

LLMs in Mobile Apps: Practices, Challenges, and Opportunities

Kimberly Hau
University of Toronto
kimberly.hau@mail.utoronto.ca

Safwat Hassan
University of Toronto
safwat.hassan@utoronto.ca

Shurui Zhou
University of Toronto
shuruiz@ece.utoronto.ca

Abstract—The integration of AI techniques has become increasingly popular in software development, enhancing performance, usability, and the availability of intelligent features. With the rise of large language models (LLMs) and generative AI, developers now have access to a wealth of high-quality open-source models and APIs from closed-source providers, enabling easier experimentation and integration of LLMs into various systems. This has also opened new possibilities in mobile application (app) development, allowing for more personalized and intelligent apps. However, integrating LLM into mobile apps might present unique challenges for developers, particularly regarding mobile device constraints, API management, and code infrastructure. In this project, we constructed a comprehensive dataset of 149 LLM-enabled Android apps and conducted an exploratory analysis to understand how LLMs are deployed and used within mobile apps. This analysis highlights key characteristics of the dataset, prevalent integration strategies, and common challenges developers face. Our findings provide valuable insights for future research and tooling development aimed at enhancing LLM-enabled mobile apps.

Index Terms—Large Language Model, LLM, Mobile Application, Android Application, Open-source Application, AI-enabled Software, ML-enabled Software

I. INTRODUCTION

Recent years have not only witnessed significant advancements in natural language processing (NLP) tasks, but also launched large language models (LLMs) into recognition beyond its niche in NLP. Since then, the area has been moving fast [1], with LLMs showing potential toward becoming the basic building block for general-purpose artificial intelligence (AI) agents, and generative AI. Characterised as transformer-based language models with over a billion parameters and pre-trained on large Web-scale text corpus, it has been found that scaling up both the size of training data and parameter count of LLMs increases the capability of the models to approximate human-level performance in various tasks [2].

As such, LLMs are now frequently integrated into various types of software to enhance functionality across a wide array of applications [1], [3]–[6]. Similarly, mobile applications (apps) are increasingly incorporating LLM components, bringing advanced AI capabilities directly to users’ devices. However, deploying LLMs to mobile devices presents significant challenges. Prior studies have explored the technical aspects of the integration of AI models into products, and observed that the resource-constrained restrictions of mobile devices make LLMs especially difficult to deploy

locally to mobile apps [7]. As studied by Alizadeh et al., the unprecedented capabilities of LLMs come with substantial computational and memory requirements for inference [8]. Also, Çöplü et al. investigated the performance of LLMs on various smartphones and concluded that further advancements in power management and system integration are required for achieving sustained performance on even top-of-the-line mobile devices [9]. This is corroborated by Murthy et al., who benchmarked the performance of LLMs on on-device use cases and concluded that current LLMs require significant resources in terms of CPU and RAM usage when deployed on mobile devices [10]. In addition to deploying LLMs on-device, another option for LLM deployment is vendor-provided APIs of commercial LLMs, such as Google’s Gemini [11] and OpenAI’s GPT [12], which allows users the ability to run inference on closed-source LLMs without requiring memory and computational costs of storing and running the model locally. However, as observed by Minaee et al., this comes with the costs of maintaining subscriptions to the LLM vendor, latency lags, and security concerns with transmitting information [1].

Overall, there remains a need for an examination of the methods commonly employed by practitioners to integrate LLMs into Android apps, along with an analysis of the distinct challenges associated with each approach. Therefore, in this study, we aim to provide insight into the practices, opportunities, and challenges that surround the current deployment of LLMs to Android apps. This includes investigating characteristics of LLM-enabled apps, integration strategies for merging LLMs with traditional code, handling the challenges of maintenance and updating apps in time with LLM updates, and other common concerns with deploying LLMs to mobile apps. For this project, we define **LLM-enabled mobile apps** as mobile apps that contain one or more LLM components and use such components to provide functionalities. We compiled a detailed dataset comprising 149 Android apps available on GitHub, each incorporating at least one LLM component, and then asked the following three research questions (RQs):

RQ1: What are the characteristics of LLM-enabled apps?

Method: For each Android app, we leveraged the GitHub API [13] to gather metrics such as the number of stars, commits, issues, and contributors. We then manually inspected each app to collect the LLM-related information, such as the types and their vendors. From this data, we summarized common traits across the dataset as well as outliers.

Results: We analyzed and interpreted the distribution of the metrics above, finding that there is little difference in value between the first 2 quartiles, indicating that a large percentage of apps are smaller projects, but there are severe outliers. For example, the median star count is 8, but the most starred app in the dataset has 58,129. Finally, we classify the app functionalities across the dataset and discover that the majority of apps' main functionality is chatbots, followed by other uses of AI APIs such as text and image generation.

RQ2: What main integration strategies are used to adopt LLMs into apps?

Method: We downloaded and qualitatively analyzed the source code of each app to identify the different methods used by developers to integrate LLMs into apps. We also provided descriptive statistics for releases, issues, LLM-related releases, and the app sizes associated with each strategy.

Results: We identified five integration strategies (IS) that are utilized across the dataset to call LLMs as described below. We find that 132 out of 149 apps integrate LLMs through the first strategy, and apps that download the LLM either on the end-device or a back-end server require fewer updates to LLM-related code, compared to strategies that do not.

- **IS1:** Using third-party APIs provided by LLM vendors.
- **IS2:** Using a third-party website and calling LLMs as a webview app.
- **IS3:** Calling LLMs hosted on a back-end server.
- **IS4:** Hosting LLMs on the user end-device.
- **IS5:** A hybrid strategy that utilizes two out of the four integration strategies.

RQ3: What drives developers to update LLM-related code?

Method: We manually identified the code used to interact with LLMs for each app then collected the number of commits to these code files as well as the commit messages. In order to understand the intention behind these updates, we fed the scrambled set of all commit messages into ChatGPT [12] to identify the top ten most common topics across these updates. In addition, we also observe the distribution of topics across each integration strategy.

Results: We observe that the top three most common topics for LLM-related code updates are *Feature Additions*, *Refactoring and Code Cleanup*, and *Version Updates*.

To sum up, our study makes the following contributions: (1) We present a novel dataset of Android apps with LLM integration, (2) We summarize the characteristics of this dataset and the strategies used to integrate LLMs into the apps, and (3) We analyze the commit messages of LLM-related code updates to obtain insight into what drives developers to update LLM-related code. Our findings shed light on how LLMs have thus far been applied to mobile apps, and the challenges and opportunities that come with it. The entire dataset can be found in the replication package [14].

II. RELATED WORK

A. Studies on ML-enabled Systems

With the popularity of machine learning (ML) increasing over recent years, there has been an increasing number of products incorporating ML models into their products [1], [3]–[6], [15]. However, adding ML models to existing systems require additional infrastructure and consideration of system architecture [5], [16]. As well, developers must consider system requirements and user concerns and experience, which poses challenges on how to best embed the ML model [15], [17]. The most commonly named problems include data quality, meshing ML models with traditional software parts, tool support for managing ML products, and quality assurance [15], [16]. Overall, we still know very little about how ML models are embedded and integrated into ML-enabled systems [15].

As a new subset of ML models, there is a similar mystery surrounding how LLM models are being integrated into LLM-enabled systems. Also, the findings from prior work focussing on general systems may not be directly applicable to mobile apps, as mobile apps are subject to more continuous, user-motivated, maintenance [18], and different non-functional constraints (e.g. energy consumption) [19]. We aim to focus on a subset of LLM-enabled systems by presenting an empirical study on 149 LLM-enabled Android apps and the strategies used to integrate LLMs.

