replicate device-specific bugs, such as those related to battery usage or sensor interactions. Moreover, LLMs can optimize app performance by analyzing usage patterns and resource consumption, suggesting efficient algorithms, and predicting user behavior to preload resources or functionalities. This is particularly crucial for mobile apps where performance directly impacts user experience and battery life.

This level of automation and insight not only accelerates the development cycle but also ensures that the final product stands up to the highest standards of quality, performance, and user satisfaction.

Privacy-related Analysis. As digital privacy [56, 71, 141, 170] becomes increasingly paramount, LLMs offer a novel approach to navigating the complexities of privacy policies and compliance. By demystifying privacy policies through data mining and ensuring that apps adhere to regulatory standards, LLMs could play a crucial role in fostering a transparent and trust-based relationship between apps and their users.

App Market Ecosystem Analysis. In the ever-changing landscape of the app market [234], staying abreast of trends and competitive dynamics is key to success. LLMs can offer unparalleled insights into market movements, user preferences, and competitive strategies, empowering developers and marketers to make informed decisions that drive growth and innovation. For example, the voice of the user, encapsulated in reviews, holds invaluable insights into the app experience. Harnessing LLMs to mine this data, developers, and researchers can extract pivotal information, classify sentiments, and detect spam with higher accuracy [43, 89, 207]. This not only amplifies the value derived from user feedback but also equips developers with the tools to prioritize enhancements and foster an engaging user experience.

3.2 Prompt Enhancement

To achieve the aforementioned objectives, there are various challenges need to be addressed. Representative ones include the infamous hallucination problem (we briefly introduce it in the next subsection) that cannot be avoided by LLMs alone. Therefore, researchers have explored various approaches to mitigate this problem (so as to improve the inference results) by enhancing the prompts. We now present some of the key techniques that have been adopted by our fellow researchers.

3.2.1 Prompting techniques. It has been demonstrated that the results of LLMs can be improved by directly updating the prompts. Some of the representative updating methods are summarized below.

Task-specific prompts. Task-specific prompts [188] are tailored instructions designed to guide LLMs in performing specialized mobile development tasks. For instance, when generating code for a mobile app's user interface, a prompt might include specific details about the desired layout, widget types, and platform-specific guidelines (e.g., Material Design for Android or Human Interface Guidelines for iOS).

Few-shot learning. Few-shot learning [190] in the context of mobile software engineering involves providing the LLM with a small number of examples to guide its output. This technique is particularly useful for maintaining consistency in coding style or implementing platform-specific patterns.

Chain-of-thought reasoning. Chain-of-thought reasoning [193] prompts guide LLMs through a step-by-step problem-solving process, which is crucial for complex mobile development tasks. This technique is particularly valuable for performing complex reasoning such as improving apps' performance.

Role-playing prompts. Role-playing prompts instruct the LLM to assume a specific role within the mobile development process [87, 192]. This technique can be particularly effective for code reviews, architecture discussions, or debugging sessions.

Template-based prompts. Template-based prompts use predefined structures to ensure consistent output in mobile app development tasks [118]. This is particularly useful for generating boilerplate code or implementing standard mobile app components.

Context-enhanced prompts. Context-enhanced prompts incorporate project-specific information, coding standards, or platform-specific details to generate more relevant and tailored code [7]. For mobile development, this might include details about the app's architecture, target platforms, or specific libraries in use.

3.2.2 RAG techniques. Except for directly updating the prompts, providing additional information (that could be retrieved from a database) could further improve LLM's results. Our fellow researchers have already explored various ways to leverage RAG. Some of the representative RAG-based methods are summarized below.

Standard RAG. This is the basic form of RAG where relevant documents are retrieved based on the input query and then used to augment the context for the language model [42]. In mobile development, this could involve retrieving relevant API documentation or code snippets based on the developer's query. For example, when a developer asks about implementing push notifications in iOS, the system retrieves relevant documentation from Apple's developer resources before generating a response.

GraphRAG. GraphRAG enhances the standard RAG approach by representing knowledge in a graph structure, allowing for more nuanced and interconnected information retrieval [31]. In mobile development, this can be particularly useful for understanding complex relationships between different components of an app or SDK. For instance, when dealing with Android's activity lifecycle, GraphRAG can help retrieve not just individual method descriptions, but also how these methods interact and influence each other, providing a more comprehensive understanding.

Adaptive RAG. This technique dynamically adjusts the retrieval process based on the complexity and specificity of the query [72]. In mobile development, this could mean retrieving more detailed technical documentation for advanced queries and simpler, high-level explanations for beginner questions. For example, a query about basic UIKit elements might trigger retrieval of introductory documentation, while a question about advanced Core Animation techniques would lead to more in-depth technical papers.

Multi-Vector RAG. Multi-Vector RAG uses multiple embedding vectors for each document, capturing different aspects or levels of information. This is particularly useful in mobile development where a single piece of documentation might contain information relevant to different levels of expertise or different aspects of development (e.g., UI, performance, security). For instance, when retrieving information about RecyclerView in Android, it could simultaneously capture beginner-level usage, performance optimization tips, and accessibility considerations.

Hybrid RAG. This approach combines retrieval-based methods with other techniques like few-shot learning or fine-tuning [211]. In mobile development, this could involve retrieving relevant documentation and then using few-shot examples to generate more context-appropriate code. For example, when assisting with SwiftUI development, the system might first retrieve relevant view modifiers documentation and then use few-shot examples to demonstrate how to combine these modifiers effectively.

Iterative RAG. Iterative RAG [208] involves multiple rounds of retrieval and generation, refining the results with each iteration. This can be particularly useful for complex mobile development tasks that require breaking down problems into smaller steps. For instance, when helping to architect a large-scale mobile application, the system might first retrieve high-level architectural patterns, then iteratively retrieve more specific information about implementing each component.

Conversational RAG. This technique is designed to maintain context over a longer conversation, retrieving relevant information based on the entire conversation history. In mobile development, this is useful for prolonged debugging sessions or when walking through the process of building a feature. For example, during a conversation about implementing authentication in a React Native app, the system would keep track of previously discussed topics (like setting up the development environment) to provide more contextually relevant information in subsequent retrievals.

4 SERVING LLM ON MOBILE

LLMs have revolutionized NLP tasks with remarkable success on general tasks. With growing concerns over data privacy and the stringent response latency requirement, running the LLM on mobile devices locally has attracted

attention from both academia and industry. However, their formidable size and computational demands present significant challenges for practical deployment on resource-constrained mobile devices. This section exclusively focuses on techniques that can be applied to pre-existing LLMs with minimal training efforts, up to the level of fine-tuning, rather than delving into the complexities of designing hardware and models specifically tailored for mobile devices. Accomplishing full-stack on-device inference optimization necessitates a comprehensive approach that takes into account various aspects of the model, hardware, software, and deployment stack. Among these optimizations, model-level optimization (model compression) is often considered the most crucial for deploying LLMs on mobile devices.

