The Journal of Systems and Software 241 (2026) 113005

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Exploratory study of corporate contributions in Android development through repository analysis☆

Sergio R. Montes-León [a,*], Antonio J. Romero-Barrera [b], David Moreno-Lumbreras [a], Hernán Montes-León [a]

[a] Escuela de Ingeniería de Fuenlabrada, Universidad Rey Juan Carlos, Madrid, Spain
[b] Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad Pontificia Comillas, Madrid, Spain

ARTICLE INFO

MSC:
68N30
68M14
68T35

Keywords:
Android Open Source Project
Repository analysis
Google
Data mining

ABSTRACT

Android has become the dominant mobile operating system worldwide, supported by a vast ecosystem of developers and organizations. While Google initiated and continues to maintain the project, its growth has relied on contributions from a wide range of technology companies and independent developers. Understanding the extent and nature of these corporate contributions is essential to assessing the dynamics, influence, and sustainability of the Android ecosystem. This paper presents an updated exploratory study of company participation in Android development, focusing on the distribution of contributions across the kernel, platform, and device modules. We analyze over 3.6 million commits from the official Android repositories, spanning the period from 2005 to April 2026. Our methodology combines automated data extraction and normalization, company identification through email domains, and longitudinal analysis of participation trends. The results show that independent contributors and small-sized companies now lead kernel development, accounting for nearly 70% of kernel commits. At the same time, Google dominates the platform and device layers with shares exceeding 90% and 98%, respectively. In the most recent data (up to April 2026), independent developers are responsible for 61.5% of new commits, with Google holding 16.8%. Other companies, such as Intel, Red Hat, and AMD, maintain a presence but with smaller shares, and most traditional hardware vendors now play only a minor role in the official repositories.

1. Introduction

Android is a Linux-based operating system that has become the leading platform for mobile devices worldwide. Since its initial release in 2008, Android has evolved through numerous major versions, reaching Android 15 (released in 2024–2025) at the time of this study's data collection (April 2026). To understand its evolution, it is essential to distinguish between the Android Open Source Project (AOSP), which refers to the open-source software stack maintained by Google, and the broader Google-led commercial project that includes proprietary Google Mobile Services (GMS). Although Google initiated and remains the primary maintainer of the official AOSP version, its development has evolved into a collaborative effort involving both individual volunteers and major technology companies. This diversity of contributors has led to a dynamic ecosystem and significant fragmentation, as companies adapt and modify the AOSP codebase for their own specific products.

Several studies have examined the AOSP contributions and governance, noting that public code availability can coexist with centralized decision-making and that industry alliance membership does not necessarily imply upstream activity. In our context, these insights help frame which companies sustain contributions and where within Android's layers, complementing earlier evidence on corporate participation dynamics and ecosystem fragmentation (Sharma, 2021b).

The evolution of Android cannot be understood without considering its roots in the Linux ecosystem. Google's decision to build Android on top of the Linux kernel was not just pragmatic, but also highly efficient. Studies show that 99% of Linux kernel functionalities were reused in Android, with only a small fraction, just 0.7% of files, needing modification (Khomh et al., 2012). This reliance on a mature open-source foundation has proven advantageous, as most kernel bugs encountered in Android are resolved by the broader Linux community, not by Android-specific teams. About 95% of kernel issues are fixed upstream, and maintenance remains manageable, with only 5% of

☆ Editor: Dr. Alexander Chatzigeorgiou.

* Corresponding author.
E-mail address: sergio.montes@urjc.es (S.R. Montes-León).

https://doi.org/10.1016/j.jss.2026.113005
Received 1 November 2025; Received in revised form 5 June 2026; Accepted 8 June 2026
Available online 13 June 2026

Android files typically affected by Linux updates. This collaborative model has allowed Android to scale rapidly while maintaining a robust core.

However, the open and collaborative nature of Android also brings challenges. As manufacturers tailor Android to their hardware, compliance with Google's standards is not always guaranteed. Recent longitudinal studies reveal that approximately 20% of Android ROMs fail to meet basic requirements set by Google's Compatibility Definition Document (Possemato et al., 2021). Even some Google-branded devices fall short of these standards. Security policies are especially vulnerable; nearly 30% of ROMs with custom SELinux configurations have been altered in ways that weaken Android's security configuration. Developers often sidestep strict security rules by commenting them out, prioritizing short-term compatibility over long-term safety. This tension between flexibility and security is a recurring theme in the evolution of the Android ecosystem.

Beyond kernel adaptation and compliance, Android development increasingly revolves around performance and code quality. As mobile devices become more powerful and user expectations rise, developers are under pressure to deliver apps that are both fast and resource-efficient. While performance-focused commits are relatively rare, they are distributed throughout the lifecycle of Android projects (Callan et al., 2022). Developers often navigate trade-offs, such as accepting higher memory usage to achieve faster execution or lower bandwidth consumption. Analysis of optimization strategies reveals that the most common approach is deleting redundant code, while frame rate improvements typically involve introducing more concurrency instead of reducing features (Callan et al., 2022).

Power management remains a central concern for mobile development. Android developers employ a range of strategies, including adjusting app behavior based on battery levels, improving overall efficiency, monitoring resource usage, and carefully managing wake locks (Bao et al., 2016). Wake locks, which prevent devices from entering sleep mode when critical tasks are running, appear in about a third of all power-related code changes. These patterns underscore the importance of thoughtful resource management in contexts where battery life is a key factor in user satisfaction.

Another layer of complexity arises from the maintainability and clarity of Android's codebase. A recent analysis of the AOSP found that more than 30% of Java files contain what researchers describe as atoms of confusion (Tabosa et al., 2024). These are small code patterns that can easily mislead developers. With one confusing pattern appearing for every 91 lines of code, this frequency is notably higher than the rates reported in other mature Java projects. For instance, empirical studies on long lived Apache libraries found significantly lower densities of such patterns (Mendes et al., 2022). This comparison suggests that the structural complexity of Android may lead to a higher concentration of ambiguous code, emphasizing that improving coding practices and developer education should remain a priority for the future of the platform.

All these aspects, including kernel adaptation, compliance, performance, power management, and code clarity, are deeply intertwined with the dynamics of corporate participation in Android's development. By examining the distribution and evolution of company contributions across key technical domains, this paper aims to shed light on how industry players shape Android's development and what this means for the platform's sustainability and openness.

A previous research conducted by the authors of this paper and published in 2014 (León et al., 2014) showed that Google employs the largest number of developers contributing to Android, especially in the platform and device modules. Samsung stood out as the top contributor to the kernel, with other companies such as Nokia, Apple, SonyEricsson, and Motorola also playing important roles. Notably, the Android kernel was already largely developed by the broader open-source community, reflecting the project's maturity and expansion at that time.

This paper represents a follow-up replication and extension of our previous 2014 study (León et al., 2014), revisiting those findings to provide new insights into the current landscape of corporate contributions to Android development. The present work compares recent data (2005–2026) with the earlier baseline results. A key methodological difference is that while the previous 2014 study collected data from GitHub's Android repositories (https://github.com/android), the current analysis uses Google's official repository platform (Android Open Source Project, 2025a), which provides more authoritative and complete data. The main objectives of this replication study are to: (1) identify which companies currently employ the most developers contributing to Android, (2) verify whether the private sector's contribution patterns observed in 2014 are still present in the current data, and (3) document the emergence of new companies as regular contributors over the past decade.

Building on these previous insights and considering the changes in both the Android ecosystem and its development practices over the past decade, our study seeks to address several key questions that have emerged from this evolving landscape. To better understand the current dynamics of corporate and independent participation in Android development, as well as the impact of automation and the shifting roles of major contributors, we have formulated the following research questions (RQs):

RQ1: Automation and contribution dynamics

RQ1.1: How has the degree of automation in Android development changed over time?
RQ1.2: What is the current balance between human and automated (bot) contributions across different technical domains?

RQ2: Developer typology and ecosystem distribution

RQ2.1: What is the relative participation of independent developers and private companies in the main layers of the Android ecosystem?
RQ2.2: How has this distribution evolved since previous studies?

RQ3: Corporate participation and evolution

RQ3.1: Which companies currently play the most significant roles in Android development?
RQ3.2: Are there emerging contributors compared to a decade ago?
RQ3.3: How have corporate engagement patterns shifted over time?

To facilitate understanding, the remainder of this paper is organized as follows. Section 2 provides an overview of the AOSP architecture, describing how its modular and layered structure shapes the organization of repositories and the distribution of technical domains. Section 3 details the evolution of the method, comparing the approach from the 2014 study with the updated pipeline used in this work. Section 4 presents the main results, analyzing the distribution of corporate contributions across different domains and highlighting the most relevant trends. Section 5 examines the evolution of participation patterns, the rise of independent developer contributions, private sector strategic shifts, and the implications for platform governance and coherence. Finally, Section 6 summarizes the key findings and discusses their implications for the Android ecosystem.

2. Android architecture overview

The AOSP architecture (Android Open Source Project, 2025b) is organized as a modular and layered stack, and this structure is directly

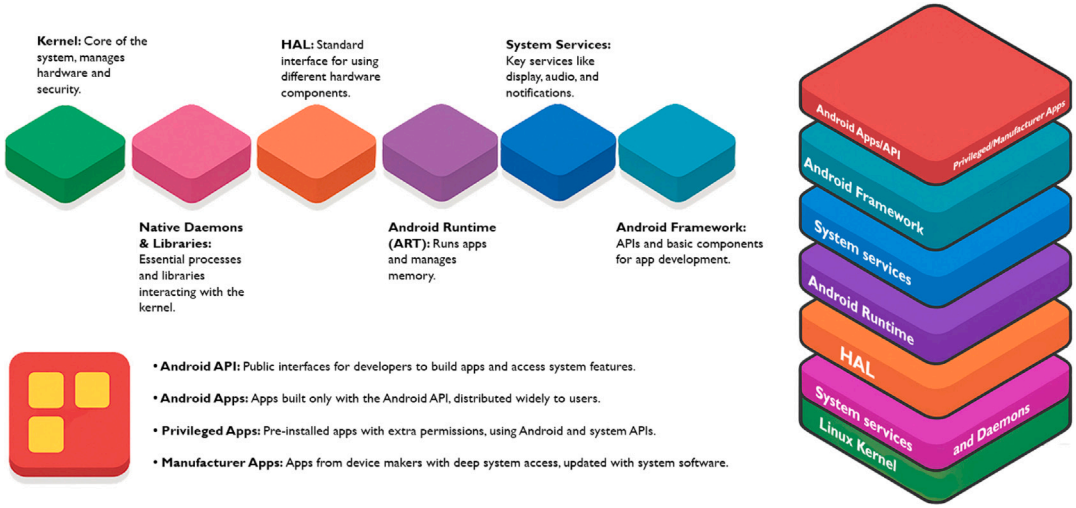

Fig. 1. Android Open Source Project (AOSP) architecture overview. Own illustration based on official documentation (Android Open Source Project, 2025b).

reflected in Google's official repositories. Each core layer, including the kernel, native daemons and libraries, hardware abstraction, runtime, system services, and framework, is managed in its own repository or group of repositories. This separation allows companies and independent developers to focus on specific technical domains. For instance, kernel development takes place in repositories dedicated to hardware support, while other repositories are responsible for device code, platform services, or the application framework.