B. On-Device LLM Deployment

One method to integrate LLMs into a system is to deploy them on-device. Deploying LLMs on device is becoming a research hotspot as it increases the potential applications and eliminates the costs of cloud deployment, such as security and internet connection [2]. However, there are restrictions on limited hardware performance, memory bandwidth, and storage capacity. LLMs have many parameters and have large memory requirements to store the model parameters, the model activations, and the gradients and corresponding statistics [2].

To facilitate on-device deployment, LLM-specific inference engines have been developed, such as Llama.cpp [20], MLC-LLM [21], or Langchain [22]. These engines are specifically designed for transformer-based LLM deployment on CPUs and GPUs. In this study, we aim to investigate the use of these engines in implementing LLMs in mobile apps, and how they are incorporated into the rest of the system.

C. Third-Party Libraries in Mobile Apps

A large number of companies develop software by way of Application Programming Interfaces (APIs), also known as third-party libraries, which allow the reuse of existing code components [23]. These libraries are widely used in mobile apps, accounting for, on average, over 60% of Android app code as well as being used in almost every popular open-source Android app project [23], [24]. When these companies issue updates to these libraries, app developers have the choice to adopt these implementations, but may be deterred due to the effort or effects of upgrading these libraries [25]. Salza et al. conducted an empirical investigation into the when, why,

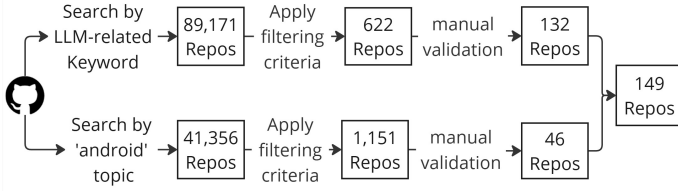


Fig. 1. Data collection overview.

and how mobile app developers update third-party libraries in code and found that only 15.52% of library uses are constantly updated by developers, most of the updates are done with the aim of avoiding bug propagation or making an app compatible with the Android releases, and some app developers do not update libraries due to low payoff or to not break existing code [26]. We aim to investigate the particular usage of LLM-related APIs, and whether they are a large factor in-app updates and new releases.

Considering the numerous factors contributing to the uncertainty and challenges associated with implementing LLMs—including the requisite architectures, infrastructures, and best practices—we aim to address key research gaps through an analysis of Android apps that have integrated LLMs. This study seeks to identify prevalent use cases, challenges, and integration strategies related to incorporating LLMs, as well as to characterize the features of such apps. By leveraging these findings and metrics, we endeavor to answer critical research questions regarding the implementation of LLMs in Android apps, ultimately offering insights to inform and facilitate the integration of LLMs in mobile apps.

III. DATA COLLECTION

Our study focused exclusively on open-source Android apps with LLM integrations. Open-source projects provide access to development practices typically unavailable in closed-source software, such as code architecture, commit history, and testing procedures. We began with a list of 33 LLM-related keywords obtained from model repositories and scholarly articles, covering widely-used LLMs, frameworks specific to LLMs, and companies focused on LLM technology [27]–[33], listed in Tab. I. We utilized the GitHub API [13] to collect a list of repositories based on two distinct search criteria as presented in Fig. 1.

TABLE I
THE LLM-RELATED KEYWORDS USED FOR DATA COLLECTION.

Category	Keywords for Data Collection
Company	openai [34], cohere [35]
Framework	langchain [22], mlc llm [21], ollama [27], llama [20]
Term	LLM, large language model
Model	alexatm [36], claude [37], deepseek [38], falcon [39], gemini [11], gemma [40], gpt [12], LaMDA [41], marcoroni [42], minichat [43], mistral [44], mixtral [45], nyxene [46], orca [47], PaLM [48], phi-1.5 [49], phi-3 [50], qwen [51], redpajama [52], shining valiant [53], stable beluga [54], stablelm [55], starling [56], una-xaberi [57], vicuna [58]

1) *Search criterion 1: Matching keywords in repository names, descriptions, or issue/PR discussions:* We collected all repositories that had one or more of the keywords in their name or description, with at least one star and one fork. We also collected all repositories with one or more stars and forks with at least one issue or pull request (PR) that contained one or more of the keywords. We specified the creation date of the issues or PRs to be after the release date of the corresponding LLM, or within the year 2022 for LLM-related terms. This ensures that the search results focus on discussions, updates, or contributions that are relevant to the deployment or utilization of the LLM in question, reflecting more current and contextually appropriate data. This resulted in a list of 89,170 unique repositories.

Following the collection of repositories, we filtered repositories that match the following exclusion criteria (EC):

- **EC1: Projects without a Readme file:** We excluded all repositories that did not have a ReadMe file. This was performed to remove toy projects, as well as to prevent toy projects that are not well maintained from evading the next step. This reduced the number of repositories to 81,955.
- **EC2: Projects that are either deprecated or not Android apps:** We discarded all repositories that had at least one of the exclusion keywords listed in Tab. II present in the ReadMe file. This was done to remove repositories that were deprecated, archived, or obvious non-product repositories, using exclusion terms such as “tutorial,” “framework,” and “obsolete.” These exclusion keywords were taken from Nahar et al.’s work [17], which utilized a similar filtering process for extracting ML products. 24,144 repositories were remaining following this step.
- **EC3: Projects that do not have an AndroidManifest.xml or lack activities in the AndroidManifest.xml file:** App developers need to include `AndroidManifest.xml` file that specifies the main settings of the app, such as required permissions and the minimum Android API version. In our study, repositories lacking an `AndroidManifest.xml` file were excluded, as its absence indicates the repository does not represent a genuine Android app. This filtering reduced the number of repositories to 670. Subsequently, repositories containing one or more `AndroidManifest.xml` files were further examined to ensure at least one activity was defined in the manifest, resulting in the exclusion of 48 repositories and leaving a total of 622.

TABLE II
THE EXCLUSION KEYWORDS USED FOR FILTERING REPOSITORIES.

deprecated, obsolete, framework, library, testing, toolkit, example, sample, guideline, guide, tutorial, blog, book, libraries, toolchain, interview notes, curated collection
--

Manual validation. Following these automatic filtrations, the 622 repositories remaining were subject to manual

examination for LLM-relation. Manual validation was employed for high accuracy in the end dataset. Each repository was examined for some LLM involvement, such as the use of REST APIs for closed-source LLMs, LLM-specific frameworks, or discussion of LLM use in issues or pull requests. Repositories were discarded for having no LLM-use or being archived. The most common reason for repositories being discarded was being flagged for LLM-related keywords that did not refer to LLMs in the context that it was flagged for. This was most common for ubiquitous terms such as ‘palm’, ‘falcon’ and ‘orca’, as well as ‘large language model’, as the GitHub API did not maintain the order of terms. Some repositories were also flagged for keywords such as ‘GPT’ and ‘Claude’ from issues, in which users asked or referenced GPT when having questions or writing code. A few projects were categorised as ‘unknown’ for whether they had LLM use, typically due to being mono-repositories or containing multiple projects, which contained both LLM-use and Android apps, but had unclear overlap.

After categorising each repository, we filtered all non-English repositories. This resulted in 518 English repositories from the previous 622. Of the 518 English repositories, 132 were true positives, 340 were false positives, and 46 were categorised as ‘unknown’ or ‘other’.