Model Compression techniques have been intensively investigated to reduce the LLM size and computational complexity without significantly impacting its performance. We categorized 4 model compression techniques as detailed in the following, including *Pruning*, *Knowledge Distillation*, *Quantization*, and *Low-rank Factorization*. **Pruning** is one extensively studied technique [58, 93, 97] for removing non-essential components in the model. Based on removing entire structural units or individual weights, *Pruning* can be divided into Structured Pruning [8, 35] or Unstructured Pruning [46, 222], respectively, both of which target weight reduction without modifying sparsity during inference. Contextual pruning [122, 179] differs from the above by its dynamic nature, adjusting the model in real-time based on the context of each inference task. **Knowledge Distillation** (KD) [62, 85, 178] enables the transferring of knowledge from a complex model (LLMs), referred to as the teacher model, to a simpler counterpart known as the student model for deployment. Most previous approaches were adopting white-box distillation [77, 155, 164], which requires accessing the entire parameters of the LLM. Due to the arising of API-based LLM services (e.g., ChatGPT), black-box distilled models attract lots of attention, such as Alpaca [171], Vicuna [22], WizardLM [201], and so on [142, 233]. **Quantization** has emerged as a widely embraced technique to enable efficient representation of model weights and activations [44, 53, 123] by transforming traditional representation (floating-point numbers) to integers or other discrete forms. According to the timing of the quantization process, it can be categorized into post-training quantization (PTQ) [36, 123, 136] and quantization-aware training (QAT) [29, 84, 169]. **LowRank Factorization** [21, 70, 144] is a model compression technique that aims to approximate a given weight matrix by decomposing it into two or more smaller matrices with significantly lower dimensions. This method factorizes a weight matrix, W, into two matrices U and V, where $W \approx UV$, with U being $m \times k$ and V being $k \times n$, and k much smaller than m and n. This approximation significantly reduces parameters and computational overhead. In LLM research, lowrank factorization has been widely used for efficient finetuning, as seen in LORA [65] and its variants. TensorGPT [203], however, applies this concept differently by compressing LLMs for edge devices. It uses TensorTrain Decomposition (TTD) to store large embeddings in a lowrank tensor format, achieving up to 3840 times compression of the embedding layer while maintaining or improving model performance. For optimal LLM deployment on mobile devices, consider a multi-pronged approach combining pruning, knowledge distillation, quantization, and low-rank factorization techniques to significantly reduce model size and computational complexity while preserving performance.

Beyond model compression, the use of the LLM on mobile devices can be further improved through other inference optimizations, which involve *Parallel Computation*, *Memory Management*, *Request Scheduling*, *Kernel Optimization*, and *Software Frameworks*. **Parallel Computation** [12, 143, 160] leverages modern hardware's parallel processing capabilities to distribute computation across multiple cores or devices, substantially speeding up inference. It can be categorized into model parallelism [137, 143, 160] and decentralized inference [12, 13, 75], depending on the target object being distributed. **Memory Management** [90, 130, 163] refers to allocating, organizing, and efficiently utilizing the available memory resources on a mobile device. The Key-Value (KV) cache is a prime optimization target for autoregressive decoder-based models due to the memory-intensive nature of transformer architectures and the need for long-sequence inference [90, 158, 227]. **Request Scheduling** [5, 57, 139], similar to general ML serving techniques, aims to schedule incoming inference requests, optimize resource utilization, guarantee response time within latency service level objective (SLO), and effectively handle varying

request loads. Common aspects involve dynamic batching[5], preemption[57], priority [139], swapping [9], model selection [50], cost efficiency [217], load balancing and resource allocation [196]. **Kernel Optimization** [1, 159, 214] focuses on optimizing the individual operations or layers within the model by leveraging hardware-specific features and software techniques to accelerate critical computation kernels. Common aspects involve kernel fusion [198], tailored attention [95], sampling optimization [34], variable sequence length [214], and automatic compilation [81].

Software Frameworks [1, 2, 172] play a crucial role in inference optimization by encapsulating complex patterns, practices, and functionalities into reusable high-level APIs or automatic processes, providing abstractions to leverage various techniques for enhanced performance, scalability, and resource utilization. Integrating a **Deep Learning (DL) Compiler** into the framework further streamlines the optimization process with a unified environment for development, optimization, and deployment [106]. The DL compiler takes trained models as input and translates them into optimized code or instructions, often represented as multi-level intermediate representations (IRs), specifically tailored for target hardware platforms, such as CPUs, GPUs, TPUs, or other accelerators. It further applies various analyses and optimization techniques to achieve frontend and backend optimization, resulting in improved performance and efficiency during inference [18, 27, 92]. Recent research also offers emerging compiler-aided security hardening techniques to protect the compiled model code [20]. Overall, the synergy between software frameworks and DL compilers simplifies the development process, enabling automatic optimization, hardware adaptation, portability, interoperability, and enhanced performance. By incorporating various advanced techniques, software frameworks offer a pragmatic strategy for boosting inference performance, scalability, and resource utilization, facilitating the development, optimization, and deployment of LLM serving on mobile.

The optimization techniques described are not standalone solutions but are often used together to achieve the best on-device inference performance. Additionally, refining LLM inference involves balancing model accuracy with optimizing model size, computational demands, and overall performance, presenting a complex challenge that requires careful consideration. Beyond striving for efficiency, ensuring the security and protection of the model's intellectual property (IP) adds another layer of intricacy to the optimization efforts. These aspects, along with their implications for the optimization process, will be further discussed in the following on-device LLM security and LLM-Powered frameworks sections.

5 ON-DEVICE LLM SECURITY

DL techniques such as LLM are deeply engaged in human life. We can use them to revise the article, provide daily recommendations, write codes, and generate image or text content. The data collection required for cloud LLM presents obvious privacy issues. Users' personal, highly sensitive data have to be shared with computing servers [161]. This may cause sensitive information leakage or violate data protection laws [230, 231]. Therefore, deploying DL models directly on devices has gained popularity in recent years. However, recent studies show that on-device DL deployment also has serious security issues, especially for LLM. As such DL models are directly hosted on mobile systems, attackers can easily unpack the mobile Apps to obtain the deployed models [231]. Because the model weights are trained by a large amount of training data and have extremely high values [3, 39], deploying LLM on devices is a high-risk decision for developers. In addition, the internal information of on-device LLM can be considered a white box for attackers. Even if developers adopt some protections to resist parsing the model information, attackers still can locate the model information and reverse engineer the model details, *i.e.*, weights and structure [231]. Moreover, recent side-channel attacks and hardware fault injection attacks (e.g., Rowhammer attacks [135]) can also be used to exploit deep learning models, even in the advanced transformer architectures [150, 228]. For instance, it is shown that these system-level or hardware-level attacks can manipulate

the model outputs [63] by performing Rowhammer attacks to flip certain critical bits in the model weights. Moreover, with the help of queries to the model, attackers can also leak the model weights [149].

To protect the deployed DL models, especially for LLM, we now have two main methods to defend the on-device models: Trusted Execution Environments (TEE) and program protection. For the Trusted Execution Environments (TEE) [19, 94, 132, 133, 181, 235], it provides secured execution environment for on-device models. These methods design customized software or hardware architecture for protect the ownership of the deployed model, disable the access of unauthorized parties, and generate an encrypted model inference pipeline. These methods are effective in protecting the deployed model. However, they are hard to apply to various mobile platforms such as Android because they usually need specially designed software or hardware architectures. In addition, attackers are capable of using side-channel attacks to infer the model architectures [11, 149, 194, 195, 200, 210, 224].

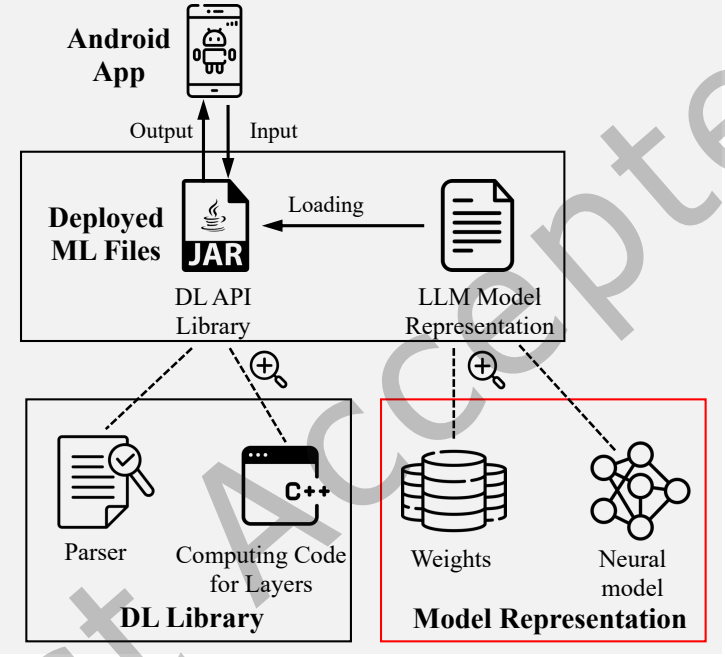

Fig. 2. The information leakage problem of on-device LLM on Android. The sensitive model representation is directly hosted on mobile devices.