This modular organization facilitates scalable collaboration and enables easier management of contributions across various parts of the platform. It also helps maintain code ownership and enables compatibility testing at every layer. Thanks to this structure, companies can contribute to the areas of Android that are most relevant for their products, whether they are working on hardware optimizations, system drivers, or new features for app developers. The result is a diverse ecosystem where contributions are distributed across many technical domains.

It is also important to distinguish between Google/AOSP and Google's proprietary Android distribution. While AOSP provides the open-source foundation of the operating system Google's version adds closed-source components such as the Play Store and other proprietary APIs that enable integration with its ecosystem. This distinction is relevant, as some contributions target the open AOSP layers, while others are directed toward Google's proprietary extensions. Therefore, the subsequent analysis in this paper explicitly differentiates between contributions made to the AOSP and those related to Google's extended Android ecosystem.

Fig. 1 visually illustrates the Android architecture, showing how each layer fits into the overall stack. The diagram highlights the separation between layers, from the Linux kernel at the base to the application layer at the top, and helps clarify how responsibilities and contributions are divided. This visualization supports the analysis of corporate participation in this paper, showing how the layered design facilitates collaboration and specialization across the Android ecosystem.

3. Methodology

The methodology for analyzing corporate participation in Android development has evolved considerably since the previous study by León et al. (2014). Below, we first summarize the original approach in Section 3.1 and then, in Section 3.2, we describe the updated pipeline used in this work. In Table 1, we have summarized the key differences between the methodologies used in each study, highlighting how the approach has evolved to address new challenges and leverage modern tools.

3.1. Previous methodology (2014)

The earlier study by León et al. (2014) relied on a semi-automated process:

1. **Downloading repositories:** The Android repositories were cloned from GitHub (https://github.com/android).
2. **Data filtering:** Using Bash scripts and CVSAnaly (Robles et al., 2004; Herraiz et al., 2004), the Git history from each repository (kernel, platform, device) was extracted and stored in MySQL databases.
3. **Data parsing and manipulation:** Email addresses were extracted from the databases and saved as CSV files for each module.
4. **Final data processing:** With Python scripts, email domains were separated and counted, and the results were sorted and visualized using spreadsheet software.

While effective for the time, this workflow was limited in terms of automation, scalability, and reproducibility, especially as the Android project and its repository structure grew more complex. Additionally, CVSAnaly is now outdated, and many of the APIs it relied on are deprecated or no longer maintained, making it increasingly difficult to apply the same process to the current Android ecosystem.

3.2. Updated methodology

To address these limitations and reflect the current structure and activity of the Android ecosystem, the methodology was redesigned with a focus on automation, reproducibility, and deeper analysis.

The repository data were collected during April 2026. The dataset spans from 2005 (the initial Android commit) through April 2026, providing over two decades of development history. The 2026 data represent a partial year (January through April 2026). This partial-year coverage explains why 2026 appears with lower commit counts in temporal analyses. The comprehensive historical coverage enables robust longitudinal analysis of corporate contribution patterns. The updated process consists of the following main phases:

1. **Data extraction:** Commit data is collected from all relevant official Google repositories (Android Open Source Project, 2025a) (android15-qpr2-release) using the Perceval tool (Dueñas et al., 2018), which is designed specifically for mining software repositories. Perceval exports the commit history in JSON Lines (JSONL) format, organizing the data by technical domain (framework, kernel, devices).

Table 1
Comparison between previous and updated methodologies for analyzing corporate participation in Android development.

Aspect	Previous methodology	Updated methodology
Data sources	GitHub (https://github.com/android)	Official Google repositories (https://android.googlesource.com), including kernel, framework, and device-specific repos
Tools used	Bash scripts, CVSAnaly, MySQL, Python scripts, spreadsheet software	Perceval (for mining), Python for processing, Jupyter notebooks for analysis, Makefiles/scripts for automation
Data extraction	Manual/semi-automated cloning and filtering using CVSAnaly	Automated extraction with Perceval, exporting to JSONL format
Data processing	Parsing email addresses, saving as CSV, manual domain separation, and counting	Normalization of company aliases, metric computation (commits, churn, directories), aggregation, CSV export
Automation	Limited; several manual steps, not fully reproducible	Full batch-mode automation via scripts/Makefiles; reproducible, quality controls, and data dictionary for consistency
Scalability	Limited by manual steps and tool obsolescence; difficult to adapt to growing/complex repo structures	Highly scalable; supports large, evolving repository structures and easy updates
Reproducibility	Low; manual interventions and deprecated tools hinder repeatability	High; all steps scripted, documented, and repeatable
Limitations	Outdated tools (CVSAnaly), deprecated APIs, manual effort, low scalability and reproducibility	Modern, actively maintained tools, automation, transparency, scalable and updatable

In particular, the following repositories and categories were considered:

- **Kernel:** `android15-6.6-r20` — Android Common Kernel 6.6 (revision r20) — `kernel/common`
- **Framework:** `android-15.0.0_r20` — AOSP Platform Components:
  - `platform/frameworks/base` — Core frameworks
  - `platform/system/core` — System core utilities
  - `platform/system/sepolicy` — SELinux policies
  - `platform/art` — Android Runtime
  - `platform/build` — Build system
  - `platform/packages/apps/Settings` — Settings app
  - `platform/packages/apps/Launcher3` — Launcher
- **Devices:** `android15-qpr2-release` — Pixel Reference Devices:
  - `device/google/bluejay`
  - `device/google/pantah`
  - `device/google/lynx`

2. **Normalization and aggregation:** The raw JSONL files are processed to unify company aliases (e.g., "Google LLC" and "Alphabet" are consolidated as "Google") and to compute key metrics such as total commits, annual time series, code churn, and top contributing directories. These results are saved as structured CSV files per domain.
3. **Analysis and visualization:** Jupyter notebooks load the aggregated CSVs to compute participation shares, historical trends, and comparative rankings. The notebooks generate all tables, bar charts, line plots, and heatmaps included in the paper, as well as summaries and top directories by company and year.
4. **Automation and reproducibility:** All steps can be executed in batch mode through scripts or Makefiles, and a data dictionary ensures consistency in metrics and column names. Quality controls are included to check for consistency in commit counts, company aliases, and time aggregation.

This new pipeline, built around Perceval and automated data science workflows, enables a more comprehensive, scalable, and transparent analysis of corporate contributions to Android, making it easier to update results and compare trends. Fig. 2 illustrates the complete architecture of the Android repository analysis pipeline, from data extraction and normalization to visualization and reproducible automation.

3.3. Author identity resolution and bot detection

The automation of software development workflows has become increasingly prevalent in recent years, with bots and automated accounts now responsible for a substantial proportion of repository activities. Studies show that in large open source ecosystems, bots can account for more than 15% of all commits and up to 30% of pull requests, depending on the project and hosting platform (Wessel et al., 2018; Dey et al., 2020). These automated agents perform a wide range of tasks, from code integration and dependency updates to continuous integration and testing (Erlenhov et al., 2019; Monperrus, 2019). While automation enhances productivity and project maintainability, it also introduces new challenges for empirical research, as bot-generated activity can distort measurements of human participation, company involvement, and collaboration patterns (Dey et al., 2020; Chidambaram et al., 2025).

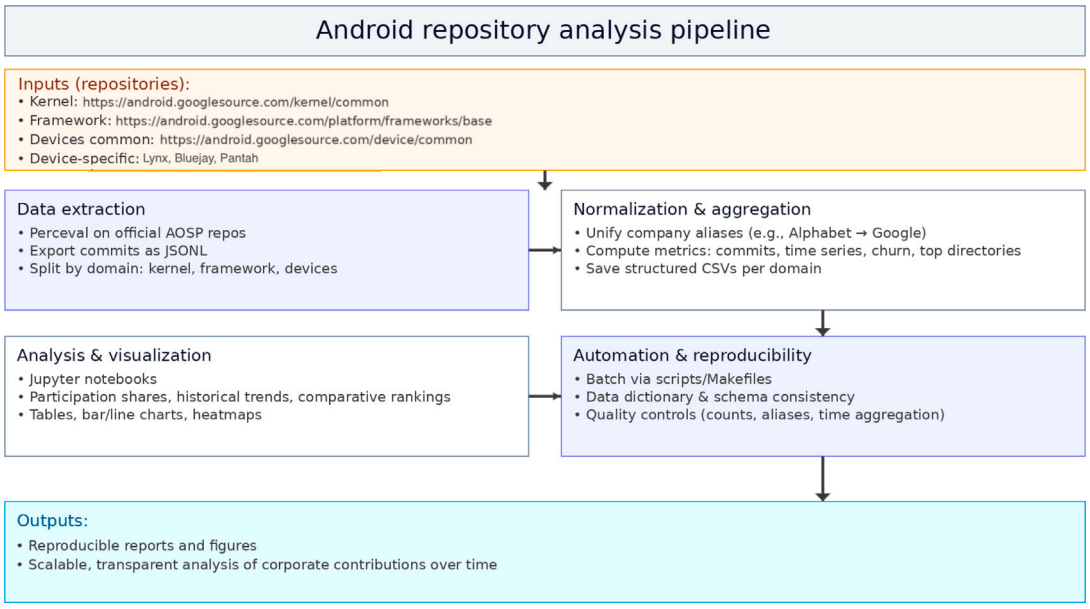

Fig. 2. Android repository analysis pipeline.