2) *Search criterion 2: Searching for Android repositories:* We performed a second data collection process to collect repositories that did not have LLM mention in their title issues/PRs but used LLMs in their code. In this process, we used the GitHub API to gather all repositories that had been added to the ‘android’ topic on GitHub, with one or more stars and forks. Following this, we cloned each repository and applied our exclusion criteria (EC1 - EC3). This step reduced our pool of candidates from 41,356 repositories to 12,063. We then searched each repository for the presence of any of our LLM-related keywords in the code or title, which further reduced our candidates to 6,678 repositories. Due to the commonality of our keywords, we applied another filtration step from this pool by searching each repository for the presence of any keyword, preceded by either a space or a special character. The result of this step was 1,151 repositories, which we performed manual validation on to find 46 repositories with LLM use, of which 17 were not also found in the previous process. This brought our final dataset to 149 repositories. From this dataset, we performed an initial data exploration into project characteristics.

IV. RQ1: WHAT ARE THE CHARACTERISTICS OF LLM-RELATED APPS?

A. Methodology

To determine the characteristics of the dataset, we gathered repository metadata related to the popularity and relevance of projects, including the counts for contributors, commits, and issues for each app using the GitHub API [13]. Additionally, we used cloc [59] to count blank lines, comment lines, and physical lines of source code in a code base, to determine the software size in lines of code (LOC) per app. We then

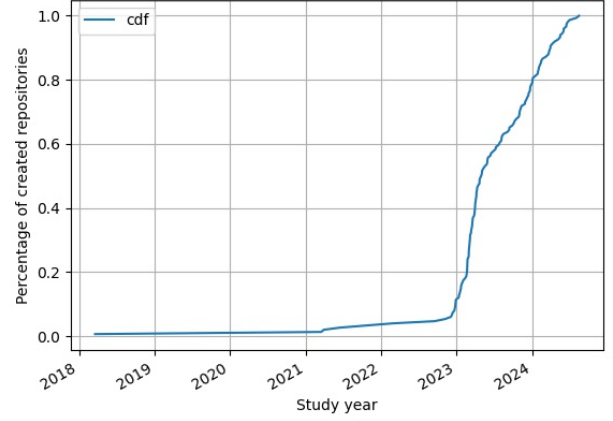


Fig. 2. Cumulative distributive function of creation time across the dataset of 149 Android apps.

TABLE III
VENDOR, LLMs PER VENDOR, AND THE NUMBER OF APPS PER VENDOR.

Vendor	LLMs	# of apps
Google	Gemini, PaLM, flan-t5	41
Anthropic	Claude	5
Meta	Llama, BART	6
Openai	GPT, Whisper, Dall-E	111
Mistral	mistral, mixtral	7
Perplexity AI	perplexity	1
Ollama	ollama	1
Huggingface	huggingchat	1
Open-source vendors	local model	4

manually investigated each app to log which LLMs were used per app, whether it was open- or closed-source, as well as used locally or with an API.

Finally, we classify the functionalities offered by the apps of this dataset. For the 132 of 149 apps that had a repository description, we retrieved the descriptions used and submitted them to OpenAI’s GPT- 4o-mini using a paid OpenAI API subscription, set to zero temperature [12]. We first asked the model to determine the top 5 functionalities presented across all repository descriptions in a zero-shot prompt and to provide a description of each functionality. We then prompted the model with another zero-shot prompt that provided the list of repository descriptions and the top 5 functionalities and their descriptions and asked the model to assign one functionality to each app. Finally, for the 17 apps that did not have a repository description, we manually categorized them by title and ReadMe file. The results of these prompts are shown in Tab. VI.

B. Results

The distribution for the collected metrics is shown in Fig. IV-A and also presented in Tab. V. Fig. 2 shows the cumulative distribution of the creation date of repositories, with the earliest repository (“YoubiMiku”) being created in 2018. We also present the top 10 repositories by star count

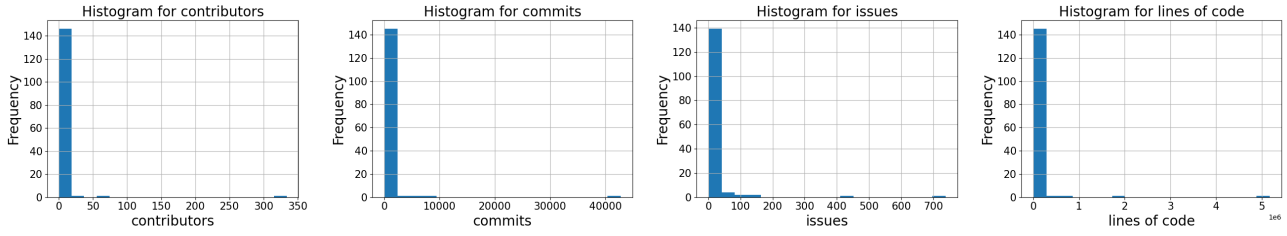


Fig. 3. Histograms displaying the distribution of several characteristics.

TABLE IV
AN OVERVIEW OF THE TOP TEN REPOSITORIES SORTED BY STAR COUNT.

App	#Stars	#Commits	#Contribors	#Issues	Google Play
AppFlowy	58,129	6,294	334	436	Yes
Maid	1,485	1,885	22	158	Yes
Futo Keyboard	775	42,801	62	739	Yes
GPT Assistant	667	44	1	51	No
chatAir	527	573	15	47	Yes
Flutter ChatGPT	461	19	6	9	No
OtakuWorld	446	2,176	5	11	No
GPT Mobile	413	377	4	38	Yes
ChatGPT App	337	19	2	22	No
SpeakGPT	291	179	2	128	Yes

in Tab. IV. We also observe a large range of values for the number of commits (spanning from 19 to 42,801) and contributors (1 to 334) of these 10 apps. Notably, six out of the 10 apps are also available on Google Play Store. Tab. III displays the overall distribution of LLMs used in this dataset, and their vendors.

Contributor Distribution. Our results show that the average number of contributors is 5 with a median of 1 contributor. The minimum, first quartile, and second quartile values for the number of contributors are all 1, indicating that at least half of the apps are maintained by a single contributor. The third quartile shows only a modest increase, reaching 2 contributors. However, the dataset contains a significant outlier: the app “AppFlowy” [60], which has 334 contributors, far exceeding the third quartile value. As described in the project Readme file, this app offers a variety of LLMs via APIs, including GPT, Claude, Llama, and Mistral, as well as locally hosted models, to power their search functionality.

In contrast, the majority of apps (101 out of 149) have only 1 contributor. These apps exhibit limited activity, with median values of 0 issues, 0 releases, 2 stars, and 34 commits. The remaining 48 apps with more than one contributor demonstrate higher engagement with a median of 21.5 stars and 21.5 commits, though their median number of issues and releases remains the same as those maintained by a single contributor.

Commit size. Although the average number of commits is 501, only 25% of the projects have more than 60 commits, and 50% have fewer than 20 commits. There is an average of 7 days per commit across all apps, with a median of 1 day per commit. Notably, “Futo Keyboard” app [61] stands out with 42,801 commits, significantly exceeding the dataset’s general trends. “Futo Keyboard” is a customizable keyboard app leveraging local LLMs Whisper and Llama, and is among

the more popular apps, with 775 stars and 62 contributors. The minimum number of commits is recorded for the app “WizGPT” [62], which was created on May 4, 2023, with only one commit labeled as “initial commit” since then. “WizGPT” was planned to be a GPT and Dall-E powered voice assistant that can be run on Android, iOS, Web, and Desktop based on the Readme file. “WizGPT” has 6 stars and 0 releases and issues.