To protect the LLM on various mobile systems and devices, model protection can be considered a special program protection problem. The general protection method for software such as obfuscation and optimization can also be applied to LLM on mobile. As shown in Figure 2, the security issue of on-device LLM is mainly caused by the exposure of the model representation (the red block of Figure 2). Attackers can reverse engineer the model representation that is packed in the deployed AI programs, *e.g.*, model files and API libraries, to steal the intellectual property [204, 218] or generate effective white-box attacks [68, 138, 216, 231, 232]. Therefore, minimizing the exposure of model representation can effectively protect the on-device LLM. To this end, Zhou *et al.* [230] adopt the idea of code obfuscation [25, 26, 156, 182, 197], which is a well-developed approach for hiding sensitive information in software, and propose to obfuscate the information of on-device ML models. Like the obfuscated code, the obfuscated on-device model contains hard-to-read information but still can be correctly run on mobile devices. It can significantly increase the difficulty of reverse engineering the deployed LLM. In

addition, a program refactorization scheme has been proposed to hide the explicit model representation on devices [229]. Unlike the other tools that only support limited number of model architectures and formats like *m2cgen*[2] and *llama.cpp*[3], it automatically trace the function call of model inference, extract the related codes, and refactor the code into an executable program. This scheme can applied to commonly used DL models such as LLM. The generated program does not have explicit model representation, *i.e.*, model weights and architecture. Attackers need to use human efforts to understand the compiled binary file to reverse engineer the deployed models. Accordingly, given the model becomes much obscure and hard to analyze, side channel attacks and hardware fault injection attacks are also hard to apply to the protected models to achieve high attack accuracies (e.g., the target critical model weights are hard to localize and manipulated) [151].

Overall, although defense strategies based on program protection can be applied to almost all mobile platforms, it is worth noting that these strategies cannot disable the reverse engineering of on-device LLM. Their goal is to significantly increase the cost of attackers, *i.e.*, using lots of human efforts to understand the binary program. The TEE-like defense methods are more suitable to be applied to high-value systems. In contrast, the program defense strategy can be applied to various Apps on various mobile OS.

6 PROVIDING LLM-POWERED FRAMEWORK APIS

The exploration of LLM-powered framework APIs for mobile app development is a vibrant and expanding field, focusing on streamlining the integration of advanced language models into mobile applications. This area of research is dedicated to the development, optimization, and deployment of APIs that enable mobile apps to leverage the capabilities of LLMs for a wide range of tasks, including natural language processing, conversational interfaces, and content generation.

Recent advancements have concentrated on creating accessible, efficient, and scalable solutions [40, 47, 91]. Frameworks are being developed to simplify the integration of LLMs into various applications, offering APIs that abstract away the complexities of direct interactions with LLMs. This makes it easier for developers to implement advanced language capabilities in their applications. Additionally, these frameworks are evolving to support more context-aware interactions, allowing LLMs to provide more relevant and personalized responses based on the user's context and previous interactions with the app [91].

Looking forward, the functionality and utility of LLM-powered framework APIs for mobile app development could be significantly enhanced through focused research in several key areas. The development of standardized API protocols promises to facilitate a more uniform development experience across different mobile operating systems and device types. Standardizing APIs could ensure that LLM-powered features are consistently available across the mobile ecosystem, catering to the diverse needs of developers and users alike.

Security is another critical area requiring attention. As the integration of LLMs into mobile apps increases, addressing the security implications of these APIs becomes imperative. Future research will need to explore ways to ensure secure data transmission between mobile devices and cloud servers, as well as secure on-device processing to minimize data exposure. This will be crucial in maintaining user trust and protecting sensitive information.

Energy efficiency is also a critical concern, given the limited battery life of mobile devices. Future directions should include research into mechanisms for minimizing the energy consumption of LLM-powered APIs. This could involve developing smarter caching strategies or optimizing the computational workload distribution between the device and the cloud, ensuring that mobile applications can deliver advanced functionalities without excessively draining battery life.

[2]https://github.com/BayesWitnesses/m2cgen
[3]https://github.com/ggerganov/llama.cpp

Additionally, the potential for LLM-powered APIs to support more interactive and multimodal inputs, such as combining text, voice, and visual inputs, opens up interesting new possibilities. This evolution could enable more natural and engaging user interactions with mobile applications, creating new possibilities for app design and functionality. Such advancements would not only enhance the user experience but also pave the way for innovative applications that fully exploit the capabilities of LLMs.

7 PROVIDING LLM-POWERED RUNTIME MONITORING

Further to the above directions, LLMs can be deployed to monitor the runtime behavior of mobile apps for various software engineering and security purposes. This is particularly important given the increasing complexity of mobile apps and the potential security threats they face; for instance, mobile apps can be attacked to leak sensitive user information, disrupt services, or even compromise the mobile device. Nevertheless, offline analysis and testing of mobile apps' behavior may be likely insufficient to detect and prevent all those runtime attacks. From this perspective, we envision that LLMs can be deployed in mobile devices to monitor the runtime behavior of mobile apps, the mobile frameworks, and even the mobile operating system (OS) itself for various software engineering and security purposes.

Offering Intelligent Runtime Analysis. LLMs have demonstrated state-of-the-art performance in a wide range of natural language and code processing tasks. In particular, it is shown that LLMs can reason real-world software artifacts and other complex scenarios, given that they have been trained on large-scale corpora which often subsume common sense knowledge and programming expertise. With the high reasoning capability, we envision that LLMs can be deployed to monitor the runtime behavior of mobile apps to facilitate various software engineering tasks, such as profiling, debugging, and performance optimization. Furthermore, given that possible attacks can be launched against mobile apps and even mobile frameworks, we see that LLMs can be deployed to monitor and reason the runtime behavior and recognize potential security threats. To enhance the intelligence of LLMs in analyzing those collected information, we envision that LLMs can be fine-tuned with relevant trace datasets to better reason the runtime behavior of mobile apps; we also expect LLMs to incorporate domain-specific knowledge of common security threats encountered by mobile apps. Prompt engineering techniques like chain-of-trust can also be adopted in this context. Overall, we see the high potential of LLMs to behave as a "smart" runtime analysis system for mobile apps, which can provide insights into the runtime behavior of mobile apps and the mobile system and outperforms traditional runtime analysis tools.

Offering Privacy-Preserving Runtime Analysis. To facilitate app vendors to continuously analyze the released mobile apps, the common practice is that mobile apps generate runtime logs (e.g., crash reports and traces) and upload them to remote servers for further analysis. This practice is widely used in real-world scenarios, yet it raises privacy concerns as the logs may contain sensitive user information. In fact, recent studies have shown the potential privacy risks of logs and traces generated by mobile apps, which can leak sensitive user information like doctor appointments [61]. While some privacy-preserving techniques have been proposed to sanitize logs and traces before uploading them to remote servers [61, 191], they essentially undermine the utility of logs and traces for further analysis. Moreover, the mainstream approaches rely on differential privacy techniques, which only offer limited privacy guarantees and may not be sufficient to protect group users' privacy and confidentiality. While some advanced techniques like secure multi-party computation (MPC) and anonymized transmissions may be used to enable remote vendor analysis without leaking sensitive information, they are often computationally expensive and impose a high requirement on the computing resources on mobile devices. From this perspective, we believe that with LLMs deployed in mobile, app logs can be analyzed for most cases without leaking sensitive information to the remote vendor servers. This offers a principled way to protect user privacy; before releasing the mobile app, the app vendor can configure the LLMs in the mobile such that the LLMs can better analyze the logs locally to decide performance issues or security threats. LLMs can analyze the raw logs to decide performance

issues or security threats and query the remote vendor servers only when necessary to obtain further insights. This way, the sensitive information in the raw logs will not be leaked to the remote vendor servers, and the user privacy will be protected.