In the context of the Android ecosystem, the presence of bots is especially prominent in large-scale projects managed by corporations such as Google, where automated accounts are routinely used for code merging, build orchestration, and quality assurance (Dey et al., 2020; Chidambaram et al., 2025). As a result, distinguishing between human and automated contributions is essential for obtaining accurate metrics on developer participation and organizational engagement. Failure to filter out bots can lead to inflated commit counts for certain companies and obscure the true dynamics of collaborative development (Fry et al., 2020).

To address these issues, we adopted a two-stage data cleaning process:

1. **Automated bot detection:** We implemented automated bot detection using heuristics established in the literature (Dey et al., 2020; Chidambaram et al., 2025). Contributors were flagged as bots if their names or email addresses matched common bot-related patterns (e.g., containing keywords such as "bot", "robot", "automerge", "build", or "noreply"), or if their commit activity exhibited highly regular, automated characteristics. This rule-based approach is widely used in large-scale repository mining because of its transparency and effectiveness in filtering out non-human accounts.
2. **Author identity resolution:** We performed author identity resolution to merge aliases and unify contributions from the same individual, even when slight variations in name or email were present. This process followed methods recommended by other previous works (Fry et al., 2020; Amreen et al., 2020), involving normalization of names (removing capitalization and special characters) and grouping email addresses by their main identifier (i.e., the part before the "@" symbol). By consolidating such aliases, we reduced the risk of overcounting contributions and ensured a more accurate representation of both company and independent developer involvement.

The cleaned and consolidated author dataset was then used for all subsequent analyses, ensuring that metrics such as commit counts and company rankings reflect human participation, not automated processes.

As software development increasingly incorporates AI-assisted coding tools and autonomous agents, distinguishing between purely human contributions and those augmented or generated by AI systems presents a growing methodological challenge (Yetiştiren et al., 2023). Current AI coding assistants such as GitHub Copilot and Amazon CodeWhisperer operate primarily as human augmentation, where developers commit AI suggestions under their own identities (Yetiştiren et al., 2023; Cui et al., 2024). More recently, agentic development paradigms have emerged where autonomous AI agents can propose, review, and commit code with minimal human oversight.

In the context of this study, most commits remain attributable to clearly identifiable human developers or traditional automation bots with explicit bot-like identities. However, AI-assisted code contributions are likely already present in the dataset, committed under human accounts, and indistinguishable from purely human-authored code using commit metadata alone (AboutCode, 2024). Detecting such contributions would require code pattern analysis beyond repository metadata, which is beyond this study's scope but represents important future research (AboutCode, 2024).

Agentic commits by AI agents under distinct identities represent an emerging category likely to become more prevalent in Android development. Future replications should identify AI agent accounts through naming conventions, email patterns, or commit characteristics. As the ecosystem evolves, distinguishing human contributions, traditional bots, AI-assisted work, and autonomous AI agents will become essential for accurately characterizing participation dynamics and assessing human effort in software evolution.

3.4. Metrics and mapping to research questions

To systematically address our research questions, we defined a set of quantitative metrics extracted from the Android repositories. Table 2 presents these metrics, their definitions, and their mapping to the corresponding research questions.

The choice of these metrics reflects both our research objectives and established best practices in mining software repositories (Codabux et al., 2024; Ampatzoglou et al., 2013; Poncin et al., 2011). Commit counts provide a consistent proxy for participation frequency and engagement, while the human-versus-bot distinction addresses the increasing role of automation in modern software development. Company participation shares and temporal trends enable comparative analysis across organizations and historical periods, supporting our investigation into shifts in corporate engagement. Code churn complements commit counts by capturing the magnitude of code changes,

Table 2
Metrics used in the study and their mapping to research questions.

Metric	Definition	Research questions
Total commit count	Total number of commits in a repository or domain, aggregated by company, contributor type (human/bot), or time period.	RQ1.1, RQ1.2, RQ2.1, RQ2.2, RQ3.1, RQ3.2
Human vs. bot ratio	Proportion of commits made by human contributors versus automated bots, computed per technical domain.	RQ1.1, RQ1.2
Company share	Percentage of total commits attributable to each company or independent developers.	RQ2.1, RQ2.2, RQ3.1, RQ3.2, RQ3.3
Temporal trends	Annual time series of commit counts, enabling longitudinal analysis of participation patterns.	RQ1.1, RQ2.2, RQ3.2, RQ3.3
Code churn	Total lines added and deleted per commit, providing insight into the magnitude of contribution.	RQ2.1, RQ3.1
Top directories	Most actively modified directories, revealing focus areas and strategic priorities.	RQ3.1, RQ3.3
Domain participation	Distribution of contributions across kernel, framework, and device layers.	RQ2.1, RQ2.2, RQ3.1, RQ3.3

thus providing a more complete picture of contribution impact. Finally, analysis of top contributing directories and domain-specific participation patterns reveals strategic focus areas and helps interpret why certain companies concentrate their efforts in particular layers of the Android architecture.

3.5. Differences between official Android repositories and GitHub mirrors

A key methodological consideration in any analysis of the AOSP is the choice of data source. While both Google and GitHub host repositories related to Android, there are important structural and operational differences between them that directly affect the completeness and accuracy of any empirical study.

The official Android repositories are hosted by Google at https://android.googlesource.com. These repositories constitute the authoritative and canonical source of all AOSP code, including the kernel, frameworks, device-specific code, and supporting infrastructure. All official development, code review, and integration processes are managed through Google's own infrastructure, and changes are reflected here first. This includes the full commit history, branch structure, and metadata, as well as the use of internal tools and bots that are essential for tracking the true evolution of the project.

In contrast, the Android repositories available on GitHub (https://github.com/android) are mirrors that are periodically synchronized from the official source. Although these mirrors make it easier for developers to browse, fork, and contribute using the familiar GitHub interface, they do not always capture the most up-to-date state of the project. Some repositories or branches may be missing, and certain metadata (such as detailed commit signatures, code review records, or bot activity) may not be fully preserved in the mirroring process. Furthermore, the GitHub mirrors are not used for official code review or integration, and contributions made via GitHub pull requests are not directly accepted into AOSP.

For these reasons, this study relies exclusively on the official repositories hosted by Google to ensure that all analyses are based on the complete and authoritative development history. This approach guarantees the most accurate measurement of corporate participation, automation levels, and long-term trends in Android development, avoiding the potential inconsistencies and omissions that may arise from relying on GitHub mirrors.

4. Results

The Android development ecosystem presents a complex landscape of corporate participation and technical contributions that has evolved over the past two decades. This section examines the distribution of development efforts across companies and technical domains, revealing both the concentration of contributions among key players and the areas where most development activity occurs. Through a comprehensive analysis of commit data spanning from 2005 to 2026, we explore how corporate influence has shifted over time and how different technical layers of the Android stack reflect varying degrees of openness and centralization.

Our analysis encompasses three primary technical domains that form the foundation of the Android ecosystem: the kernel layer, which provides the core operating system functionality; the framework layer, which implements the Android-specific APIs and services; and the devices layer, which handles hardware-specific implementations and drivers. Each domain exhibits distinct patterns of corporate participation and community involvement, reflecting different strategic priorities and development philosophies within the broader Android project.

Fig. 3 provides a global overview of corporate contributions to Android development, broken down by company and technical domain. On the left, the stacked bar chart shows the total number of commits attributed to each major company or group, with colors distinguishing between the three main layers of the Android project: *Devices* (yellow), *Kernel* (blue), and *Framework* (red). For each company, the height of the bar corresponds to the aggregate number of commits, and the colored segments indicate the proportion of contributions to each domain. Notably, the categories "Others" (representing independent and smaller contributors) and "Google" dominate the chart, especially in the *Kernel* and *Framework* layers. On the right, the donut chart summarizes the overall distribution of commits across domains, regardless of company. It reveals that the *Kernel* and *Framework* account for roughly equal shares (49.3% and 50.4% respectively), while the *Devices* layer represents only a small fraction (0.4%) of total contributions. Together, these visualizations highlight both the concentration of contributions among a few key players and the technical areas where most development activity occurs.

The evolution of company participation in Android development is clearly reflected in the trends shown in the percentage share of commits. Fig. 4 captures these dynamics, highlighting both the shifts

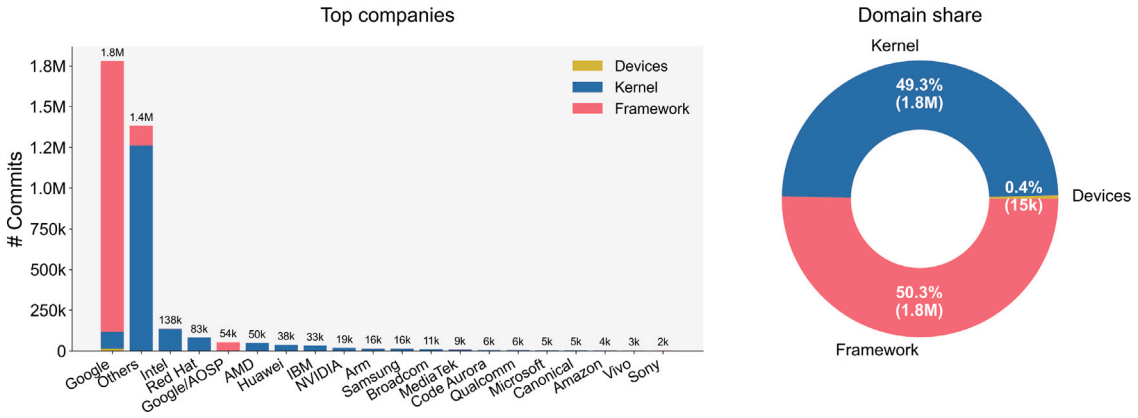

Fig. 3. Commits by company and development layer (global overview).