Number of issues. The dataset shows a median of 0 issues per project, with an average of 16 issues. Half of the projects have no reported issues, and three-quarters have 4 or fewer. The app “Futo Keyboard” [61] significantly skews the upper range with 739 issues, which inflates the average despite most projects exhibiting minimal or no issues. A total of 82 apps, representing the majority, have 0 issues. These projects typically exhibit minimal activity, with median values of 0 releases, 4 stars, and 11 commits. In contrast, apps with at least 1 issue demonstrate higher engagement, with median values of 1 release, 24 stars, and 38 commits.

Sizes of the apps. The median size of the collected apps, measured in LOC, is 3,838, with an average size of 70,360 LOC. The average LOC is nearly twenty times the median, indicating a skewed distribution with some exceptionally large apps. The app with the highest LOC, at 5,176,508, is *chatAir* [63], a native Android app for chatting with GPT, Claude, and Gemini. As one of the most prominent apps in the dataset, *chatAir* boasts 485 stars and ranks above the upper quartile for issues, contributors, and commits. Its LOC count is over five times the dataset’s average and represents the largest size among all projects. The app also has the most tags in the dataset, reflecting its active development. Conversely, the app with the smallest LOC is *OldGPT* [64], a lightweight chat app designed for users with outdated browsers or operating systems. *OldGPT* integrates GPT through the OpenAI API and has minimal activity, with just 1 star, 3 releases, and no reported issues.

App functionalities. As shown in Tab. VI, the most common functionality seen across the repository descriptions is Chatbot. Over half of the dataset (56%) is categorized into this functionality. Conversely, the least common functionality is Task Management and Productivity Tools, with only 6 apps categorized into this functionality. Examining the descriptions of each functionality, we can see that all functionalities mention AI, indicating that the majority of apps use LLMs to provide a large part of their service.

TABLE V

A SUMMARY OF THE DISTRIBUTION OF QUANTITATIVE MEASUREMENTS TAKEN ON THE DATASET. Q1–FIRST QUANTILE, Q3–THIRD QUANTILE.

Category	Min.	Q1	Median	Q3	Max.	Mean
#Stars	1	3	8	27	58,129	449.4
#Commits	1	7	22	61.0	42,801	500.7
#Releases	0	0	0	1	88	4.6
Project age (days)	0	4	64	293	2,432	199
#Contributors	1	1	1	2	334	4.7
#Issues	0	0	0	4	739	15.7
KLOC	0.2	2.2	3.8	9.1	5176.5	70.3
Avg. days/commit	0.0	0.3	1.3	4.5	195.3	6.8

V. RQ2: WHAT MAIN INTEGRATION STRATEGIES ARE USED TO ADOPT LLMs INTO APPS?

A. Methodology

To identify the strategies employed for integrating LLMs within the dataset, we analyzed the source code for each app by extracting and inspecting files containing code responsible for importing or invoking LLM functionalities. Similar methods of model invocation were categorized into distinct integration strategies, for which we computed median metrics, including frequency of usage, number of releases, reported issues, and lines of code. Furthermore, we calculated the median proportion of LLM-related releases by pinpointing commits that modified files containing LLM-related code. We then counted the number of releases that followed one or more such commits since the prior release. LLM-related code was defined as code-facilitating tasks such as selecting, switching, downloading, or invoking LLMs, including elements such as API URLs, model identifiers, or prompt templates.

B. Results

Through qualitative analysis, we identified five distinct strategies employed by developers to integrate LLMs into their workflows. Tab. VII provides an overview of the key characteristics associated with each strategy. These strategies are described in detail below.

1) Strategy 1: Using Third-party API Calls: This strategy is by far the most common across the dataset, with 132 apps in our dataset using this integration strategy.

Description: This strategy (shown in Fig. 4) allows developers to access popular closed-source LLMs (e.g., GPT and Gemini) using an API key. As shown in Fig. 5, developers set up the model parameters (e.g., select the used model, define the required temperature, and the maximum output token size). Then, the mobile app calls the third-party model using an API key for authentication. Given this strategy’s popularity, packages are available from both vendors and third parties that can help implement it. Two-thirds (87) of this strategy’s apps call directly the third-party API using HTTP requests.

Our analysis reveals that 45 apps employ mediator libraries, which serve as intermediaries to facilitate integration with

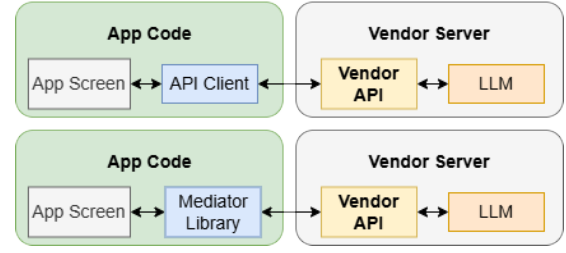


Fig. 4. An overview of the two methods for calling LLMs using Third-party APIs.

third-party APIs. These mediator libraries can be categorized into two distinct types:

- 1) Custom mediators that facilitate the connection to a single vendor’s API:** In this type, the third-party API providers (e.g., Google, OpenAI) design the mediator library to facilitate the connection to their API. For example, as shown in Fig. 5, “*AI Study Assistant*” app imports Google’s official generative library to access the GenerativeModel object for calling Gemini. We find 21 apps that adopt custom mediators to integrate LLMs into their code.
- 2) Generic mediators that facilitate the connection to multiple vendors’ API:** In this type, a generic mediator library that facilitates the connection to multiple LLM APIs. For example, “*ChatAir*” app imports the theokanning.openai package that facilitates a connection to GPT, Gemini, and Claude. We found 24 apps that adopt generic mediators to integrate LLMs into their code.

Insights and Discussion: Consistent with the overall dataset, the subset of apps utilizing third-party API calls exhibits a right-skewed distribution across most metrics. The median values for both releases and issues in this subset are zero. However, for apps with at least one release, the median number of releases is 2.5, with 66.7% of these releases involving modifications to LLM-related code. These findings highlight the necessity for developers to actively maintain and update LLM-related code as part of their app development process.

```

suspend fun generateText(
    prompt: String,
    key: String,
): String? {
    val generativeModel = GenerativeModel(
        modelName = "gemini-1.0-pro-latest",
        apiKey = key,
        generationConfig = generationConfig {
            temperature = temp
            maxOutputTokens = 2048
            candidateCount = candidates
        },
        safetySettings = safetySettings
    )
    return generativeModel.generateContent(prompt).text
}

```

Fig. 5. An example of integrating Gemini using Google API [65].

2) Strategy 2: Using Third-party Website: Six apps use a web view wrapper to access third-party LLMs.

TABLE VI
APP FUNCTIONALITIES ACROSS THE DATASET.