Design Considerations. To facilitate such demanding runtime analysis, we expect to conduct the following tasks. On one hand, this requires the mobile apps and mobile system components under protection to provide proper logs and introspection interfaces. LLMs can hook the provided interfaces to capture the runtime behavior of mobile apps, and even the mobile frameworks and the mobile OS. Interestingly, instead of forming a "passive" runtime analysis system where LLMs wait for logs and traces to be generated, we envision that LLMs can be trained to actively interact with mobile apps and the system software to perform investigation. For instance, once the LLM detects a potential security threat, it can interact with the mobile app to further confirm the threat and then decide to take corresponding actions like alerting the user or even terminating the app. This shall offer a more proactive and efficient runtime analysis system for mobile apps. On the other hand, we anticipate the demand of fine-tuning LLMs for such security tasks. Our tentative exploration shows that mainstream LLMs available on the market are not sufficiently trained with software trace data, which is crucial for runtime analysis. Therefore, we advocate the community to provide relevant datasets to support LLM fine-tuning and customization for runtime monitoring and analysis tasks.

Recent research has illustrated the high feasibility of using LLMs in relevant fields [76, 110]; this indicates the high potential of using LLMs for mobile runtime analysis for software engineering and security purposes. However, there still exist several challenges to be addressed in the context of mobile. For instance, we see the demand of augmenting the LLMs' response time to avoid noticeable delays in mobile devices. More importantly, we envision the need to ensure the LLMs' robustness against even privileged adversaries with access to the device or the LLM model itself. One may also need to consider the potential "memorization" issues of LLMs, which may lead to cross-app privacy leakage when malicious apps are installed on the same device and exploit the LLMs' memorization capabilities. We believe that addressing these challenges will pave the way for deploying LLMs in mobile devices for runtime analysis tasks.

8 DISCUSSION

LLM has been demonstrated to be useful in many software engineering fields. In this work, we are specifically interested in adopting LLMs in the mobile field, which not only inherits many of the benefits (and challenges) from the general SE field but also brings new challenges (i.e., mobile-specific concerns) to the community. Indeed, LLMs targeting the mobile field may need to be improved with dedicated MSE (Mobile Software Engineering) data. The distinction between MSE data and SE (Software Engineering) data is primarily rooted in their application contexts and processing requirements. SE data encompasses a broad spectrum of software development tasks across various platforms, including codebases, bug reports, and version histories. It is typically collected from open-source repositories and industrial environments, with pre-processing focused on enhancing generalizability for tasks such as code comprehension and bug detection. In contrast, MSE data is meticulously curated to address the specific constraints and requirements of mobile software engineering scenarios. Representative MSE data includes apps' GUI pages, UI transition graphs, energy consumption, runtime logs, usages of permissions or sensitive APIs, reviews, etc. Integrating Retrieval-Augmented Generation (RAG) into the preparation of datasets offers a novel way to enhance model performance and adaptability. RAG enables real-time access to and retrieval of relevant data, improving the relevance and accuracy of generated outputs by leveraging the most contextually appropriate information. Devices like mobile phones, tablets, and IoT devices generate diverse data—user interaction logs, sensor data, application traces, and system logs—that can be crucial for LLMs. By using RAG, researchers and developers can dynamically incorporate this device-specific data during the generation process without extensive pre-processing of static datasets. For instance, in SE, logs and runtime traces can enhance the model's ability to

generate context-relevant code snippets or debug suggestions. In UX, real-time user data can drive personalized recommendations or interface adjustments. Additionally, RAG facilitates the integration of various data types in multi-modal processing, ensuring outputs remain accurate and contextually relevant. This approach bridges the gap between static dataset preparation and the dynamic needs of modern applications, leading to more intelligent, responsive, and context-aware solutions.

When we have the LLMs ready, there are still various mobile-specific challenges that need to be addressed. First, it is extremely challenging to reduce their size to be suitable for on-device deployment without compromising performance. Mobile devices only have limited computational resources compared with online servers. Normally the powerful model compress technologies will highly impact the model performance. Second, when putting LLM directly on mobile devices, even if the LLM runs very well, it may unavoidably impact the normal execution of the system (or other mobile apps) because large resources such as memory are constantly occupied by the LLM. Third, putting LLM on devices could also increase the attack surface. The most significant concern is how to defend against reverse-engineering attacks. It is relatively easy for attackers to physically access mobile devices and subsequently, attackers could extract and reverse-engineer the deployed LLMs to steal intellectual property or produce effective white-box attacks. In contrast, attackers can only use black-box methods to attack the cloud LLMs, which is far less efficient than the white-box attacks. Therefore, for on-device LLMs, we need to increase the on-device model security to a similar level as the cloud models.

Except for the mobile-specific concerns mentioned above, the LLMs applied in the mobile field will unfortunately also suffer from problems faced by normal LLMs. Indeed, taking hallucination as an example, it has been extensively studied within the field of natural language processing (NLP) [223] that is hard to mitigate. Hallucination refers to the phenomenon where the model generates information that is either factually incorrect, fabricated, or misleading, and it also impacts the use of LLMs in mobile software engineering, especially for mobile app development and testing [33]. In mobile software development, hallucinations can manifest when LLMs are utilized to assist developers in generating code, documentation, or design patterns [115, 148]. For instance, when an LLM generates incorrect code snippets or suggests inappropriate libraries or frameworks, it can lead to the integration of faulty or suboptimal components into the mobile application. Such errors might not be immediately apparent, particularly in the early stages of development, but they can result in significant technical debt, decreased app performance, or security vulnerabilities that are difficult to trace and resolve later in the development process. Moreover, LLMs are increasingly being used in the generation of automated test cases and in the identification of potential bugs or security issues within mobile applications. Hallucinations in this context can lead to the creation of test cases that are based on incorrect assumptions or that target non-existent functionality [38, 49]. This can result in a false sense of security, where developers believe that certain aspects of the application have been thoroughly tested when, in reality, critical issues remain undetected. Additionally, misleading or fabricated bug reports generated by an LLM can divert resources away from addressing real issues, thus compromising the overall quality and reliability of the mobile software.

9 RELATED WORK

The rapid advancement of Large Language Models (LLMs) has led to their widespread application across various domains, from natural language processing to mobile and edge computing environments. Through a comprehensive collection of surveys and literature reviews on LLMs, we first provide a concise overview of the evolution and development of LLMs. Subsequently, we investigate the challenges and security issues associated with applying LLMs to mobile devices, along with examining current examples of LLM deployments on mobile platforms.

Evolution and Development of Large Language Models. The evolution of Large Language Models (LLMs) began with early statistical models like n-grams, which struggled with data sparsity [55]. Neural language models, such as RNNs [114], introduced word embeddings to improve sequence prediction [225]. Pre-trained models like

BERT [28] marked a shift, enabling training on large datasets followed by task-specific fine-tuning [131]. The introduction of transformers, particularly in GPT models [14, 146, 147], leveraged self-attention mechanisms, significantly enhancing scalability and performance [131]. Modern LLMs, such as LLaMA [175, 176] and PaLM [24], now contain billions of parameters, pushing the boundaries of language understanding across domains [64]. Much of the initial focus was on optimizing these models for high-performance computing platforms, with more recent attention shifting to adapting LLMs for mobile devices, where computational resources are limited. The development, capabilities, and adaptation of LLMs provide a foundation for their integration into mobile platforms.

Challenges in Adapting LLMs for Mobile Devices. One of the key challenges in deploying LLMs on mobile devices is balancing model size, performance, and efficiency. LLMs are computationally intensive, often requiring substantial hardware to process large-scale data in real time, making them difficult to implement directly on resource-constrained mobile platforms. To address this, recent advancements in model compression techniques, such as quantization, pruning [21], knowledge distillation [62], and low-rank adaptation (LoRA) [65], have enabled a reduction in model size without sacrificing performance. Techniques such as Mixture of Experts (MoE) [37] further improve efficiency by selectively activating parts of the model, reducing the computational load. These innovations are crucial for making LLMs viable on mobile devices, where power consumption, latency, and computation capacity are critical concerns.

LLM Deployment and Security on Mobile Devices. Deploying LLMs on mobile devices also introduces potential security risks, such as reverse engineering and data leakage. To mitigate these risks, the literature suggests implementing techniques like Trusted Execution Environments (TEE) [153] and code obfuscation [25, 26, 156, 182, 197] to protect the integrity of the models. These approaches ensure that LLMs can be deployed securely on mobile devices, particularly when dealing with sensitive user data. Future research should continue to explore how to optimize LLM performance on mobile devices while maintaining data security and privacy.