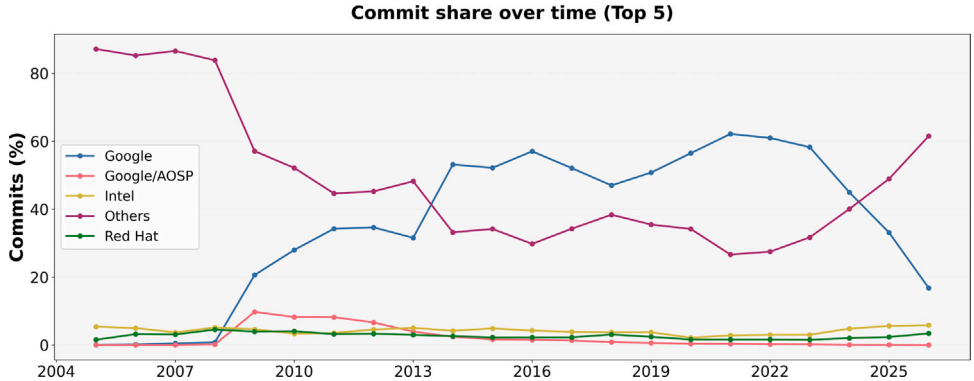

Fig. 4. Percentage trends in company contributions over the years.

in corporate influence and the resilience of independent contributions within the Android ecosystem. The line chart illustrates the annual shifts in the relative contributions of each major company and the independent contributors group from 2005 to 2026, with the y-axis representing the share of commits (as a percentage) and the x-axis indicating each year. "Others" tag held a dominant position in the early years, consistently exceeding 80% of all commits, but this share steadily declined as Google's involvement grew. After 2008, Google's share rose sharply, eventually overtaking the independent group and remaining the leading corporate contributor for much of the last decade. In recent years, however, independent contributors have regained the lead, while Google's share has decreased. Other companies, such as Intel, Red Hat, AMD, and Samsung, show much smaller and relatively stable participation throughout the period.

Several distinct inflection points are visible in Fig. 4, each corresponding to strategic decisions or external events in Android's evolution:

- **2008–2010: Google's rapid ascent.** Following Android's public release in 2008 and the launch of the first commercial Android device (HTC Dream/T-Mobile G1), Google increased its direct involvement in Android's development, while the platform was still in a rapid growth phase. This early period required substantial iteration to stabilize the ecosystem and expand the platform's functionality (Sharma, 2021a).
- **2011–2014: Hardware manufacturer engagement.** The early 2010s saw substantial participation from handset manufacturers and other firms in the Android Open Source Project. Prior work on Android contributions shows that manufacturers such as Samsung, HTC, and others contributed source code and that such contributions were associated with faster time-to-market for their first Android devices (Sharma, 2021a; Yamaguchi, 2016).
- **2015–2019: Strategic withdrawal to private forks.** As Android matured, some actors appear to have shifted effort away from visible upstream contribution and toward alternative or indirect contribution strategies. Sharma notes that some contributors were not contributing directly but were instead circumventing the open contribution method, suggesting that governance and intellectual-property considerations shaped public contribution patterns (Sharma, 2021a).
- **2020–present: Kernel ecosystem fragmentation and modularization.** Recent changes in Android development reflect a more fragmented kernel ecosystem, where upstream Linux, AOSP, and OEM kernels interact through delayed patch propagation. Zhang et al. (2021) show that many OEM kernels remain binary and that patch delays can last months or even years, which helps explain why contribution patterns in the kernel domain may differ from framework-level development. Google's modularization efforts also shifted some platform maintenance toward updateable system components (Sharma, 2021a).

These inflection points demonstrate how Android development patterns respond to both internal strategic decisions (Google's evolving governance model, manufacturers' competitive strategies) and external market dynamics (smartphone market maturation, hardware commoditization, upstream Linux evolution). Understanding these drivers is essential for interpreting raw commit statistics, as changes in contribution volumes often reflect governance and business model shifts rather than technical factors.

The company's participation in Android development has changed over the past decade. In Table 3, we observe how the distribution of commits among leading contributors has shifted from a somewhat more diverse mix to a scenario where Google and the group labeled as "Others" together account for the vast majority of activity. The evolution

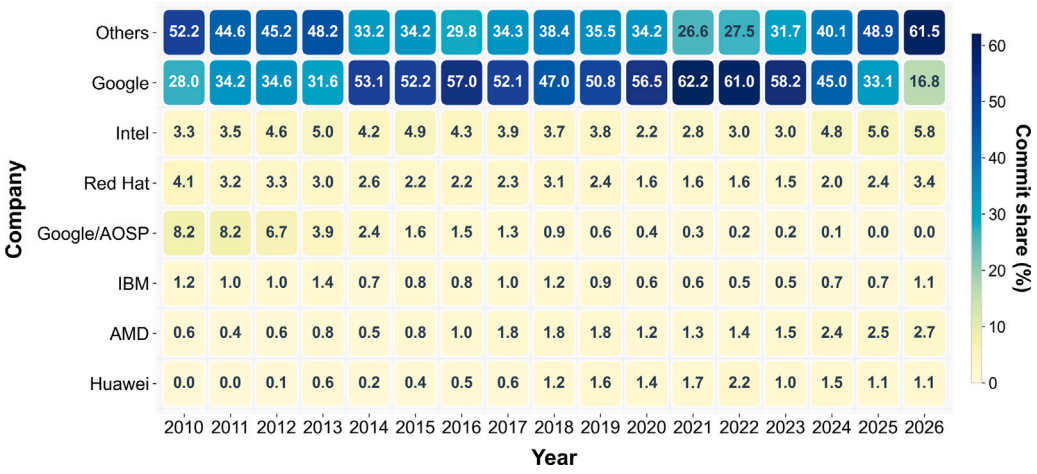

Fig. 5. Year-by-year commit share distribution (%) for top contributors (2010-April 2026).

Table 3
Commits distribution by companies – Historical (2005–2025) vs Q1 2026 (2026).

Historical (2005–2025)			January-April 2026		
Company	Commits	Share [%]/Rank	Company	Commits	Share [%]/Rank
Google	1,773,849	48.91/1	Others	24,391	61.37/1
Others	1,357,796	37.44/2	Google	5827	14.66/2
Intel	135,251	3.73/3	Intel	2681	6.75/3
Red Hat	81,810	2.26/4	Red Hat	1300	3.27/4
Google/AOSP	54,009	1.49/5	AMD	1150	2.89/5
AMD	48,755	1.34/6	Qualcomm	1146	2.88/6
Huawei	37,637	1.04/7	NVIDIA	660	1.66/7
IBM	33,090	0.91/8	Huawei	633	1.59/8
NVIDIA	18,291	0.50/9	IBM	397	1.00/9
Samsung	15,653	0.43/10	Xiaomi	235	0.59/10
Arm	15,567	0.43/11	Arm	222	0.56/11
Broadcom	11,026	0.30/12	Broadcom	180	0.45/12
MediaTek	9048	0.25/13	Canonical	174	0.44/13
Code Aurora/Qualcomm	6305	0.17/14	MediaTek	133	0.33/14
Microsoft	5190	0.14/15	OPPO	122	0.31/15
Qualcomm	4779	0.13/16	Microsoft	115	0.29/16
Canonical	4703	0.13/17	Vivo	111	0.28/17
Amazon	3778	0.10/18	Samsung	106	0.27/18
Vivo	2642	0.07/19	Amazon	81	0.20/19
Sony	1884	0.05/20	Sony	48	0.12/20

has been particularly notable in the most recent years. In 2024, Google still held a larger share than independent contributors, but this balance reversed in 2025, with independent developers surpassing Google for the first time since the early 2010s. This trend accelerated further in the first quarter of 2026 (January-April), when independent contributors surged to 61.37% while Google's share collapsed to just 14.66%, marking a significant inflection point in the ecosystem's balance. Meanwhile, companies such as Intel, Red Hat, and AMD maintain a presence but with much smaller shares, and most traditional hardware vendors now play only a minor role.

To provide deeper insight into the temporal evolution of corporate participation, Fig. 5 presents the complete year-by-year progression from 2010 onwards through a heatmap representation, where color intensity represents commit share percentage for each company-year combination. This visualization makes temporal patterns immediately apparent, highlighting the consistent dominance of independent developers and Google, the relative stability of other contributors, and specific inflection points such as 2022, when Google reached its maximum share. The share of independent developers has fluctuated, declining from 52.2% in 2010 to 27.5% in 2022 and then recovering to 40.1% in 2024, further increasing to 48.5% in 2025, and reaching 61.4% in April 2026. Google's share follows an inverse pattern, increasing from 28.0% in 2010 to a peak of 61.0% in 2022 and subsequently decreasing to 45.0% in 2024, further declining to 33.7% in 2025 and dropping to 14.7% in April 2026. Other major contributors, including Intel, Red Hat, and AMD, maintain relatively stable yet modest shares, typically ranging between 2% and 6% throughout the period. This temporal analysis supports the earlier discussion of inflection points, providing a quantitative demonstration of how the balance of influence has shifted between independent contributors and corporate participants over the evolution of Android.

On the other hand, Table 4 details how these contributions are distributed across the main technical domains of Android, specifically devices, framework, and kernel. This breakdown reveals even greater centralization: independent contributors now dominate the kernel, while Google's influence in the framework and device layers is overwhelming. The following subsections examine each domain in detail, highlighting the main trends and their implications for the Android ecosystem.

4.1. Kernel contributions

The kernel remains the most community-driven part of Android. Looking at the Kernel column in Table 4, the distribution of commits among the top contributors confirms this open and collaborative nature. The leading positions are held by independent developers and small-sized companies (Others) with 1,262,015 commits (69.72% of kernel commits overall), followed by Intel with 136,282 commits

Table 4
Top 10 contributing companies by number of commits in kernel, framework, and devices (2005–2026).