Functionality	# (%)	Description	E.g., App and Description
Chatbot	84 (56%)	Apps feature a chatbot interface that allows users to engage in natural language conversations with AI models like ChatGPT or Gemini. This functionality enables users to ask questions, seek information, or have casual conversations, providing a personalized and interactive experience.	ChatGemini - this app is a multiplatform chatbot app powered by Gemini.
Integration with LLM APIs	27 (18%)	Apps utilize APIs from OpenAI or other AI providers to access advanced features such as text generation, image creation, and summarization. This integration allows developers to leverage powerful AI models to enhance the app's capabilities and provide users with sophisticated functionalities.	Sense - this app's description describes itself as an OpenAI client with ChatGPT support.
Content Generation	19 (13%)	Apps are designed to generate content based on user input, such as recipes, stories, or summaries of videos and documents. This functionality allows users to create personalized content quickly and efficiently, often using AI to enhance creativity and reduce effort.	AI Study Assistant - this app aims to enhance learning experiences by offering lecture summarization, essay writing, and question generation.
Voice Interaction	13 (9%)	Apps incorporate voice recognition and synthesis capabilities, allowing users to interact with the AI using voice commands. This functionality enhances accessibility and convenience, enabling hands-free operation and real-time voice responses.	SpeakGPT - this app is described as a personal voice assistant, based on ChatGPT.
Task Management and Productivity Tools	6 (4%)	Apps focus on productivity by offering features like task management, to-do lists, and journaling. These apps often use AI to help users generate tasks, track their mood, or summarize information, making it easier to manage daily activities and improve mental well-being.	ToDo list with ChatGPT - this app's functionality is facilitating to-do lists by generating tasks using ChatGPT.

TABLE VII
A SUMMARY OF THE DISTRIBUTION OF QUANTITATIVE MEASUREMENTS BY INTEGRATION STRATEGY.

Integration Strategy	Frequency	Median #releases	Median #issues	Median %LLM-related releases	Median #LOC
IS1: Using third-party API calls	132	0.0	0.0	66.7%	2,929.0
IS2: Using third-party website	6	6.5	6.5	83.3%	916.5
IS3: Hosting LLMs on back-end servers	5	1.0	1.0	0.0%	6,519.0
IS4: Hosting LLMs on user devices	2	4.0	371.5	7.1%	26,3723.5
IS5: Mixing multiple integration strategies	4	21.0	100.5	40.3%	61,890.5

Description: Vendors (e.g., ChatGPT [12] and HuggingChat [66]) can allow users to access their LLMs via their webpage. As shown in Fig. 6, apps can facilitate sending prompts and receiving responses by implementing a webview client to load the Vendors' webpage URL. This strategy avoids calling an API to access closed-source LLMs (shown in Fig. 7). However, it may offer reduced capabilities compared to those provided by APIs, due to rate limits placed by the vendor [67].

Insights and Discussion: We find that five apps access the ChatGPT website [12] and one app accesses HuggingChat [66], a chat interface provided by Huggingface that offers a list of LLMs to chat with. This strategy had the highest median percentage of LLM-related releases (83.8%). When examining the code updates and commit messages associated with these releases, we observed that developers frequently tweaked code surrounding the web client. This may be because this strategy is not an officially provided integration method, so developers may need to work around changes to the vendor websites, such as updating URLs or domains.

3) **Strategy 3: Hosting LLMs on Back-end Servers:** Five apps utilize LLMs that are hosted on back-end servers.

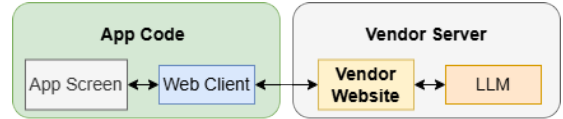


Fig. 6. An overview of calling LLMs using a webview wrapper to submit prompts to the official vendor website.

Description: In this integration strategy, developers either host the LLM on a back-end server and allow their apps to access the hosted LLM using API calls, or provide the infrastructure for the user to host the LLM on the user's server. This strategy avoids memory and computation demands on the user-end devices and reduces the risks of sending users' data to third-party vendors (like sending users' photos/text to OpenAI). However, this strategy requires app developers or the user to set up and run a live server on a separate device.

Insights and Discussion: Out of the five apps, two have no releases. The median number of releases is 1 but rises to 5 when excluding the two apps without releases. Of the three apps that have one or more releases, two have no LLM-related releases, resulting in a median of 0.0% of LLM-related releases. These results show that hosting LLMs on back-end

```

private val chatUrl = "https://chat.openai.com/"
webView.webViewClient = object : WebViewClient() {
    override fun shouldOverrideUrlLoading(
        view: WebView?,
        request: WebResourceRequest?
    ): Boolean {
        val url = request?.url ?: return false

        if (url.toString().contains(chatUrl)) {
            return false
        }

        if (webView.url.toString().contains(chatUrl) &&
            !webView.url.toString().contains("/auth"))
        {
            val intent = Intent(Intent.ACTION_VIEW, url)
            startActivity(intent)
            return true
        }

        return false
    }
}

```

Fig. 7. An example of using a webview client to send prompts with OpenAI’s chat URL [68].

servers may require less maintenance compared to calling third-party APIs. This could be because using a local LLM does not require the developers to adjust to the updates from the vendor side.

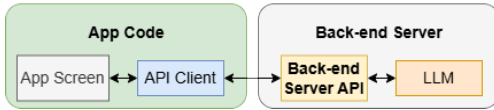


Fig. 8. Calling LLMs hosted on a back-end server.

```

await addMessage(m);
Stream<GenerateCompletionResponse> result =
    await ChatService().generateCompletionStream(
        prompt,
        context: messageContext,
        model: selectedLanguageModel.value ?? '',
    );
...
response?.listen((value) async {
    if (!didStartResponse) {
        didStartResponse = true;
        _currentMessage?.text = '';
    }
    _handleResponseGeneration(value);
});

```

Fig. 9. An example view model for receiving a response from an LLM hosted on a back-end server [69].

4) Strategy 4: Hosting LLMs on User Devices: This is the least popular integration strategy across the dataset, as we find two apps in the dataset that used LLMs locally.

Description: This strategy (shown in Fig. 11) requires the user to load and store the LLM on the end device, which consumes memory and computation but allows the user to run inference privately and without requiring the internet. However, we notice that app developers do not use frameworks to facilitate running LLMs locally, requiring developers to develop their own app-specific infrastructure.

```

Message _createMessage(MessageSource type, String message,
    {String? modelName, context, Map<String, dynamic>? summary}) {
    return Message.fromMap({
        "source": type == MessageSource.userInput
            ? MessageSource.userInput.name
            : MessageSource.generated.name,
        "text": message,
        "chatUuid": chatUuid,
        "createdOn": DateTime.now().millisecondsSinceEpoch,
        "context": context,
        "model": modelName ?? summary?["model"],
        "total_duration": summary?["total_duration"] ?? 0,
        "eval_count": summary?["eval_count"] ?? 0,
        "eval_duration": summary?["eval_duration"] ?? 0,
    });
}

```

Fig. 10. An example view model for creating a response from an LLM hosted on a back-end server [69].

Insights and Discussion: Unlike apps that apply a hybrid strategy, which offers options for local models, both apps are built for specific LLMs. One of the two apps, “*Futo Keyboard*” [61] app, uses llama.cpp [20] and whisper.cpp [70] to run Llama and Whisper models with few parameters, reducing memory demands on the end device. The other runs an open-source GPT4All model.

The “*Futo Keyboard*” app also has the largest number of commits and issues in the dataset, which is the cause of this strategy’s high median for its number of issues. Despite hosting the model and running inference locally, as well as having the highest median software lines of code, this strategy also has a relatively low median percentage for releases that contained updates to LLM-related code. Similar to the previous strategy, this could be because the local LLM requires fewer updates once implemented.