Current Works on Deploying LLMs on Mobile Devices. Recent advancements in large language models (LLMs) have led to significant progress in optimizing them for mobile platforms, overcoming challenges such as computational constraints, inference speed, and privacy concerns. OpenELM[129], through its layer-wise scaling, optimizes parameter allocation for mobile environments. Apple Foundation Models (AFM)[51] enhances inference efficiency using grouped-query attention and quantization for privacy-centric applications. MobileLLM[124] offers a "deep and thin" architecture suitable for hardware-constrained devices, excelling in tasks like zero-shot reasoning. Qwen2.5[173, 205], trained on extensive datasets, provides domain-specific capabilities with models ranging from 0.5B to 72B parameters, optimizing multilingual and long-text generation tasks, thus paving the way for practical, efficient LLM deployment on mobile devices.

10 CONCLUSION

In this position paper, we have motivated the strong necessity to apply LLMs for the mobile ecosystem and subsequently provided an initial roadmap for our fellow researchers to achieve that objective. In the roadmap, we summarized six directions that we believe are urgently required to be researched, including (1) preparing more datasets, (2) Addressing MSE tasks, (3) Serving LLM on mobile (4) Enhancing the security of on-device LLMs, (5) facilitating intelligent app development through LLM-powered framework APIs, and (6) providing LLM-powered runtime monitoring. We acknowledge to the community that, these six directions should not be considered as representative to the whole space of applying LLMs for mobile. We would like to invite our fellow researchers to help in identifying more research gaps that need to be filled in order to achieve intelligent user experiences.

REFERENCES

[1] 2020. NVIDIA Effective Transformer. Online. https://github.com/NVIDIA/FasterTransformer Commit: df4a753 Accessed: 2023-11-25.
[2] 2023. FlexGen. Online. https://github.com/FMInference/FlexGen Commit: d34f7b4, Accessed on: 2023-11-25.

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

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

[5] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: Machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[6] Saud Alotaibi, Abdulrahman Alruban, Moneerah N. Alotaibi, Ali Alshumrani, and Abdulaziz Altamimi. 2019. A New User-Apps Interactions Dataset for Behavioral Profiling Using Smartphones. *Journal of Internet Technology and Secured Transactions* (2019). https://api.semanticscholar.org/CorpusID:195253758

[7] Avinash Anand, Kritarth Prasad, Ujjwal Goel, Mohit Gupta, Naman Lal, Astha Verma, and Rajiv Ratn Shah. 2023. Context-enhanced language models for generating multi-paper citations. In *International Conference on Big Data Analytics*. Springer, 80–94.

[8] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 1–18.

[9] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. 2020. {PipeSwitch}: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 499–514.

[10] Patrick Bareiß, Beatriz Souza, Marcelo d'Amorim, and Michael Pradel. 2022. Code generation tools (almost) for free? a study of few-shot, pre-trained language models on code. *arXiv preprint arXiv:2206.01335* (2022).

[11] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*. 515–532.

[12] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. 2022. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188* (2022).

[13] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. 2024. Distributed Inference and Fine-tuning of Large Language Models Over The Internet. *Advances in Neural Information Processing Systems* 36 (2024).

[14] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[15] Haipeng Cai, Ziyi Zhang, Li Li, and Xiaoqin Fu. 2019. A Large-Scale Study of Application Incompatibilities in Android. In *The 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*.

[16] Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez. 2023. Improving code generation by training with natural language feedback. *arXiv preprint arXiv:2303.16749* (2023).

[17] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[18] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[19] Yu Chen, Fang Luo, Tong Li, Tao Xiang, Zheli Liu, and Jin Li. 2020. A training-integrity privacy-preserving federated learning scheme with trusted execution environment. *Information Sciences* 522 (2020), 69–79.

[20] Yanzuo Chen, Yuanyuan Yuan, and Shuai Wang. 2023. OBSan: An Out-Of-Bound Sanitizer to Harden DNN Executables.. In *NDSS*.

[21] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).

[22] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)* 2, 3 (2023), 6.

[23] Muslim Chochlov, Gul Aftab Ahmed, James Vincent Patten, Guoxian Lu, Wei Hou, David Gregg, and Jim Buckley. 2022. Using a Nearest-Neighbour, BERT-Based Approach for Scalable Clone Detection. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 582–591.

[24] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[25] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. A taxonomy of obfuscating transformations.

[26] Christian Collberg, Clark Thomborson, and Douglas Low. 1998. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 184–196. https://doi.org/10.1145/268946.268962

[27] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*

3 (2021), 800–811.
[28] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[29] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence* 5, 3 (2023), 220–235.
[30] Shihan Dou, Junjie Shan, Haoxiang Jia, Wenhao Deng, Zhiheng Xi, Wei He, Yueming Wu, Tao Gui, Yang Liu, and Xuanjing Huang. 2023. Towards Understanding the Capability of Large Language Models on Code Clone Detection: A Survey. *arXiv preprint arXiv:2308.01191* (2023).
[31] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).
[32] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. Automated handling of anaphoric ambiguity in requirements: a multi-solution study. In *Proceedings of the 44th International Conference on Software Engineering*. 187–199.
[33] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.
[34] Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833* (2018).
[35] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16091–16101.
[36] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun. 2020. Post-training piecewise linear quantization for deep neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 69–86.
[37] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.
[38] Robert Feldt, Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Towards autonomous testing agents via conversational large language models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1688–1693.
[39] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines* 30 (2020), 681–694.
[40] FlowiseAI [n. d.]. FlowiseAI - Build LLM Apps Easily. https://flowiseai.com/. Accessed: 2024-03-20.
[41] Jun Gao, Pingfan Kong, Li Li, Tegawendé F Bissyandé, and Jacques Klein. 2019. Negative Results on Mining Crypto-API Usage Rules in Android Apps. In *The 16th International Conference on Mining Software Repositories (MSR 2019)*.
[42] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
[43] Lobna Ghadhab, Ilyes Jenhani, Mohamed Wiem Mkaouer, and Montassar Ben Messaoud. 2021. Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. *Information and Software Technology* 135 (2021), 106566.
[44] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*. Chapman and Hall/CRC, 291–326.
[45] Henry Gilbert, Michael Sandborn, Douglas C Schmidt, Jesse Spencer-Smith, and Jules White. 2023. Semantic Compression With Large Language Models. *arXiv preprint arXiv:2304.12512* (2023).
[46] Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307* (2020).
[47] GradientJ [n. d.]. GradientJ - Everything you need to build LLM Native Applications. https://www.gradientj.com/. Accessed: 2024-03-20.
[48] Andrea; Mercaldo Francesco; Visaggio Corrado A; Canfora Gerardo; Panichella Sebastiano Grano, Giovanni; Di Sorbo. 2017. Software Applications User Reviews. https://huggingface.co/datasets/sealuzh/app_reviews
[49] Siqi Gu, Chunrong Fang, Quanjun Zhang, Fangyuan Tian, and Zhenyu Chen. 2024. TestART: Improving LLM-based Unit Test via Co-evolution of Automated Generation and Repair Iteration. *arXiv preprint arXiv:2408.03095* (2024).
[50] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. 2022. Cocktail: A multidimensional optimization for model serving in cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1041–1057.
[51] Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, et al. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075* (2024).
[52] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[53] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.
[54] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964* (2020).
[55] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints* (2023).
[56] Aamir Hamid, Hemanth Reddy Samidi, Tim Finin, Primal Pappachan, and Roberto Yus. 2023. A Study of the Landscape of Privacy Policies of Smart Devices. arXiv:2308.05890 [cs.CY]
[57] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale preemption for concurrent {GPU-accelerated}{DNN} inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 539–558.
[58] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).
[59] Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608* (2024).
[60] Yu Hao, Weiteng Chen, Ziqiao Zhou, and Weidong Cui. 2023. E&V: Prompting Large Language Models to Perform Static Analysis by Pseudo-code Execution and Verification. arXiv:2312.08477 [cs.SE]
[61] Yu Hao, Sufian Latif, Hailong Zhang, Raef Bassily, and Atanas Rountev. 2021. Differential privacy for coverage analysis of software traces. *Leibniz international proceedings in informatics* 194 (2021).
[62] Geoffrey Hinton. 2015. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531* (2015).
[63] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. 2019. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. 497–514.
[64] Xinying Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John C. Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. *ArXiv* abs/2308.10620 (2023). https://api.semanticscholar.org/CorpusID:261048648
[65] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
[66] Yangyu Hu, Haoyu Wang, Yajin Zhou, Yao Guo, Li Li, Bingxuan Luo, and Fangren Xu. 2019. Dating with Scambots: Understanding the Ecosystem of Fraudulent Dating Applications. *IEEE Transactions on Dependable and Secure Computing (TDSC)* (2019).
[67] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. https://doi.org/10.1145/3238147.3238191
[68] Yujin Huang and Chunyang Chen. 2022. Smart app attack: hacking deep learning models in android apps. *IEEE Transactions on Information Forensics and Security* 17 (2022), 1827–1840. https://doi.org/10.1109/tifs.2022.3172213
[69] Yuchao Huang, Junjie Wang, Zhe Liu, Yawen Wang, Song Wang, Chunyang Chen, Yuanzhe Hu, and Qing Wang. 2024. Crashtranslator: Automatically reproducing mobile application crashes directly from stack trace. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.
[70] Yerlan Idelbayev and Miguel A Carreira-Perpinán. 2020. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8049–8059.
[71] Akshath Jain, David Rodriguez, Jose M. del Alamo, and Norman Sadeh. 2023. ATLAS: Automatically Detecting Discrepancies Between Privacy Policies and Privacy Labels. arXiv:2306.09247 [cs.CR]
[72] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403* (2024).
[73] Nan Jiang, Chengxiao Wang, Kevin Liu, Xiangzhe Xu, Lin Tan, and Xiangyu Zhang. 2023. Nova$^+$: Generative Language Models for Binaries. *arXiv preprint arXiv:2311.13721* (2023).
[74] Shuyang Jiang, Yuhao Wang, and Yu Wang. 2023. SelfEvolve: A Code Evolution Framework via Large Language Models. *arXiv preprint arXiv:2306.02907* (2023).
[75] Youhe Jiang, Ran Yan, Xiaozhe Yao, Beidi Chen, and Binhang Yuan. 2023. Hexgen: Generative inference of foundation model over heterogeneous decentralized environment. *arXiv preprint arXiv:2311.11514* (2023).
[76] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R. Lyu. 2024. LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. arXiv:2310.01796 [cs.SE]
[77] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).