Kernel			Framework			Devices		
Company	Commits	Share [%]	Company	Commits	Share [%]	Company	Commits	Share [%]
Others	1,262,015	69.72	Google	1,661,391	90.21	Google	14,520	98.10
Intel	136,282	7.53	Others	119,891	6.51	Others	281	1.90
Google	103,765	5.73	Google/AOSP	52,295	2.84	–	–	–
Red Hat	83,099	4.59	Intel	1650	0.09	–	–	–
AMD	49,902	2.76	Sony	1163	0.06	–	–	–
Huawei	38,209	2.11	Xiaomi	779	0.04	–	–	–
IBM	33,487	1.85	Code Aurora/Qualcomm	722	0.04	–	–	–
NVIDIA	18,879	1.04	Samsung	707	0.04	–	–	–
Arm	15,158	0.84	Arm	631	0.03	–	–	–
Samsung	15,052	0.83	Motorola	624	0.03	–	–	–

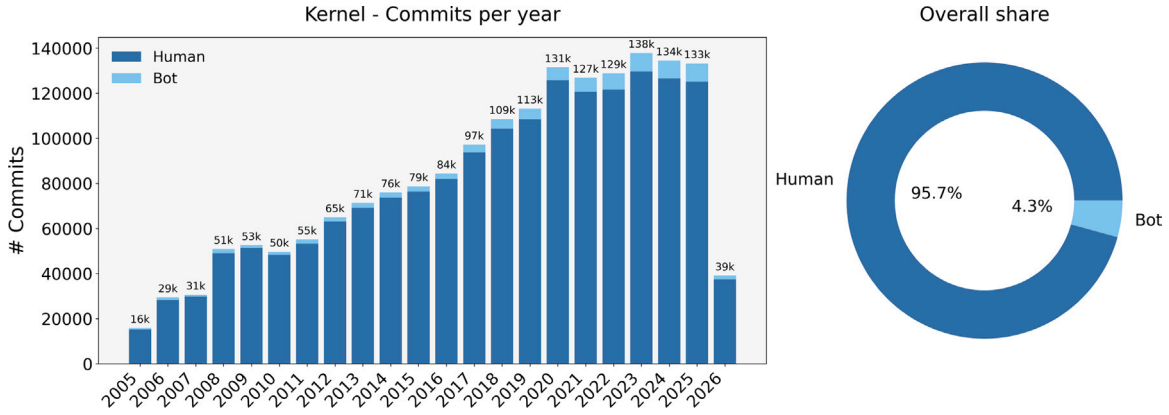

Fig. 6. Annual evolution of human and bot commits in the kernel domain, and overall share.

(7.53%) and Google with 103,765 commits (5.73%). Other notable contributors include Red Hat (83,099 commits, 4.59%), AMD (49,902 commits, 2.76%), and Huawei (38,209 commits, 2.11%). This distribution highlights the distributed and upstream-driven nature of kernel work, where no single company dominates and community participation decisively shapes the direction. This pattern matches the chart, where human developers clearly drive the majority of kernel activity.

The annual results show steady long-run growth in yearly commits from 2005 into the early 2020s, with several recent peaks above 130k total commits. Each year is overwhelmingly dominated by human contributions, while bot activity appears as a thin layer that becomes slightly more visible later but never approaches the human volume. The overall split confirms this structural pattern: 95.7% of kernel commits are from humans and 4.3% from bots (see Fig. 6).

Historically, this balance has a clear rationale. Kernel development depends on nuanced reviews, upstream coordination, and long-lived technical decisions. Automation supports integration, regression checks, and routine verifications, but it does not replace human judgment. From a company perspective, the kernel is the layer where upstream presence and deep expertise translate most directly into visible activity and long-term influence. It also means aggregate commit counts are a reasonable proxy for human effort in this domain, unlike in more centralized layers. In short, the kernel shows a healthy mix in which automation strengthens robustness and cadence while human contributors carry the core workload and shape direction.

4.2. Framework contributions

The framework layer is far more automated and centralized. The annual series starts modestly in 2008 and grows in waves, with strong surges around 2014 to 2016 and then a major jump from 2020 onward. Peak years exceed 160k total commits, with large stacked segments for both humans and bots. The overall split shows about 75.0% human and 25.0% bot commits, a share large enough that any interpretation of contribution volumes must account for automation (see Fig. 7).

The increase in framework commits from 2020 onward coincides with several major Android platform changes:

- **Project Mainline expansion (2019–2022):** Android 10 introduced Project Mainline, which modularizes selected system components and allows them to be updated outside the normal Android release cycle through Google Play system updates or partner OTA mechanisms (Android Open Source Project, 2025c). This architectural shift increased the importance of clear module boundaries and compatibility layers, and it likely contributed to additional framework work as the updatable surface of the platform grew.
- **Major UI overhaul – Material You (2021):** Android 12 introduced Material You, a design system centered on dynamic color and a more expressive visual experience (Android Open Source Project, 2026). Official Android documentation describes dynamic color support as part of the Android 12 Material You color extraction flow, which required system-level support across framework and UI components (Android Open Source Project, 2026; Android Developers Blog, 2022).
- **Privacy and security enhancements (2020–2026):** Android 11 introduced permission auto-reset, one-time permissions, and scoped storage enhancements, while Android 12 added privacy indicators. Android 12/13 continued strengthening privacy-related platform behavior (Android Developers, 2026a,b; Android Open Source Project, 2025d; Android Developers, 2023). Android 14 further expanded privacy controls with advanced photo picker capabilities and enhanced notification permissions, while Android 15 (released 2024–2025) introduced private space for sensitive apps and refined background process restrictions. These progressive changes required sustained updates to permission handling, storage behavior, and system UI across multiple releases, all of which are implemented in or depend on framework-level code.
- **Automation and validation infrastructure:** As platform changes became more modular and privacy-sensitive, automated testing

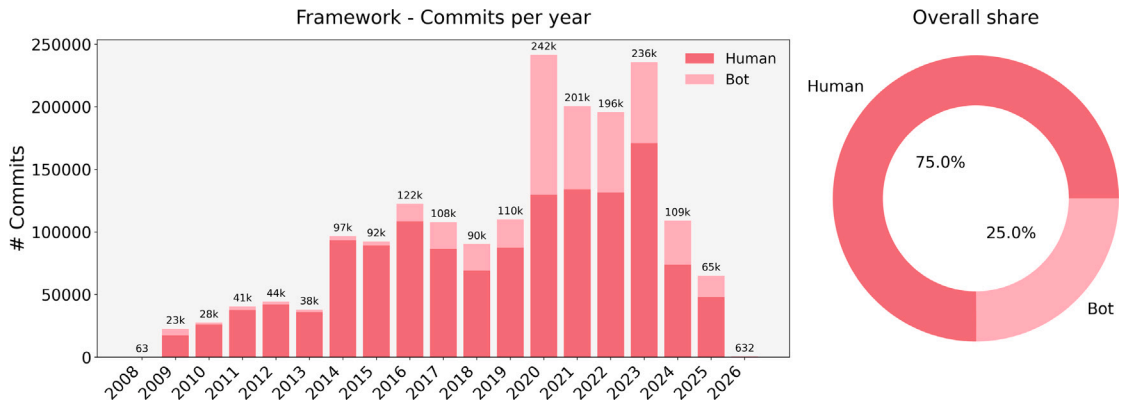

Fig. 7. Annual evolution of human and bot commits in the framework domain, and overall share.

and validation became increasingly important for preserving platform stability. In a repository analysis such as ours, this can coincide with a larger share of bot-generated activity, so commit volume should be interpreted as a mix of human development and automation rather than a direct proxy for manual effort.

Over time, automation appears to have become part of the framework workflow, including high-volume merges and rebases, dependency updates, pre-release validations, and consistency checks. As a result, the bars in the 2020s reflect not only higher activity but also a larger share of bot-mediated work. This has two practical implications. Automated pipelines likely improve throughput and consistency, helping preserve API stability and platform coherence. Commit totals are therefore no longer a straightforward proxy for human effort, since a non-trivial portion of the observed volume is automated. In this setting, external contributors may face a higher barrier to visible upstream impact, because much of the operational flow is mediated by internal tooling and automated accounts.

The Framework column in Table 4 illustrates the extreme centralization of this layer. Google dominates with 1,661,391 commits (90.21% of total framework commits), while independent developers contribute 119,891 commits (6.51%). The 13.9-fold ratio between Google and independent developers in framework development reflects Google's tight control over platform coherence and API stability, where internal automation and corporate pipelines play a pivotal role in the daily management and evolution of the platform.

4.3. Device contributions

The device layer shows the tightest coupling between centralization and automation. The series begins in 2021 with small volumes and then jumps clearly in 2022, where the annual total reaches 6.8k commits, representing the peak activity in this domain. Activity declines from that peak in 2023 and continues to decrease in 2024 and 2025. The aggregate split is about 56.9% human and 43.1% bot, making devices the only domain among the three where bot automation approaches half of all commits (see Fig. 8).

There are clear reasons for this configuration. Device code reflects hardware variability, security requirements, and release deadlines. Managing that space without a tight automation backbone is unrealistic. Automated pipelines ensure firmware constraints, driver updates, and platform changes move in lockstep. The trade-off is minimal visible upstream participation from external companies. Many manufacturers maintain private forks or parallel repositories, optimizing for specific hardware and product cycles without being bound by mainline cadence. That choice reduces their footprint in public repositories but increases speed and control on their side.

The Devices column of Table 4 shows that Google is the near-exclusive contributor with 14,520 commits (98.10% of total device commits), with independent developers contributing only 281 commits (1.90%). The nearly complete absence of any other companies and independent groups reflects how device-level development is tightly controlled through Google-managed accounts, many of which are automated systems. This extreme concentration underscores how internal automation and centralized workflows have become the backbone of device development. Few external organizations contribute directly to public device repositories; most hardware manufacturers maintain private repositories instead to preserve product differentiation and accelerate their release cycles.

4.4. Code churn analysis: Beyond commit counts

While commit counts provide valuable insights into participation frequency, they do not capture the magnitude of code changes. A single commit might add or modify just a few lines, or it could represent thousands of lines of new functionality. To address this limitation and provide a more complete picture of contribution impact, we analyzed code churn (the total lines of code added and removed) across all domains and contributors.