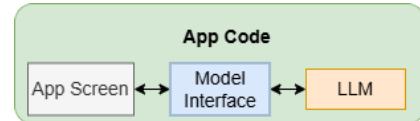


Fig. 11. An overview of calling LLMs hosted on user devices.

5) Strategy 5: Mixing Multiple Integration Strategies:

There are four occurrences of a hybrid strategy in our dataset. **Description:** In this strategy, developers utilize multiple integration strategies, giving users choices between using vendor-provided APIs or calling LLMs locally or from a third party.

Insights and Discussion: Two apps used LLMs both locally and through an API, and two apps used LLMs through both API and back-end servers. This subset contains some of the most popular apps in the dataset, with three out of the four apps in the third quartile across the entire dataset for stars and contributors, and all four in the third quartile for issues and releases. In particular, the “*AppFlowy*” app, which supports APIs for GPT and Claude as well as local models such as Mistral and Llama, has the most stars and releases in the entire dataset, as detailed in RQ1. Another very popular app using this hybrid strategy is “*Maid*” app [72], which is a chatting app that also offers LLMs both through API and locally. This


```

File modelFile = new File(modelPath);
if(!modelFile.exists()){
    String downloadUrl =
        "https://gpt4all.io/models/ggml-gpt4all-j-v1.3-groovy.bin";
    DownloadManager downloadManager =
        (DownloadManager) getSystemService(Context.DOWNLOAD_SERVICE);
    DownloadManager.Request request =
        new DownloadManager.Request(Uri.parse(downloadUrl));
    request.setTitle("File Download");
    request.setDescription("Downloading file...");
    request.setDestinationInExternalPublicDir
        (Environment.DIRECTORY_DOCUMENTS, "ggml-gpt4all-j-v1.3-groovy.bin");
    downloadID = downloadManager.enqueue(request);
    registerReceiver(receiver, new IntentFilter
        (DownloadManager.ACTION_DOWNLOAD_COMPLETE));
    showDownload();
} else {
    pref.edit().putBoolean("model", true).apply();
    showLoading();
    new Thread() -> {
        createEmptyTextFile();
        model = loadGo(modelPath, numThreads);
    }.start();
}

```

Fig. 12. An example of starting a chat with a local model [71].

Below are git commit messages, each separated by newline escape. Provide the list of the top 10 most common topics presented in all the messages with a brief description of each topic.
{commit messages}

Fig. 13. Prompt to generate the ten most common topics.

app supports APIs for GPT, Claude, and Gemini, and local support for Ollama, Llama, Mistral, and other local models, and is second in stars to “AppFlowy” app. As a result, this strategy has a high median number of releases and issues. The median percentage of LLM-related releases is about one in every four releases.

VI. RQ3: WHAT DRIVES DEVELOPERS TO UPDATE LLM-RELATED CODE?

A. Methodology

To determine what drives developers to update LLM-related code, we collected the commit messages from each LLM-related update across all repositories, totaling 2,003 commits, and submitted them to OpenAI’s GPT-4o-mini using a paid OpenAI API subscription, set to zero temperature [34]. We submitted two prompts to the model. The first was a zero-shot prompt that provided all of the commit messages, randomized, and asked the model to provide the list of the top ten most common topics across all of the commit messages as well as a description of each topic (shown in Fig. 13).

The second prompt (shown in Fig. 14) is a few-shot prompt that provides the model with the commit messages in batches of 100 messages. We inputted the list of topics and their descriptions, generated by the previous prompt, and five examples of commit messages and the expected outputs, and asked the model to assign each commit message to one of the topics.

B. Results

We observe that *Feature Additions*, *Refactoring and Code Cleanup*, and *Version Updates* are the most discussed topics when developers change LLM-related code. Tab. VIII shows

the identified ten topics along with their description, frequency, and example commit messages. As shown in table VIII, the most common topics are “*Feature Additions*”, “*Refactoring and Code Cleanup*”, and “*Version Updates*”. We also notice that “*Bug fixes*” and “*Error handling*” topics contribute to 242 commits (12.1%), which indicates the prevalence of developers’ mistakes while adopting LLMs in their apps. We observe that the least common topic is “*Testing and Quality Assurance*”, which may indicate that developers need more testing effort to ensure that the integrated LLMs work properly.

Assign only one topic from the list below to each git commit message, separated by newline escape. Never assign a topic that is not from the list.

Git commit messages:
{commit messages}

List of topics and their meaning:

1. API Integration and Updates: Many commits focus on ...
- ...
10. Testing and Quality Assurance: A number of commits mention ...

Examples:

Message 1: "Update to 6.0.0 (1908)"

Output: Message 1, "Update to 6.0.0 (1908)", Version Updates

Message 2: "refactor: constants"

Output: Message 2, "refactor: constants", Refactoring and Code Cleanup

Message 3: "Implement OpenAI Chat completion API"

Output: Message 3, "Implement OpenAI Chat completion API", API Integration and Updates

Message 4: "feat: add menu on chat activity"

Output: Message 4, "feat: add menu on chat activity", Feature Additions

Message 5: "[fix] Modify OkHttp timeout"

Output: Message 5, "[fix] Modify OkHttp timeout", Bug Fixes

Provide the output as CSV content: Message#, "Message text", Assigned Topic.
Do not provide any explanation or notes. Do not include the word "topic".

Fig. 14. Prompt to assign each commit message to a topic.

We find that “*Feature Additions*” and “*Refactoring and Code Cleanup*” occurs in the top 3 most common topics across all integration strategies. Tab. IX displays the total number of commits across all apps, per integration strategy. The percentage breakdown of commit messages assigned to each topic per integration strategy is shown in Fig. 15. “*Feature Additions*” is not only the most common topic across all commit messages, but also the most common topic for every integration strategy. As well, “*Refactoring and Code Cleanup*” is also in the top 3 most common topics across all integration strategies. *Localization and Language Support*, the second least common topic, is only present for apps that use third-party APIs as their integration strategy.

VII. THREATS TO VALIDITY

Regarding *internal validity*, our study required manual effort in classifying apps as LLM-enabled and identifying LLM-related code, which introduces potential subjectivity. To mitigate this, multiple authors independently reviewed and crosschecked classifications, following predefined criteria to ensure consistency. Additionally, our study’s reproducibility is constrained by the use of an evolving LLM (GPT-4o-mini) for categorizing app functionality and commit message

TABLE VIII
THE COMMIT MESSAGE TOPICS.

Topic	# (%)	Description	E.g., Commit Messages
Feature Additions	607 (30%)	Messages indicate the addition of new features, such as voice input/output, image generation, and chat functionalities.	<i>“feat: suppport translate and summary using local ai (#5858)”</i>
Refactoring and Code Cleanup	442 (22%)	Messages indicate code refactoring activities, such as renaming variables, restructuring files, and removing unused code.	<i>“refactor: move openai state, store and logic into its own service”</i>
Version Updates	293 (15%)	Messages indicate updates to the app’s version.	<i>“Upgrade LangChain.dart to v0.5.0”</i>
Bug Fixes	190 (10%)	Messages indicate fixing bugs, such as handling crashes, incorrect API responses, and UI glitches.	<i>“fix: local ai enabledisable (#6151)”</i>
API Integration and Updates	177 (9%)	Messages indicate the integration of new APIs or updating the integration of existing APIs, such as modifying code to ensure compatibility with the latest API specifications.	<i>“Fixed the model to the latest gpt-3.5-turbo model”</i>
User Interface (UI) Improvements	156 (8%)	Messages indicate enhancing the user interface, such as layout changes, theme updates, and overall design improvements.	<i>“don’t show cost in \$ but in tokens”</i>
Error Handling	52 (3%)	Messages indicate improving error handling mechanisms, such as adding error messages and handling exceptions.	<i>“update: handle open ai not responding or failing in general with a default message”</i>
Performance Optimizations	43 (2%)	Messages indicate optimizing the app’s performance, such as improving loading times, reducing memory usage, and enhancing the efficiency of API calls.	<i>“Batch API calls when there is more than 1000 UUIDs”</i>
Localization and Language Support	27 (1%)	Messages indicate efforts to add or improve localization features, such as adding translations and adjusting language settings.	<i>“Added support of multiple languages of voice assistant”</i>
Testing and Quality Assurance	16 (1%)	Messages indicate testing activities, such as unit tests, integration tests, and quality assurance measures.	<i>“:white_check_mark: model and no stream completions has pass test”</i>

TABLE IX
THE TOTAL NUMBER OF COMMITS PER INTEGRATION STRATEGY.