[78] Sungmin Kang, Juyeon Yoon, Nargiz Askarbekkyzy, and Shin Yoo. 2023. Evaluating Diverse Large Language Models for Automatic and General Bug Reproduction. *arXiv preprint arXiv:2311.04532* (2023).
[79] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2022. Large language models are few-shot testers: Exploring llm-based general bug reproduction. *arXiv preprint arXiv:2209.11515* (2022).
[80] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring llm-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2312–2323.
[81] Navdeep Katel, Vivek Khandelwal, and Uday Bondhugula. 2022. MLIR-based code generation for GPU tensor cores. In *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction*. 117–128.
[82] Mohamad Khajezade, Jie JW Wu, Fatemeh Hendijani Fard, Gema Rodríguez-Pérez, and Mohamed Sami Shehata. 2024. Investigating the Efficacy of Large Language Models for Code Clone Detection. arXiv:2401.13802 [cs.SE]
[83] Adam Khakhar, Stephen Mell, and Osbert Bastani. 2023. PAC prediction sets for large language models of code. In *International Conference on Machine Learning*. PMLR, 16237–16249.
[84] Minsoo Kim, Sihwa Lee, Sukjin Hong, Du-Seong Chang, and Jungwook Choi. 2022. Understanding and improving knowledge distillation for quantization-aware training of large transformer encoders. *arXiv preprint arXiv:2211.11014* (2022).
[85] Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947* (2016).
[86] Takashi Koide, Naoki Fukushi, Hiroki Nakano, and Daiki Chiba. 2023. Detecting phishing sites using chatgpt. *arXiv preprint arXiv:2306.05816* (2023).
[87] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. 2023. Better zero-shot reasoning with role-play prompting. *arXiv preprint arXiv:2308.07702* (2023).
[88] Pingfan Kong, Li Li, Jun Gao, Kui Liu, Tegawendé F Bissyandé, and Jacques Klein. 2018. Automated Testing of Android Apps: A Systematic Literature Review. *IEEE Transactions on Reliability* (2018).
[89] Bonan Kou, Muhao Chen, and Tianyi Zhang. 2023. Automated Summarization of Stack Overflow Posts. *arXiv preprint arXiv:2305.16680* (2023).
[90] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 611–626.
[91] LangChain [n. d.]. LangChain: Applications that can reason. Powered by LangChain. https://www.langchain.com/. Accessed: 2024-03-20.
[92] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2020. MLIR: A compiler infrastructure for the end of Moore's law. *arXiv preprint arXiv:2002.11054* (2020).
[93] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems* 2 (1989).
[94] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.
[95] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, et al. 2022. xformers: A modular and hackable transformer modelling library.
[96] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).
[97] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).
[98] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick Mcdaniel. 2015. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps. In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*.
[99] Li Li, Tegawendé F Bissyandé, and Jacques Klein. 2019. Rebooting Research on Detecting Repackaged Android Apps: Literature Review and Benchmark. *IEEE Transactions on Software Engineering (TSE)* (2019).
[100] Li Li, Tegawendé F Bissyandé, Damien Octeau, and Jacques Klein. 2016. DroidRA: Taming Reflection to Support Whole-Program Analysis of Android Apps. In *The 2016 International Symposium on Software Testing and Analysis (ISSTA 2016)*.
[101] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Yves Le Traon. 2017. Static Analysis of Android Apps: A Systematic Literature Review. *Information and Software Technology* (2017).
[102] Li Li, Tegawendé F Bissyandé, Haoyu Wang, and Jacques Klein. 2018. CiD: Automating the Detection of API-related Compatibility Issues in Android Apps. In *The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*.
[103] Li Li, Jun Gao, Tegawendé F Bissyandé, Lei Ma, Xin Xia, and Jacques Klein. 2020. CDA: Characterising Deprecated Android APIs. *Empirical Software Engineering (EMSE)* (2020).
[104] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting. *IEEE Transactions on Information Forensics & Security*