Table 5 presents churn metrics for the top contributors, revealing differences in contribution intensity. Independent developers, despite leading in total commit count, show substantial churn per commit (approximately 1034 lines per commit on average), indicating that their frequent commits often contain meaningful code changes. In contrast, Google exhibits even higher churn per commit (approximately 2949 lines per commit), reflecting substantially larger changes that often involve framework refactoring, feature additions, and integration work across multiple systems.

The churn analysis reveals several important patterns. First, hardware-focused contributors (Intel, AMD, NVIDIA) show the highest lines-per-commit ratios, with Intel reaching 1306 lines per commit while AMD and NVIDIA average 282 and 1192 respectively. This reflects the nature of driver and hardware abstraction layer development, where individual commits frequently involve substantial code additions for new hardware support or major refactoring efforts. Second, the correlation between commit count and total churn is strong (Spearman's $\rho = 0.91$, $p < 0.001$), indicating that commit counts are a reasonable proxy for contribution magnitude, though contribution patterns vary across organizations. Some contributors favor many small, incremental commits, while others prefer fewer but more substantial changes.

Domain-specific churn patterns also reveal strategic differences. In the kernel domain, where independent contributors dominate by commit count (70%), they account for a minority of code churn (approximately 24%), indicating that while they contribute frequently, individual commits are typically smaller in magnitude. In the framework domain, Google's dominance extends to churn metrics, accounting for over 97% of total lines changed, reinforcing the centralized nature of framework development.

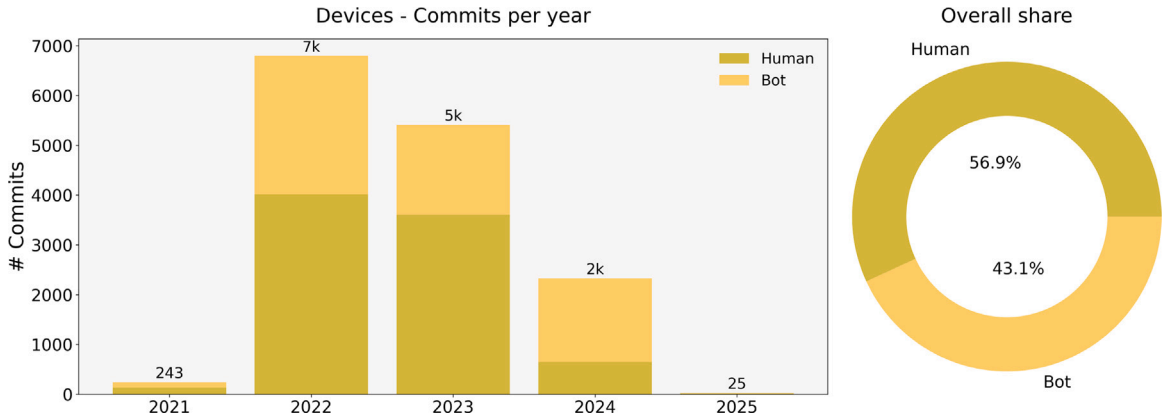

Fig. 8. Annual evolution of human and bot commits in the device domain, and overall share.

Table 5
Code churn metrics for top contributors (2005–2026). Note: Only commits with measurable code churn (additions or deletions) are included.

Contributor	Total commits	Total churn (M lines)	Lines/Commit
Google	1,779,676	5249.3	2949
Others	1,382,187	1429.8	1034
Intel	137,932	180.1	1306
Red Hat	83,110	64.3	773
Google/AOSP	54,009	19.8	367
AMD	49,905	14.1	282
Huawei	38,270	0.9	23
IBM	33,487	9.3	276
NVIDIA	18,951	22.6	1192
Arm	15,789	2.2	142

These churn metrics complement commit counts by revealing contribution intensity and helping distinguish between high-frequency incremental work and less frequent but more substantial changes. The consistency between commit-based and churn-based rankings for top contributors validates that commit counts serve as a reasonable proxy for overall contribution magnitude, while also highlighting cases where contribution patterns diverge from simple frequency measures.

Fig. 9 provides a visual comparison of churn patterns. The left panel shows total code churn (lines added plus removed) for top contributors, illustrating the absolute magnitude of code changes. The right panel displays churn intensity (lines per commit), revealing differences in contribution style. Hardware-focused companies (Intel, NVIDIA) and independent developers show moderate to high intensity, reflecting substantial per-commit changes, while Google's pattern varies across domains depending on the mix of human-driven development and automated integration work.

5. Discussion

The evolution of corporate participation in Android development over the past decade reveals shifts in how the ecosystem operates and who drives its technical direction. When we compare our current findings with the preliminary study conducted by León et al. (2014) in 2014, several transformative patterns emerge that reflect broader changes in the technology industry and open source governance models.

5.1. Response to research questions

This section summarizes the answers to our main research questions (see Section 1), integrating the findings from our analysis and discussing the underlying reasons behind the observed trends.

RQ1: Automation and contribution dynamics

Automation has grown substantially across Android's technical layers over the past decade, though its penetration is uneven. The kernel remains largely human-driven, while the framework and especially the device layer have integrated automated pipelines as a core part of their daily workflows. This asymmetry reflects the distinct governance and operational demands of each layer.

RQ1.1: How has the degree of automation in Android development changed over time?

Our analysis reveals that automation has become increasingly prominent in Android development over the last decade, though the degree varies across technical domains. This transformation reflects both the growing complexity of the Android platform and Google's strategic investment in automated workflows to maintain consistency and reliability.

RQ1.2: What is the current balance between human and automated (bot) contributions across different technical domains?

The current balance varies dramatically across Android's technical layers. The kernel layer remains predominantly human-driven, with approximately 95.7% of commits from individual developers and 4.3% from bots, reflecting its community driven-nature and close ties to the Linux ecosystem. In contrast, the framework domain shows higher automation, with bots responsible for about 25.0% of all commits, indicating that automated pipelines for code merging, validation, and maintenance are now integral to daily workflows. The device layer exhibits the highest degree of automation, where bots account for 43.1% of all commits. This shift reflects the necessity of automation for handling integration, testing, and release management efficiently at scale, though it also means less visible individual human work in these upper layers.

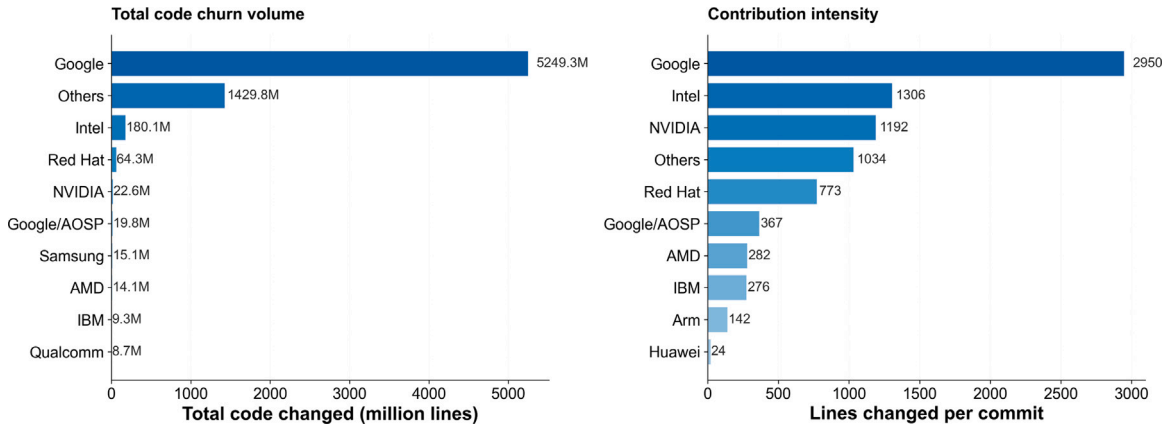

Fig. 9. Code churn analysis for top contributors.

RQ2: Developer typology and ecosystem distribution

Contribution patterns differ sharply across Android's layers. Independent developers have significantly expanded their presence in the kernel since 2014, while Google has maintained near-total control over the framework and device domains. Hardware manufacturers have progressively retreated from public repositories, reshaping the ecosystem's balance toward a community-driven kernel and a corporate-controlled upper stack.

RQ2.1: What is the relative participation of independent developers and private companies in the main layers of the Android ecosystem?

Current contribution patterns reveal differences across Android's technical layers. Independent developers and smaller organizations dominate kernel development, accounting for nearly 70% of commits in that domain. This reflects the kernel's nature as part of the broader Linux ecosystem, which encourages open collaboration and makes centralized control difficult. In contrast, Google remains the dominant player in the framework and device layers, with contribution shares of approximately 90.21% and 98.10%, respectively. This disparity reflects each layer's distinct character: the kernel benefits from broad community participation, while the upper layers require tighter control to maintain platform coherence and compatibility through both human and automated activity.

RQ2.2: How has this distribution evolved since previous studies?

The distribution of contributions has changed compared to the 2014 baseline León et al. (2014). The most notable change is the dramatic rise of independent developer participation in the kernel, which has increased from approximately 40% to nearly 70% of commits. Meanwhile, many hardware manufacturers, including Samsung, Nokia, and Motorola, have largely withdrawn from upstream contributions in favor of maintaining private forks and proprietary modifications. This strategic shift allows these companies to differentiate their products and move faster in competitive markets, but also reduces their visible footprint in public repositories. Google's dominance in framework and device layers has remained stable, though the company has adopted a more collaborative stance in kernel development, allowing the broader open-source community to lead evolution in that domain.

RQ3: Corporate participation and evolution

Google's dominance in the upper layers of Android has remained essentially unchanged, but the broader corporate landscape has transformed considerably. Hardware manufacturers that were visible contributors a decade ago have largely stepped back, while independent developers have filled much of the gap in kernel development. Corporate engagement has shifted from direct upstream contribution toward proprietary forks and private integration pipelines, consolidating a hybrid governance model where community and corporate actors operate in largely separate but complementary spheres.

RQ3.1: Which companies currently play the most significant roles in Android development?