Integration Strategy	Number of Commits
IS1: Using third-party API calls	1,712
IS5: Mixing multiple integration strategies	189
IS2: Using third-party website	57
IS4: Hosting LLMs on user devices	28
IS3: Hosting LLMs on back-end servers	17

topics. Variability in LLM outputs over time and potential API changes could affect exact replication. To enhance *reliability*, all LLM-based classifications were manually verified by the authors. About *external validity*, our study considered a limited set of LLMs, which may restrict the generalizability of our findings. The results may not fully extend to LLMs with different architectures, training data, or application domains. Future work could expand the analysis to a broader range of models to improve external validity.

VIII. CONCLUSION

In this study, we present a novel dataset of 149 LLM-enabled Android apps and report an empirical investigation into the characteristics of this dataset, the strategies that developers use to integrate LLMs into their apps, and the reasons why developers may update LLM-related code as well as its implications on app releases. We identify five integration strategies used: using a third-party API, calling LLMs from a third-party website, hosting LLMs on a back-end server, hosting LLMs on the user end-device, and a hybrid strategy.

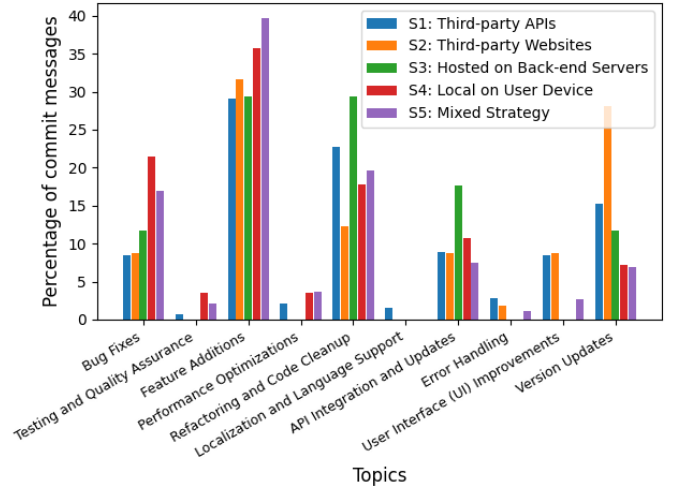


Fig. 15. The distribution of commit message topics per strategy.

Among them, LLM integration via API is the most common in our dataset, while storing LLMs locally on-device is the least common. We also investigate reasons why developers may update LLM-related code by using OpenAI’s GPT-4o-mini to identify the top ten most common topics among all commit messages used for changes to LLM-related code.

IX. ACKNOWLEDGMENTS

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2021-03538 and RGPIN-2021-03969.

REFERENCES

- [1] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.
- [2] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.
- [3] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.
- [4] J. G. Meyer, R. J. Urbanowicz, P. C. Martin, K. O'Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tatonetti, K. J. Won, G. Gonzalez-Hernandez *et al.*, "Chatgpt and large language models in academia: opportunities and challenges," *BioData Mining*, vol. 16, no. 1, p. 20, 2023.
- [5] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "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, 2023, pp. 31–53.
- [6] K. Yang, J. Liu, J. Wu, C. Yang, Y. R. Fung, S. Li, Z. Huang, X. Cao, X. Wang, Y. Wang *et al.*, "If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents," *arXiv preprint arXiv:2401.00812*, 2024.
- [7] S. Carreira, T. Marques, J. Ribeiro, and C. Grilo, "Revolutionizing mobile interaction: Enabling a 3 billion parameter gpt llm on mobile," *arXiv preprint arXiv:2310.01434*, 2023.
- [8] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, and M. Farajtabar, "Llm in a flash: Efficient large language model inference with limited memory," *arXiv preprint arXiv:2312.11514*, 2023.
- [9] T. Çöplü, M. Loedi, A. Bendiken, M. Makohin, J. J. Bouw, and S. Cobb, "A performance evaluation of a quantized large language model on various smartphones," *arXiv preprint arXiv:2312.12472*, 2023.
- [10] R. Murthy, L. Yang, J. Tan, T. M. Awalganekar, Y. Zhou, S. Heinicke, S. Desai, J. Wu, R. Xu, S. Tan *et al.*, "Mobileaibench: Benchmarking llms and llms for on-device use cases," *arXiv preprint arXiv:2406.10290*, 2024.
- [11] Google, "Gemini," 2024, <https://gemini.google.com/>, Last accessed on 2024-11-05.
- [12] OpenAI, "Chatgpt," 2024, <https://chatgpt.com/>, Last accessed on 2024-11-05.
- [13] Github Docs, "About the rest api," 2022, <https://docs.github.com/en/rest/about-the-rest-api/about-the-rest-api?apiVersion=2022-11-28>, Last accessed on 2024-11-05.
- [14] "Replication packages," <https://github.com/kimberlyhau/LLM-enabledAndroidAppsReplicationPackage>.
- [15] Y. Sens, H. Knopp, S. Peldszus, and T. Berger, "A large-scale study of model integration in ml-enabled software systems," *arXiv preprint arXiv:2408.06226*, 2024.
- [16] S. A. Paguthaniya, F. Y. Patel, E. Aminu, and A. Adeleye, "Integration of machine learning models into backend systems: Challenges and opportunities,"
- [17] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kästner, "The product beyond the model—an empirical study of repositories of open-source ml products," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2024, pp. 63–75.
- [18] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, 2014, pp. 221–233.
- [19] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "On the impact of code smells on the energy consumption of mobile applications," *Information and Software Technology*, vol. 105, pp. 43–55, 2019.
- [20] ggerganov, "llama.cpp," 2024, <https://github.com/ggerganov/llama.cpp>, Last accessed on 2024-11-05.
- [21] MLC-AI, "Mlc-llm," 2024, <https://github.com/mlc-ai/mlc-llm>, Last accessed on 2024-11-05.
- [22] LangChain, "Langchain," 2024, <https://www.langchain.com/>, Last accessed on 2024-11-05.
- [23] H. Wang and Y. Guo, "Understanding third-party libraries in mobile app analysis," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 515–516.
- [24] A. Polese, S. Hassan, and Y. Tian, "Adoption of third-party libraries in mobile apps: a case study on open-source android applications," in *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, 2022, pp. 125–135.
- [25] M. Ahasanuzzaman, S. Hassan, and A. E. Hassan, "Studying ad library integration strategies of top free-to-download apps," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 209–224, 2020.
- [26] P. Salza, F. Palomba, D. Di Nucci, A. De Lucia, and F. Ferrucci, "Third-party libraries in mobile apps: When, how, and why developers update them," *Empirical Software Engineering*, vol. 25, pp. 2341–2377, 2020.
- [27] Ollama, "Ollama," 2024, <https://ollama.com>, Last accessed on 2024-11-05.
- [28] Hugging Face, "Open llm leaderboard best models," 2024, <https://huggingface.co/collections/open-llm-leaderboard/llm-leaderboard-best-models-652d6c7965a4619fb5c27a03>, Last accessed on 2024-11-05.
- [29] Novita AI, "Top 10 llm models on hugging face," 2024, https://medium.com/@marketing_novita.ai/top-10-llm-models-on-hugging-face-365c21120bdb, Last accessed on 2024-11-05.
- [30] Nitika Sharma, "Top 10 large language models on hugging face," 2024, <https://www.analyticsvidhya.com/blog/2023/12/large-language-models-on-hugging-face/>, Last accessed on 2024-11-05.
- [31] Tasmia Ansari, "16 best closed-source llms you must know about," 2024, <https://analyticsindiamag.com/ai-mysteries/16-largest-closed-source-llms-you-must-know-about/>, Last accessed on 2024-11-05.
- [32] Ben Lutkevich, "19 of the best large language models in 2024," 2024, <https://www.techtarget.com/whatis/feature/12-of-the-best-large-language-models>, Last accessed on 2024-11-05.
- [33] Harry Guinness, "The best large language models (llms) in 2024," 2024, <https://zapier.com/blog/best-llm/>, Last accessed on 2024-11-05.
- [34] OpenAI, "Openai," 2024, <https://openai.com/>, Last accessed on 2024-11-05.
- [35] Cohere Inc., "Cohere," 2024, <https://cohere.com/>, Last accessed on 2024-11-05.
- [36] Amazon Science, "Alexa teacher models," 2024, <https://github.com/amazon-science/alexa-teacher-models>, Last accessed on 2024-11-05.
- [37] Anthropic, "Claude," 2024, <https://www.anthropic.com/claude>, Last accessed on 2024-11-05.
- [38] deepseek, "deepseek," 2024, <https://www.deepseek.com/>, Last accessed on 2024-11-05.
- [39] Technology Innovation Institute, "Falcon," 2024, <https://falconllm.tii.ae/>, Last accessed on 2024-11-05.
- [40] Google Deepmind, "Gemma," 2024, <https://github.com/google-deepmind/gemma>, Last accessed on 2024-11-05.
- [41] Google, "Lamda: our breakthrough conversation technology," 2021, <https://blog.google/technology/ai/lamda/>, Last accessed on 2024-11-05.
- [42] Huggingface, "Marcoroni 7b - gguf," 2024, <https://huggingface.co/TheBloke/Marcoroni-7b-GGUF>, Last accessed on 2024-11-05.
- [43] C. Zhang, D. Song, Z. Ye, and Y. Gao, "Towards the law of capacity gap in distilling language models," *arXiv preprint arXiv:2311.07052*, 2023.
- [44] Mistral AI, "Mistral," 2024, <https://chat.mistral.ai/>, Last accessed on 2024-11-05.
- [45] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.
- [46] Huggingface, "Nyxene-v3-11b," 2024, <https://huggingface.co/beberik/Nyxene-v3-11B>, Last accessed on 2024-11-05.
- [47] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," *arXiv preprint arXiv:2306.02707*, 2023.
- [48] Google AI, "Palm 2 models," 2024, https://ai.google.dev/palm_docs/palm, Last accessed on 2024-11-05.
- [49] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, and Y. T. Lee, "Textbooks are all you need ii: phi-1.5 technical report," *arXiv preprint arXiv:2309.05463*, 2023.
- [50] M. Abidin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

