Forensic Science International: Digital Investigation 56 (2026) 302046

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS EU 2026 - Selected Papers from the 13th Annual Digital Forensics Research Conference Europe

Inside the black box: In-depth analysis of geolocation mechanisms in android mobile devices

Samuele Mombelli *, Thomas R. Souvignet

School of Criminal Justice, Faculty of Law, Criminal Justice and Public Administration, University of Lausanne, 1015, Lausanne, Switzerland

ARTICLE INFO

Keywords:
Geolocation traces
Android
Google Mobile Services (GMS)
Fused Location Provider (FLP)
Trace generation mechanisms
Reverse engineering

ABSTRACT

Geolocation traces extracted from mobile devices are increasingly used as critical evidence in forensic investigations, offering insights into user activity and physical presence. Yet, the opaque nature of geolocation processes—especially those relying on proprietary components—creates major challenges in assessing their provenance, accuracy, and reliability, and increases the risk of misinterpretation. This work investigates the internal mechanisms implemented by Android's Fused Location Provider (FLP)—the core geolocation framework used by most Android devices today—through reverse engineering, dynamic testing and forensic analysis. It details how multiple sources of location data are fused into location fixes, and how contextual parameters influence geolocation calculations. The study also uncovers a set of previously undocumented local traces of geolocation activity, analyzing their structure, persistence, and forensic potential. Our findings highlight the complexity and adaptive nature of Android's geolocation system and provide a technical foundation for the forensic interpretation of geolocation traces on Android devices.

1. Introduction

Mobile devices have become indispensable in everyday life, generating large volumes of digital traces. Among these, geolocation traces are particularly valuable in forensic investigations, as they can reveal a device's position and movements, and can help reconstruct events or timelines (Amundsen and Ovens, 2017; Berger et al., 2024).

Despite their forensic value, the processes by which such traces are created remain poorly understood. For example, Android devices—which dominate the global smartphone market—rely on a complex mix of positioning technologies, including GNSS, cellular, Wi-Fi, Bluetooth, and onboard sensors (Maghdid et al., 2016). However, their integration and implementation remain largely opaque: multiple studies have described these systems as "black boxes," with limited insight into how geolocation fixes[1] are computed, how accuracy is assessed, or what intermediate data are involved (Moayeri et al., 2019; Spichiger, 2022). This lack of transparency poses a significant challenge for forensic practitioners. As Berger et al. (2024) argue, a geolocation trace cannot be evaluated solely on its presence; understanding its reliability requires knowledge of the mechanisms and conditions that produced it.

In this study, we investigate Android's geolocation internals, with a particular focus on Google Mobile Services (GMS) and its Fused Location Provider (FLP), i.e., the location framework recommended by Google for Android application development (Google, 2023) and widely adopted in flagship applications such as Google Maps (Moayeri et al., 2019). We reconstruct the lifecycle of trace generation—from initial requests through provider activation, data retrieval, fusion, and delivery—while identifying the intermediate traces[2] produced along the way. Our analysis addresses two research questions:

RQ1: What internal mechanisms are used by Android devices to compute intermediate and final geolocation fixes, and how do these mechanisms influence the resulting traces?
RQ2: What local traces are produced throughout the geolocation process? What governs their creation and persistence?

To answer these questions, we combine reverse engineering of GMS components, validation testing, and forensic analysis of newly

* Corresponding author.
E-mail addresses: samuele.mombelli@unil.ch (S. Mombelli), thomas.souvignet@unil.ch (T.R. Souvignet).

[1] We use the term *geolocation fix* to refer to a device's computed position—central location and horizontal accuracy, optionally with a timestamp—produced as a final result or by a single provider (e.g., network, GNSS).

[2] In the context of this study, we use the term *intermediate traces* to refer to the local traces generated during the internal stages of the geolocation process.

https://doi.org/10.1016/j.fsidi.2026.302046

discovered local traces. As a result, the contributions of this study are threefold:

- Documentation of FLP mechanisms: We reconstruct the geolocation process on Android, focusing on the sources of geolocation data and the computation of fixes.
- Discovery of intermediate traces: We identify and characterize a set of previously undocumented local traces generated by FLP during the geolocation process.
- Scientific basis for forensic inference: We provide a technical foundation for interpreting Android geolocation traces, supporting forensic inference and a more informed assessment of uncertainty and potential error sources.

2. Background and related work

This section introduces key concepts and challenges related to geolocation traces on mobile devices (Section 2.1), summarizes the main geolocation sources and techniques used in modern mobile systems (Section 2.2), and reviews Android geolocation APIs and related prior work (Section 2.3).