Google remains the dominant corporate contributor to Android development, particularly in the framework and device layers, where it accounts for over 90% and 98% of commits, respectively. This dominance reflects Google's strategic control over platform coherence, compatibility, and the development tools ecosystem. Beyond Google, the landscape is fragmented. Independent developers and smaller organizations collectively represent the second-largest contributor group, especially in kernel development, where they account for nearly 70% of commits. Other notable corporate participants include companies like Microsoft (0.29% of recent commits), whose modest but strategic engagement reflects broader industry shifts toward embracing open-source collaboration.

RQ3.2: Are there emerging contributors compared to a decade ago?

Comparison with the 2014 baseline reveals shifts in the contributor landscape. Major hardware manufacturers such as Samsung, Nokia, Apple, Sony Ericsson, and Motorola, who were actively contributing to official Android repositories a decade ago, have largely withdrawn from upstream contributions. Meanwhile, the most notable emerging contributor category is independent developers, whose participation in kernel development has surged from approximately 40% to nearly 70% of commits. Microsoft's participation, though numerically modest, represents a shift for a company that historically viewed open source as a threat, now engaging strategically to support its Azure cloud platform and other initiatives.

RQ3.3: How have corporate engagement patterns shifted over time?

Corporate engagement patterns have fundamentally transformed over the past decade. The shift reflects a strategic reorientation: many hardware manufacturers have moved toward maintaining private forks and proprietary modifications, allowing them to differentiate products in competitive markets while reducing visible participation in public repositories. This strategic withdrawal coincides with the rise of independent developer contributions, particularly in the kernel domain, where

community-driven development has proven remarkably effective for ensuring stability and innovation. Google's role has also evolved; while maintaining absolute control over framework and device layers for platform coherence, the company has adopted a more collaborative stance in kernel development, allowing the broader Linux community to lead. This hybrid governance model, combining community-driven infrastructure with corporate-controlled platform services, appears to be stabilizing and may serve as a template for other large-scale open source projects seeking to balance openness, technical consistency, and commercial viability.

5.2. Comparative insights to prior open-source ecosystems

When comparing our findings for Android with previous studies of OpenStack and other community-driven open source projects (Butler et al., 2019; Zhang et al., 2019), we observe both notable similarities and important differences in patterns of corporate and independent participation.

- **Similarities:**
  - There is a clear concentration of contributions to core components by one or a few leading companies.
  - Independent contributors maintain a persistent presence in infrastructure or foundational layers.
  - Corporate participation tends to fluctuate in response to product cycles and changing company strategies.
- **Differences:**
  - Android exhibits a sharper separation between a community-driven kernel and corporate-controlled platform and device layers, with extensive automation and integration managed by Google. This results in a more centralized pattern of contributions in the upper layers.
  - In OpenStack, the distribution of contributions among leading firms is more balanced, and the governance model actively distributes control across multiple corporations. This multi-polar governance structure can mitigate the level of centralization observed in Android and promote a more diverse ecosystem.

These comparisons suggest that the transferability of our findings from Android to other open source projects depends on several key factors: the architecture and modularity of the platform (such as the separation between infrastructure and application layers), the technical governance model (centralized versus federated or distributed decision-making), and the dominant business model (device/services versus cloud or platform-as-a-service).

5.3. Statistical validation

To validate the patterns observed in our descriptive analysis, we applied standard statistical techniques commonly used in empirical software engineering and mining software repositories. Bhattarai et al. (2022) used Mann–Whitney U tests with Cliff's delta to compare repository metrics between papers with and without GitHub repos. Claes et al. (2018) applied Mann–Whitney U to compare emoticon usage ratios across Apache and Mozilla projects. We used these same tests for group comparisons, Spearman's rank correlation for temporal trends, and Gini coefficients for concentration measures (Kitchenham et al., 2017; de Oliveira Neto et al., 2019).

Domain-specific contribution differences. Mann–Whitney U tests are appropriate for comparing two independent groups when the data are ordinal or non-normally distributed, as demonstrated in studies of repository attributes (Bhattarai et al., 2022) and emoticon usage patterns in OSS issue trackers (Claes et al., 2018). In our setting, these tests confirm that contribution patterns differ across domains between Google and independent developers. In the kernel domain, independent developers contribute more commits than Google ($U = 130$, $p < 0.001$), while in the framework domain Google is dominant ($U = 320$, $p < 0.001$). These results support the descriptive finding that the kernel is more community-driven, whereas the framework is more centrally controlled.

Temporal trends. Spearman's rank correlation is widely used in empirical software engineering to assess monotonic relationships without assuming normality (de Oliveira Neto et al., 2019; Kitchenham et al., 2017). Here, it shows temporal trends in contribution patterns, with Google ($\rho = 0.70$, $p < 0.001$) exhibiting a moderately strong association between year and total commits, while independent contributors show a weaker trend ($\rho = 0.30$, $p = 0.125$). This suggests that Google's contributions have followed a more consistent temporal trajectory, while independent contributions show greater year-to-year variability.

Contribution concentration. To quantify concentration and inequality in contribution distributions, it is common to report inequality measures such as the Gini coefficient, which complements hypothesis tests by describing how unevenly activity is distributed (de Oliveira Neto et al., 2019; Barros et al., 2021). The kernel domain shows relatively high inequality (Gini = 0.8733), but this is consistent with a broad contributor base in which most developers contribute modestly and a small number of contributors are highly active. The framework domain shows even higher concentration (Gini = 0.9499), consistent with Google's dominance. The devices domain exhibits moderate inequality (Gini = 0.4810), although this mainly reflects the fact that most contributors in that domain are Google-affiliated and have more similar activity levels.

Churn-commit correlation. Spearman's correlation is also appropriate for assessing the relationship between two skewed quantitative metrics such as commit count and code churn (Kitchenham et al., 2017; de Oliveira Neto et al., 2019). The strong correlation between commit count and code churn ($\rho = 0.91$, $p < 0.001$) indicates that commit counts are a reasonable proxy for contribution magnitude, while still capturing a different aspect than churn.

Fig. 10 visualizes these key statistical findings across three complementary perspectives. The left panel shows the Mann–Whitney U test results comparing Google versus independent contributors across domains, clearly demonstrating the domain-specific differences in contribution patterns. The center panel presents the Spearman rank correlations for temporal trends, revealing strong and consistent growth patterns for Google ($\rho = 0.7041$, $p < 0.001$) while independent contributors show weaker variability ($\rho = 0.2972$, $p = 0.125$, not statistically significant). The right panel displays the Gini coefficients quantifying contribution concentration across domains, with values indicating moderate to very high inequality depending on the technical layer.

5.4. Limitations and threats to validity

Our measures are subject to several limitations. Regarding construct validity, commit counts do not capture effort, complexity, or impact, and automation can inflate integration volumes. To mitigate this, we also report churn and top directories, and we suggest complementary analyses of issues, reviews, and effective code changes. With respect to internal validity, corporate classification via email domains can be inaccurate due to aliases and shared accounts. This is partially addressed by implementing alias rules and manual verification for top authors, although some residual uncertainty remains. In terms of external validity, generalization to other ecosystems depends on governance, platform architecture, and business models. Comparisons with

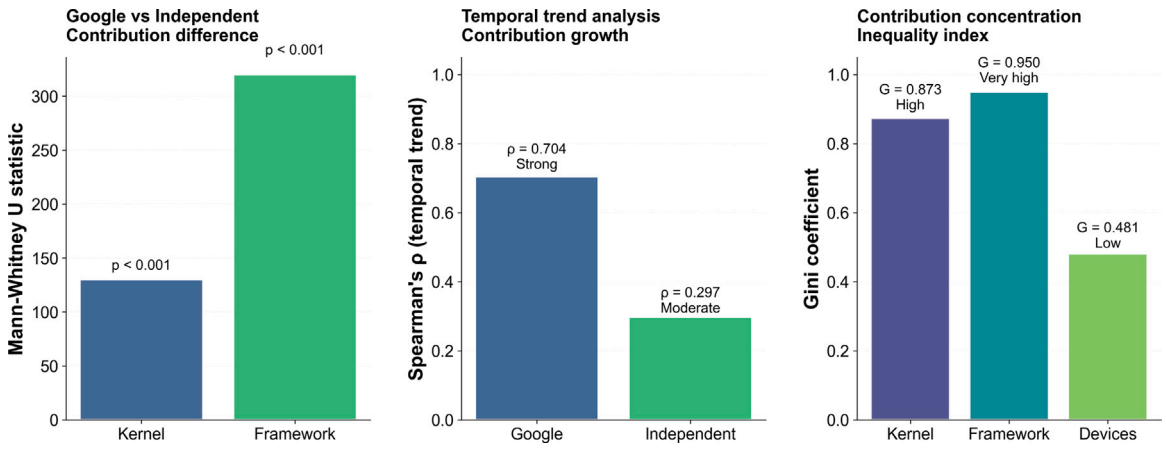

Fig. 10. Statistical validation: key differences between Google and independent contributions.

OpenStack and mobile platform studies suggest transferability under certain conditions, but replication is still necessary. From an ethics and privacy perspective, individual names are not listed in tables and figures; instead, aggregated affiliations are reported. Although the data are public, minimization principles are followed for scientific use, and individuals are not reidentified beyond affiliation-level aggregation.

6. Conclusions

This study of over 3.6 million commits from Android repositories highlights a transformation in the ecosystem's development dynamics over the last two decades. The results show that the Android project has evolved into a hybrid model of open source, where the balance between human and automated contributions varies greatly across technical domains.

The kernel remains the most open and community-driven layer. Here, independent contributors and smaller organizations are responsible for the majority of commits, with automation playing only a supporting role. The data show that around 95.7% of kernel commits are made by humans, and 4.3% by bots, reflecting a collaborative model where technical expertise and upstream coordination are key. This layer continues to attract a broad set of actors, and corporate influence is distributed rather than concentrated.

In contrast, the framework and device domains are characterized by high levels of centralization and automation. In the framework, Google is responsible for the vast majority of commits, and automated systems now account for approximately one quarter (25.0%) of all activity. The device layer exhibits even higher automation levels, with bots responsible for 43.1% of device commits, making automation nearly as prominent as human contributions. In both domains, the top contributor rankings are overwhelmingly occupied by Google-managed accounts, many of which are automated, and there is minimal visible participation from other companies or independent developers. This reflects a workflow where integration, testing, and large-scale maintenance are handled through internal pipelines and automated processes.