- [51] Alibaba Cloud, “Qwen,” 2024, <https://qwenlm.github.io/>, Last accessed on 2024-11-05.
- [52] together.ai, “Redpajama, a project to create leading open-source models, starts by reproducing llama training dataset of over 1.2 trillion tokens,” 2024.
- [53] Valiant Labs, “Valiant labs,” 2024, <https://www.valiantlabs.ca/>, Last accessed on 2024-11-05.
- [54] stability.ai, “Meet stable beluga 1 and stable beluga 2, our large and mighty instruction fine-tuned language models,” 2024, <https://stability.ai/news/stable-beluga-large-instruction-fine-tuned-models>, Last accessed on 2024-11-05.
- [55] Stability AI, “Stablelm: Stability ai language models,” 2024, <https://github.com/Stability-AI/StableLM>, Last accessed on 2024-11-05.
- [56] Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, Jiantao Jiao, “Starling-7b: Increasing llm helpfulness harmlessness with rlaiif,” 2024, <https://starling.cs.berkeley.edu/>, Last accessed on 2024-11-05.
- [57] Huggingface, “fblgit/una-xaberi-us-34b-v1beta,” 2024, <https://huggingface.co/fblgit/una-xaberi-us-34b-v1beta>, Last accessed on 2024-11-05.
- [58] Vicuna Team, “Vicuna: An open-source chatbot impressing gpt-4 with 90% chatgpt quality,” 2023, <https://lmsys.org/blog/2023-03-30-vicuna/>, Last accessed on 2024-11-05.
- [59] Al Danial, “cloc,” 2024, <https://github.com/AIDanial/cloc>, Last accessed on 2024-11-05.
- [60] AppFlowy-IO, “Appflowy.io,” 2024, <https://github.com/AppFlowy-IO/AppFlowy>, Last accessed on 2024-11-05.
- [61] futo-org, “Futo keyboard,” 2024, <https://github.com/futo-org/android-keyboard?tab=readme-ov-file>, Last accessed on 2024-11-05.
- [62] virtualwiz1, “Wizgpt,” 2024, <https://github.com/virtualwiz1/wizGPT-flutter>, Last accessed on 2024-11-05.
- [63] flyun, “Chatair,” 2024, <https://github.com/flyun/chatAir>, Last accessed on 2024-11-05.
- [64] itsskyballs, “Oldgpt,” 2024, <https://github.com/itsskyballs/OldGPT>, Last accessed on 2024-11-05.
- [65] Mohamed Hassan, “Ai study assistant,” 2024, <https://github.com/mhss1/AIStudyAssistant>, Last accessed on 2024-11-05.
- [66] HuggingChat, “Huggingchat,” 2024, <https://huggingface.co/chat/>, Last accessed on 2024-11-05.
- [67] OpenAI, “Rate limits,” 2024, <https://platform.openai.com/docs/guides/rate-limits>, Last accessed on 2024-11-05.
- [68] matt haigh, “Chatgpt app,” 2024, <https://github.com/matthaigh27/ChatGPT-android-app>, Last accessed on 2024-11-05.
- [69] Mike Thongvanh, “amallo,” 2024, <https://github.com/mthongvanh/amallo>, Last accessed on 2024-11-05.
- [70] ggerganov, “whisper.cpp,” 2024, <https://github.com/ggerganov/whisper.cpp>, Last accessed on 2024-11-05.
- [71] ronith256, “Localgpt-android,” 2024, <https://github.com/ronith256/LocalGPT-Android>, Last accessed on 2024-11-05.
- [72] Mobile Artificial Intelligence Distribution, “Maid,” 2024, <https://github.com/Mobile-Artificial-Intelligence/maid>, Last accessed on 2024-11-05.