(TIFS) (2017).
[105] Lingwei Li, Li Yang, Huaxi Jiang, Jun Yan, Tiejian Luo, Zihan Hua, Geng Liang, and Chun Zuo. 2022. AUGER: automatically generating review comments with pre-training models. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1009–1021.
[106] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. 2020. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 708–727.
[107] Tsz-On Li, Wenxi Zong, Yibo Wang, Haoye Tian, Ying Wang, and Shing-Chi Cheung. 2023. Finding Failure-Inducing Test Cases with ChatGPT. *arXiv preprint arXiv:2304.11686* (2023).
[108] Tsz-On Li, Wenxi Zong, Yibo Wang, Haoye Tian, Ying Wang, Shing-Chi Cheung, and Jeff Kramer. 2023. Nuances are the key: Unlocking chatgpt to find failure-inducing tests with differential prompting. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 14–26.
[109] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190* (2021).
[110] Yichen Li, Yintong Huo, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, and Michael R. Lyu. 2023. Exploring the Effectiveness of LLMs in Automated Logging Generation: An Empirical Study. arXiv:2307.05950 [cs.SE]
[111] Zongjie Li, Chaozheng Wang, Zhibo Liu, Haoxuan Wang, Shuai Wang, and Cuiyun Gao. 2022. CCTEST: Testing and Repairing Code Completion Systems. *arXiv preprint arXiv:2208.08289* (2022).
[112] Soo Ling Lim. 2014. Worldwide Mobile App User Behavior Dataset. https://doi.org/10.7910/DVN/27459
[113] Feng Lin, Dong Jae Kim, et al. 2024. When llm-based code generation meets the software development process. *arXiv preprint arXiv:2403.15852* (2024).
[114] Zachary Chase Lipton. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv Preprint, CoRR, abs/1506.00019* (2015).
[115] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, and Li Zhang. 2024. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971* (2024).
[116] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems* 35 (2022), 1950–1965.
[117] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210* (2023).
[118] Xiao Liu, Heyan Huang, Ge Shi, and Bo Wang. 2022. Dynamic prefix-tuning for generative template-based event extraction. *arXiv preprint arXiv:2205.06166* (2022).
[119] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. 2022. Deep Learning for Android Malware Defenses: a Systematic Literature Review. *ACM Computing Surveys (CSUR)* (2022).
[120] Yue Liu, Chakkrit Tantithamthavorn, Yonghui Liu, and Li Li. 2024. On the Reliability and Explainability of Language Models for Program Generation. *ACM Transactions on Software Engineering and Methodology* (2024).
[121] Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. 2022. Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing. arXiv:2212.04732 [cs.SE]
[122] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*. PMLR, 22137–22176.
[123] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. 2021. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems* 34 (2021), 28092–28103.
[124] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. 2024. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905* (2024).
[125] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 647–658.
[126] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. 2023. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782* (2023).
[127] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489* (2021).
[128] Antonio Mastropaolo, Luca Pascarella, and Gabriele Bavota. 2022. Using deep learning to generate complete log statements. In *Proceedings of the 44th International Conference on Software Engineering*. 2279–2290.

[129] Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Seyed Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, et al. 2024. Openelm: An efficient language model family with open training and inference framework. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*.

[130] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781* (2023).

[131] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).

[132] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*. 94–108.

[133] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 161–174.

[134] Ambarish Moharil and Arpit Sharma. 2022. Identification of intra-domain ambiguity using transformer-based machine learning. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*. 51–58.

[135] Onur Mutlu and Jeremie S Kim. 2019. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2019), 1555–1571.

[136] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*. PMLR, 7197–7206.

[137] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*. PMLR, 7937–7947.

[138] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 739–753.

[139] Kelvin KW Ng, Henri Maxime Demoulin, and Vincent Liu. 2023. Paella: Low-latency Model Serving with Software-defined GPU Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 595–610.

[140] Humphrey Obie, Waqar Hussain, Xin Xia, John Grundy, Li Li, Burak Turhan, Jon Whittle, and Mojtaba Shahin. 2021. A First Look at Human Values-Violation in App Reviews. In *The 43rd ACM/IEEE International Conference on Software Engineering, SEIS Track (ICSE-SEIS 2021)*.

[141] Shidong Pan, Zhen Tao, Thong Hoang, Dawen Zhang, Zhenchang Xing, Xiwei Xu, Mark Staples, and David Lo. 2023. SeePrivacy: Automated Contextual Privacy Policy Generation for Mobile Applications. arXiv:2307.01691 [cs.CR]

[142] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277* (2023).

[143] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023).

[144] Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. 2018. Semi-orthogonal low-rank matrix factorization for deep neural networks.. In *Interspeech*. 3743–3747.

[145] Rohith Pudari and Neil A Ernst. 2023. From Copilot to Pilot: Towards AI Supported Software Development. *arXiv preprint arXiv:2303.04142* (2023).

[146] Alec Radford. 2018. Improving language understanding by generative pre-training. (2018).

[147] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[148] Mirza Masfiqur Rahman and Ashish Kundu. 2024. Code Hallucination. *arXiv preprint arXiv:2407.04831* (2024).

[149] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. 2022. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1157–1174. https://doi.org/10.1109/sp46214.2022.9833743

[150] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2020. Tbt: Targeted neural network attack with bit trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13198–13207.

[151] Ashay Rane, Calvin Lin, and Mohit Tiwari. 2015. Raccoon: Closing digital {Side-Channels} through obfuscated execution. In *24th USENIX Security Symposium (USENIX Security 15)*. 431–446.

[152] Prakhar Rathi. 2019. Google Play Store Reviews. https://www.kaggle.com/datasets/prakharrathi25/google-play-store-reviews

[153] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, Vol. 1. IEEE, 57–64.

[154] Jordan Samhi, Li Li, Tegawendé F Bissyandé, and Jacques Klein. 2022. Difuzer: Uncovering Suspicious Hidden Sensitive Operations in Android Apps. In *The 44th International Conference on Software Engineering (ICSE 2022)*.

[155] Victor Sanh, L Debut, J Chaumond, and T Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. arXiv 2019. *arXiv preprint arXiv:1910.01108* (2019).
[156] Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, and Edgar Weippl. 2016. Protecting software through obfuscation: Can it keep pace with progress in code analysis? *ACM Computing Surveys (CSUR)* 49, 1 (2016), 1–37. https://doi.org/10.1145/2886012
[157] Oussama Ben Sghaier and Houari Sahraoui. 2023. A Multi-Step Learning Approach to Assist Code Review. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 450–460.
[158] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285* (2023).
[159] Yining Shi, Zhi Yang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Ziming Miao, Yuxiao Guo, Fan Yang, and Lidong Zhou. 2023. Welder: Scheduling deep learning memory access via tile-graph. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 701–718.
[160] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).
[161] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1310–1321.
[162] Mohammed Latif Siddiq, Joanna Santos, Ridwanul Hasan Tanvir, Noshin Ulfat, Fahmid Al Rifat, and Vinicius Carvalho Lopes. 2023. Exploring the Effectiveness of Large Language Models in Generating Unit Tests. *arXiv preprint arXiv:2305.00418* (2023).
[163] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. Powerinfer: Fast large language model serving with a consumer-grade gpu. *arXiv preprint arXiv:2312.12456* (2023).
[164] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355* (2019).
[165] Tiezhu Sun, Kevin Allix, Kisub Kim, Xin Zhou, Dongsun Kim, David Lo, Tegawendé F. Bissyandé, and Jacques Klein. 2023. DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode. arXiv:2212.05976 [cs.SE]
[166] Xiaoyu Sun, Xiao Chen, Li Li, Haipeng Cai, John Grundy, Jordan Samhi, Tegawendé F. Bissyandé, and Jacques Klein. 2022. Demystifying Hidden Sensitive Operations in Android apps. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2022).
[167] Xiaoyu Sun, Xiao Chen, Yanjie Zhao, Pei Liu, John Grundy, and Li Li. 2022. Mining Android API Usage to Generate Unit Test Cases for Pinpointing Compatibility Issues. In *The 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022)*.
[168] Zhensu Sun, Li Li, Yan Liu, Xiaoning Du, and Li Li. [n. d.]. On the Importance of Building High-quality Training Datasets for Neural Code Search. ([n. d.]).
[169] Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. 2020. Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000* (2020).
[170] Feiyang Tang and Bjarte M. Østvold. 2023. User Interaction Data in Apps: Comparing Policy Claims to Implementations. arXiv:2312.02710 [cs.SE]
[171] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.
[172] MLC team. 2023. MLC-LLM. Online. https://github.com/mlc-ai/mlc-llm Commit: 3358029, Accessed on: 2023-11-25.
[173] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/
[174] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938* (2023).
[175] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[176] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[177] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *Proceedings of the 44th International Conference on Software Engineering*. 2291–2302.
[178] Frederick Tung and Greg Mori. 2019. Similarity-preserving knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1365–1374.
[179] Tim Valicenti, Justice Vidal, and Ritik Patnaik. 2023. Mini-gpts: Efficient large language models through contextual pruning. *arXiv preprint arXiv:2312.12682* (2023).
[180] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M McCune. 2012. Trustworthy execution on mobile devices: What security properties can my mobile platform give me?. In *Trust and Trustworthy Computing: 5th International Conference, TRUST 2012, Vienna, Austria, June 13-15, 2012. Proceedings 5*. Springer, 159–178.