2.1. Geolocation traces: definitions and challenges

In the context of mobile forensics, *location-related evidence* is defined as "data generated by the operation of a mobile device as a function of its geographical location" (Casey et al., 2020, p. 2). Prior work by Spichiger (2022) has distinguished between direct traces (*localisations*), which result from the device's positioning process, and indirect traces that simply reflect location (*location-related features*). Following this distinction, this paper uses the term *geolocation traces* to describe localisations produced by geolocation mechanisms, which are positioning mechanisms that estimate the geographic location of a specific device or system.

From a forensic perspective, interpreting such traces and evaluating their uncertainty is a challenging but necessary task (Froklage, 2024; Ryser et al., 2024). Errors may arise at every stage of a geolocation trace's lifecycle, from its initial production to interpretation and integration into event reconstruction (Berger et al., 2024). Since such errors propagate from the moment a trace is created, understanding trace generation mechanisms is essential. However, many positioning methods operate as black boxes (Macarulla Rodriguez et al., 2018; Spichiger, 2022), making high-level inference alone insufficient. To support reliable interpretation, it is necessary to examine the internal logic of geolocation services—their algorithms, input dependencies, and control flows—in order to understand how traces originate and to reduce uncertainty about both their generation and the circumstances under which the observed information may be produced, supporting their use as forensic evidence (Berger et al., 2024).

2.2. Geolocation sources and techniques

Mobile devices rely on multiple technologies to determine their position, often combining them to improve availability, accuracy, and energy efficiency (Maghdid et al., 2016; Dabove and Di Pietra, 2019). The main sources and techniques can be grouped as follows:

2.2.1. Global navigation satellite systems (GNSS)

GNSS (e.g., GPS, GLONASS, Galileo, BeiDou) provide absolute positioning by measuring ranges to at least four satellites. Assisted GNSS (A-GNSS) can be employed to reduce energy use and time-to-first-fix by outsourcing orbital data and initial estimates to a network server. Typical smartphone GNSS accuracy is 5–10 m in open sky, but degrades significantly in urban canyons or indoors (Maghdid et al., 2016; Merry and Bettinger, 2019; Casey et al., 2020).

2.2.2. Wi-Fi positioning

Wi-Fi-based positioning uses nearby Access Points (APs) to determine location, typically via fingerprinting or trilateration, and is especially effective in closed environments. Most modern systems rely on large crowdsourced databases that map AP identifiers to geographic coordinates. Typical accuracy ranges from a few meters in dense indoor environments to tens of meters in sparse areas, but depends heavily on AP density and database freshness (Moayeri et al., 2019; Casey et al., 2020).

2.2.3. Cellular positioning

Cell-based methods estimate the device's position based on the serving base station or multiple neighboring towers. Techniques range from simple Cell-ID, where the device is assumed to be within the coverage of a tower, to timing and signal-strength based methods, such as fingerprinting, Time Difference of Arrival (TDOA), or Angle Of Arrival (AOA). The accuracy of these methods can vary widely: from 20 to 200 m in dense urban areas to kilometers in rural ones (Maghdid et al., 2016; Spichiger, 2022).

2.2.4. Bluetooth and proximity signals

Bluetooth technology can be used for short-range wireless communication and localization. Bluetooth Low Energy (BLE) is particularly relevant, allowing smartphones to detect "beacons" via Received Signal Strength (RSS) measurements and use these values for short-range location estimation. The accuracy of BLE positioning generally ranges between 1 and 10 m (Obeidat et al., 2021; Spichiger, 2022).

2.2.5. Inertial sensors

Smartphones are equipped with inertial sensors (e.g., accelerometers, gyroscopes, magnetometers), which can be used to track movement and orientation. These data can be used to infer relative movement and update location estimation through techniques like dead-reckoning, which estimates the device's position from the last known location and sensor-detected motion when external signals are temporarily unavailable (e.g., in a tunnel). These methods degrade quickly due to drift and noise but can complement other sources (Maghdid et al., 2016).

2.2.6. Hybrid and fused approaches

Modern smartphones rarely rely on a single source; instead, they often combine GNSS, Wi-Fi, cellular, Bluetooth, and sensor data. The fusion of multiple techniques can improve accuracy, reduce fix latency, and be more resilient and adapt to the current context (e.g., indoors vs. outdoors) (Maghdid et al., 2016; Moayeri et al., 2019).