These patterns have important implications. In the kernel, commit counts remain a reasonable proxy for human effort, and the distributed nature of contributions supports resilience and innovation. In the upper layers, however, automation has become essential for managing complexity, ensuring platform coherence, and delivering at scale. As a result, raw commit numbers must be interpreted with caution, since a substantial proportion of the activity is now driven by bots rather than human developers.

From a corporate perspective, the landscape has shifted from a scenario where several large companies and independent contributors coexisted to one where Google's strategic control is reinforced by automation, especially in the framework and device layers. Many traditional hardware manufacturers have shifted their focus to private forks or downstream adaptations, reducing their visible footprint in the official repositories.

Overall, the Android development model now combines community-driven infrastructure with corporate-controlled platform and device services. Automation has enabled greater scale, reliability, and speed, but has also concentrated operational control. This hybrid governance structure may serve as a template for other large-scale open source projects seeking to balance openness, technical consistency, and commercial sustainability in an increasingly automated world.

CRediT authorship contribution statement

Sergio R. Montes-León: Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization. **Antonio J. Romero-Barrera:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **David Moreno-Lumbreras:** Writing – review & editing, Writing – original draft, Investigation. **Hernán Montes-León:** Writing – original draft, Investigation.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4.1 to revise grammar and spelling.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was partially funded by the Spanish government as part of the Dependentium project, grant number PID2022-139551NB-I00.

Reproducibility and data availability

Data will be made available on request.

The specific data presented in this paper (annotations and final results), along with the software for producing results, are available in a reproduction package (Montes-León et al., 2025).

References

AboutCode, 2024. AI-generated code search. URL https://github.com/aboutcode-org/ai-gen-code-search. Tools to detect and identify AI-generated code in repositories.

Ampatzoglou, A., Michou, O., Stamelos, I., 2013. Building and mining a repository of design pattern instances: Practical and research benefits. Entertain. Comput. 4 (2), 131–142.

Amreen, S., Mockus, A., Zaretzki, R., Bogart, C., Zhang, Y., 2020. ALFAA: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems. Empir. Softw. Eng. 25 (2), 1136–1167.

Android Developers, 2023. Approximate location. URL https://developer.android.com/codelabs/approximate-location. (Accessed 28 March 2026).

Android Developers, 2026a. Permissions updates in android 11. URL https://developer.android.com/about/versions/11/privacy/permissions. (Accessed 28 March 2026).

Android Developers, 2026b. Storage updates in android 11. URL https://developer.android.com/about/versions/11/privacy/storage. (Accessed 28 March 2026).

Android Developers Blog, 2022. Material you: Coming to more android devices near you. URL https://android-developers.googleblog.com/2022/02/material-you-coming-to-more-android.html. (Accessed 28 March 2026).

Android Open Source Project, 2025a. Android git repositories. https://android.googlesource.com. (Accessed 15 September 2025).

Android Open Source Project, 2025b. Architecture overview. https://source.android.com/docs/core/architecture?hl=es-419. (Accessed 23 September 2025).

Android Open Source Project, 2025c. Mainline. URL https://source.android.com/docs/core/ota/modular-system. (Accessed 28 March 2026).

Android Open Source Project, 2025d. Privacy indicators. URL https://source.android.com/docs/core/permissions/privacy-indicators. (Accessed 28 March 2026).

Android Open Source Project, 2026. Material you design. https://source.android.com/docs/core/display/material. (Accessed 26 March 2026).

Bao, L., Lo, D., Xia, X., Wang, X., Tian, C., 2016. How android app developers manage power consumption? an empirical study by mining power management commits. In: Proceedings of the 13th International Conference on Mining Software Repositories. MSR '16, Association for Computing Machinery, New York, NY, USA, pp. 37–48. http://dx.doi.org/10.1145/2901739.2901748.

Barros, D., Horita, F., Wiese, I., Silva, K., 2021. A mining software repository extended cookbook: Lessons learned from a literature review. In: Proceedings of the XXXV Brazilian Symposium on Software Engineering. pp. 1–10.

Bhattarai, P., Ghassemi, M., Alhanai, T., 2022. Open-source code repository attributes predict impact of computer science research. In: Proceedings of the ACM/IEEE Joint Conference on Digital Libraries. http://dx.doi.org/10.1145/3529372.3530927.

Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Sjöberg, J., Mattsson, A., Gustavsson, T., Feist, J., Lönroth, E., 2019. On company contributions to community open source software projects. IEEE Trans. Softw. Eng. 47 (7), 1381–1401.

Callan, J., Krauss, O., Petke, J., Sarro, F., 2022. How do android developers improve non-functional properties of software? Empir. Softw. Eng. 27 (5), 113.

Chidambaram, N., Decan, A., Mens, T., 2025. A bot identification model and tool based on GitHub activity sequences. J. Syst. Softw. 221, 112287.

Claes, M., Mäntylä, M., Farooq, U., 2018. On the use of emoticons in open source software development. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–4.

Codabux, Z., Fard, F., Verdecchia, R., Palomba, F., Di Nucci, D., Recupito, G., 2024. Teaching mining software repositories. In: Handbook on Teaching Empirical Software Engineering. Springer, pp. 325–362.

Cui, K.Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., Salz, T., 2024. The productivity effects of generative AI: Evidence from a field experiment with GitHub copilot. URL https://mit-genai.pubpub.org/pub/v5iixksv. MIT Generative AI Working Group.

de Oliveira Neto, F.G., Torkar, R., Feldt, R., Gren, L., Furia, C.A., Huang, Z., 2019. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. J. Syst. Softw. 156, 246–267.

Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., Mockus, A., 2020. Detecting and characterizing bots that commit code. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 209–219.

Dueñas, S., Cosentino, V., Robles, G., Gonzalez-Barahona, J.M., 2018. Perceval: software project data at your will. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. pp. 1–4.

Erlenhov, L., de Oliveira Neto, F.G., Scandariato, R., Leitner, P., 2019. Current and future bots in software development. In: 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering. BotSE, IEEE, pp. 7–11.

Fry, T., Dey, T., Karnauch, A., Mockus, A., 2020. A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 518–522.

Herraiz, I., Robles, G., González-Barahona, J.M., 2004. CVSAnalY: una herramienta libre para el análisis de repositorios CVS. IV Jornadas Andal. Softw. Libr. Algeciras.

Khomh, F., Yuan, H., Zou, Y., 2012. Adapting linux for mobile platforms: An empirical study of android. In: 2012 28th IEEE International Conference on Software Maintenance. ICSM, IEEE, pp. 629–632.

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., Pohthong, A., 2017. Robust statistical methods for empirical software engineering. Empir. Softw. Eng. 22 (2), 579–630.

León, S.R.M., Benalcazar, J.D.R., León, H.M., 2014. Estudio preliminar de las empresas que más contribuyen al desarrollo de android. Rev. Eletrônica Sist. Inf. 13 (2).

Mendes, W., Pinheiro, O., Santos, E., Rocha, L., Viana, W., 2022. Dazed and confused: Studying the prevalence of atoms of confusion in long-lived java libraries. In: 2022 IEEE International Conference on Software Maintenance and Evolution. ICSME, pp. 106–116. http://dx.doi.org/10.1109/ICSME55016.2022.00018.

Monperrus, M., 2019. Explainable software bot contributions: Case study of automated bug fixes. In: 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering. BotSE, IEEE, pp. 12–15.

Montes-León, S.R., Romero-Barrera, A.J., Montes-León, H., 2025. Reproduction package for the paper Exploratory Study of Corporate Contributions in Android Development through Repository Analysis. Zenodo, http://dx.doi.org/10.5281/zenodo.17199815, Data set.

Poncin, W., Serebrenik, A., Van Den Brand, M., 2011. Process mining software repositories. In: 2011 15th European Conference on Software Maintenance and Reengineering. IEEE, pp. 5–14.

Possemato, A., Aonzo, S., Balzarotti, D., Fratantonio, Y., 2021. Trust, but verify: A longitudinal analysis of android oem compliance and customization. In: 2021 IEEE Symposium on Security and Privacy. SP, IEEE, pp. 87–102.

Robles, G., Koch, S., Gonzalez-Barahona, J.M., 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In: Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems. RAMSS, Edinburgh, Scotland, UK, pp. 51–56.

Sharma, H., 2021a. Android Open Source Project: a Study of Contribution Analysis & Degree of Openness in Governance (Ph.D. thesis). Chalmers University of Technology, URL https://odr.chalmers.se/bitstreams/69364e59-da36-4d56-b25c-c883958b2c08/download.

Sharma, H., 2021b. Android open source project: A study of contribution analysis & degree of openness in governance.

Tabosa, D., Pinheiro, O., Rocha, L., Viana, W., 2024. A dataset of atoms of confusion in the android open source project. In: Proceedings of the 21st International Conference on Mining Software Repositories. pp. 520–524.

Wessel, M., De Souza, B.M., Steinmacher, I., Wiese, I.S., Polato, I., Chaves, A.P., Gerosa, M.A., 2018. The power of bots: Characterizing and understanding bots in oss projects. Proc. ACM Human-Comput. Interact. 2 (CSCW), 1–19.

Yamaguchi, e.a., 2016. Benefitting from contributions to the android open source community. Asia-Pac. J. Bus. Adm. 15 (5), 239–250, URL https://www.jstage.jst.go.jp/article/abas/15/5/15_0160825a/_article.

Yetiştiren, B., Özsoy, I., Ayerdem, M., Tüzün, E., 2023. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. arXiv preprint arXiv:2304.10778.

Zhang, Z., Zhang, H., Qian, Z., Lau, B., 2021. An investigation of the android kernel patch ecosystem. In: 30th USENIX Security Symposium. USENIX Security 21, USENIX Association, pp. 3649–3666, URL https://www.usenix.org/conference/usenixsecurity21/presentation/zhang-zheng.

Zhang, Y., Zhou, M., Mockus, A., Jin, Z., 2019. Companies' participation in oss development–an empirical study of openstack. IEEE Trans. Softw. Eng. 47 (10), 2242–2259.