[181] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229* (2020).
[182] Chenxi Wang. 2001. *A security architecture for survivability mechanisms*. University of Virginia.
[183] Haoyu Wang, Hao Li, Li Li, Yao Guo, and Guoai Xu. 2018. Why are Android Apps Removed From Google Play? A Large-scale Empirical Study. In *The 15th International Conference on Mining Software Repositories (MSR 2018)*.
[184] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets. In *The 2018 Internet Measurement Conference (IMC 2018)*.
[185] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing with Large Language Models: Survey, Landscape, and Vision. arXiv:2307.07221 [cs.SE]
[186] Jian Wang, Shangqing Liu, Xiaofei Xie, and Yi Li. 2023. Evaluating AIGC detectors on code content. *arXiv preprint arXiv:2304.05193* (2023).
[187] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).
[188] Shuo Wang, Jing Li, Zibo Zhao, Dongze Lian, Binbin Huang, Xiaomei Wang, Zhengxin Li, and Shenghua Gao. 2024. Tsp-transformer: Task-specific prompts boosted transformer for holistic scene understanding. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 925–934.
[189] Yawen Wang, Lin Shi, Mingyang Li, Qing Wang, and Yun Yang. 2020. A deep context-wise method for coreference detection in natural language requirements. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 180–191.
[190] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.
[191] Zhaoyu Wang, Pingchuan Ma, Huaijin Wang, and Shuai Wang. 2024. PP-CSA: Practical Privacy-Preserving Software Call Stack Analysis. In *OOPSLA*.
[192] Zekun Moore Wang, Zhongyuan Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhan Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Man Zhang, et al. 2023. Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models. *arXiv preprint arXiv:2310.00746* (2023).
[193] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[194] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 125–137. https://doi.org/10.1109/dsn48063.2020.00031
[195] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. 2018. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 393–406.
[196] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. {MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 945–960.
[197] Gregory Wroblewski. 2002. General method of program code obfuscation. (2002).
[198] Kaixin Wu, Bojie Hu, and Qi Ju. 2021. TenTrans High-Performance Inference Toolkit for WMT2021 Efficiency Task. In *Proceedings of the Sixth Conference on Machine Translation*. 795–798.
[199] Yonghao Wu, Zheng Li, Jie M Zhang, Mike Papadakis, Mark Harman, and Yong Liu. 2023. Large language models in fault localisation. *arXiv preprint arXiv:2308.15276* (2023).
[200] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. 2020. Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs* 67, 11 (2020), 2717–2721.
[201] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).
[202] Huatao Xu, Liying Han, Qirui Yang, Mo Li, and Mani Srivastava. 2024. Penetrative AI: Making llms comprehend the physical world. In *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*. 1–7.
[203] Mingxue Xu, Yao Lei Xu, and Danilo P Mandic. 2023. Tensorgpt: Efficient compression of the embedding layer in llms based on the tensor-train decomposition. *arXiv preprint arXiv:2307.00526* (2023).
[204] Mingfu Xue, Yushu Zhang, Jian Wang, and Weiqiang Liu. 2021. Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations. *IEEE Transactions on Artificial Intelligence* 3, 6 (2021), 908–923.
[205] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671* (2024).
[206] Aidan ZH Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. 2024. Large language models for test-free fault localization. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[207] Chengran Yang, Bowen Xu, Junaed Younus Khan, Gias Uddin, Donggyun Han, Zhou Yang, and David Lo. 2022. Aspect-based api review classification: How far can pre-trained transformer model go?. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 385–395.

[208] Hao Yang, Min Zhang, and Daimeng Wei. 2024. IRAG: Iterative Retrieval Augmented Generation for SLU. In *CSPA*. IEEE, 30–34.

[209] Juyeon Yoon, Robert Feldt, and Shin Yoo. 2023. Autonomous Large Language Model Agents Enabling Intent-Driven Mobile GUI Testing. arXiv:2311.08649 [cs.SE]

[210] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. 2020. Deepem: Deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 209–218.

[211] Ye Yuan, Chengwu Liu, Jingyang Yuan, Gongbo Sun, Siqi Li, and Ming Zhang. 2024. A Hybrid RAG System with Comprehensive Enhancement on Complex Reasoning. *arXiv preprint arXiv:2408.05141* (2024).

[212] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. 2023. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. *arXiv preprint arXiv:2305.04207* (2023).

[213] Zhengran Zeng, Hanzhuo Tan, Haotian Zhang, Jing Li, Yuqun Zhang, and Lingming Zhang. 2022. An extensive study on pre-trained models for program understanding and generation. In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*. 39–51.

[214] Yujia Zhai, Chengquan Jiang, Leyuan Wang, Xiaoying Jia, Shang Zhang, Zizhong Chen, Xin Liu, and Yibo Zhu. 2023. Bytetransformer: A high-performance transformer boosted for variable-length inputs. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 344–355.

[215] Xian Zhan, Tianming Liu, Lingling Fan, Li Li, Sen Chen, Xiapu Luo, and Yang Liu. 2021. Research on Third-Party Libraries in Android Apps: A Taxonomy and Systematic Literature Review. *IEEE Transactions on Software Engineering* (2021).

[216] Chaoning Zhang, Philipp Benz, Adil Karjauv, Jae Won Cho, Kang Zhang, and In So Kweon. 2022. Investigating Top-k White-Box and Transferable Black-box Attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15085–15094.

[217] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. {MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1049–1062.

[218] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia conference on computer and communications security*. 159–172.

[219] Jingxuan Zhang, Siyuan Liu, Lina Gong, Haoxiang Zhang, Zhiqiu Huang, and He Jiang. 2022. Beqain: An effective and efficient identifier normalization approach with bert and the question answering system. *IEEE Transactions on Software Engineering* 49, 4 (2022), 2597–2620.

[220] Jiyang Zhang, Sheena Panthaplackel, Pengyu Nie, Junyi Jessy Li, and Milos Gligoric. 2022. Coditt5: Pretraining for source code and natural language editing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[221] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792* (2023).

[222] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European conference on computer vision (ECCV)*. 184–199.

[223] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).

[224] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. 2021. Stealing neural network structure through remote FPGA side-channel analysis. *IEEE Transactions on Information Forensics and Security* 16 (2021), 4377–4388.

[225] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[226] Yanjie Zhao, Li Li, Kui Liu, and John Grundy. 2022. Towards Automatically Repairing Compatibility Issues in Published Android Apps. In *The 44th International Conference on Software Engineering (ICSE 2022)*.

[227] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2023. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104* (2023).

[228] Mengxin Zheng, Qian Lou, and Lei Jiang. 2023. Trojvit: Trojan insertion in vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4025–4034.

[229] Mingyi Zhou, Xiang Gao, Pei Liu, John Grundy, Chunyang Chen, Xiao Chen, and Li Li. 2024. Model-less Is the Best Model: Generating Pure Code Implementations to Replace On-Device DL Models. arXiv:2403.16479 [cs.SE]

[230] Mingyi Zhou, Xiang Gao, Jing Wu, John Grundy, Xiao Chen, Chunyang Chen, and Li Li. 2023. Modelobfuscator: Obfuscating model information to protect deployed ml-based systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1005–1017.

[231] Mingyi Zhou, Xiang Gao, Jing Wu, Kui Liu, Hailong Sun, and Li Li. 2024. Investigating White-Box Attacks for On-Device Models. In *ICSE*. 1–12.
[232] Mingyi Zhou, Jing Wu, Yipeng Liu, Shuaicheng Liu, and Ce Zhu. 2020. Dast: Data-free substitute training for adversarial attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 234–243. https://doi.org/10.1109/cvpr42600.2020.00031
[233] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592* (2023).
[234] Wenhan Zhu, Sebastian Proksch, Daniel M. German, Michael W. Godfrey, Li Li, and Shane McIntosh. 2024. What is an app store? The software engineering perspective. *Empirical Software Engineering* 29, 1 (Jan. 2024). https://doi.org/10.1007/s10664-023-10362-3
[235] Pengfei Zuo, Yu Hua, Ling Liang, Xinfeng Xie, Xing Hu, and Yuan Xie. 2021. Sealing neural network models in encrypted deep learning accelerators. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1255–1260.