2.3. Geolocation on android devices

On Android, access to these sources of location data is abstracted mainly through two API layers. The Android Framework Location APIs (AOSP, 2025a), maintained as part of the Android Open Source Project (AOSP), are openly accessible, but have been increasingly replaced by Google's proprietary Location and Context APIs (Google, 2025e), within Google Mobile Services (GMS), introduced in 2013 (Google, 2013, 2023). At the core of GMS is the Fused Location Provider (FLP), which combines multiple data sources (e.g., GNSS, Wi-Fi, and cell towers) and fuses them while balancing accuracy, power consumption, and latency (Google, 2025b). While widely adopted and pre-installed on most Android devices, the internals of FLP are closed-source and largely undocumented (Moayeri et al., 2019; Google, 2025a, 2025c).

Both APIs allow Location-Based Services (LBS) to retrieve multiple types of geolocation fixes (e.g., current location, last known location, location updates) through the submission of location requests, which specify the desired type of information and include parameters such as priority (i.e., balance between power consumption and accuracy), granularity (i.e., precision level of the final fix), update frequency, and expiration timeout (Google, 2024a,b; AOSP, 2025c).

Multiple forensic studies have examined geolocation traces on Android devices, mainly focusing on the detection and extraction of artifacts from system files, caches, or third-party apps (see, for example, Maus et al. (2011); Kröger and Creutzburg (2012); Bays and Karabiyik (2019); Salamh et al. (2021)). These studies provide valuable sources of traces but generally treat geolocation mechanisms as black boxes, focusing on what data can be recovered rather than how they are produced. To the best of our knowledge, no prior work has studied in detail the internal mechanisms underlying the generation of geolocation traces within Android's location stack.

3. Methodology

This section outlines the methodology we developed to study the inner workings of the FLP. We first established a controlled test environment to enable reproducible reverse engineering of GMS components. We then applied static and dynamic analysis, combined with network interception, to examine the classes and methods involved in geolocation determination. In the process, we identified undocumented intermediate traces and developed ad-hoc parsers to extract them. Finally, we validated our findings via targeted experiments, testing the reconstructed mechanisms, examining the temporal evolution of these traces, their presence in real-world forensic acquisitions, and the ability of existing forensic tools to detect and parse them.

3.1. Test environment

We performed our experiments primarily on a rooted Google Pixel 6a running Android 15 stock (build AP4A.241205.013.A2) with an active SIM card and enabled location services. This device was selected for both practical and methodological reasons: it was readily available, fully supported rooting and instrumentation, and ran an up-to-date, largely unmodified Android distribution with timely GMS updates. As a Google reference device, it provided a representative baseline for current Android devices while minimizing manufacturer-specific customizations. We set up a man-in-the-middle (MITM) proxy (Cortesi et al., 2010) to capture cleartext network traffic, and we deployed the Frida server (Vadla Ravnås, 2025) to support dynamic analysis during the reverse engineering and validation phases.

To issue standardized location requests to the FLP, we implemented a lightweight test application based on the Android Platform Samples toolkit (AOSP, 2025b), available at Mombelli (2025). The app wraps the `getCurrentLocation` and `getLastLocation` methods of the `FusedLocationProviderClient` interface (Google, 2024a,b), allowing us to vary parameters such as priority and granularity while collecting the resulting fixes and attributes.

3.2. Reverse engineering of GMS

We initially focused our analysis on location-related namespaces within the `com.google.android.gms` package (e.g., `location.network`, `location.fused`, `location.provider`). We performed static reverse engineering primarily on GMS version 25.06.32, which served as our reference implementation. To assess GMS stability, we also extracted and decompiled major releases received as automatic updates during the study period (25.06.32 to 25.30.31), using JADX (skylot, 2025) and Bytecode Viewer (Konloch, 2025), and cross-checked for changes in classes of interest. We conducted dynamic analysis and validation tests across multiple GMS versions in this range to confirm that the observed mechanisms and behaviors persisted across updates.

We analyzed the geolocation process in three phases: (1) network-based methods (cellular and Wi-Fi), (2) GNSS-based methods, and (3) the fusion of multiple intermediate fixes. For each phase, we started by identifying candidate classes using two complementary criteria: (1) classes within the `com.google.android.gms.location` namespace, whose names are largely unobfuscated and thus directly indicative of their role, and (2) other classes containing relevant strings (e.g., "location," "location request," "network location," "gnss," "fused location"). We then conducted a preliminary review of these classes and their methods, expanding the scope to related classes (e.g., imported, instantiated, or exchanged as parameters/returns). Next, we applied dynamic analysis to trace how input data (e.g., Wi-Fi scans, cell tower observations, GNSS measurements) were transformed into geolocation fixes and to capture relevant stack traces.

We then examined each stack trace of interest in detail, focusing on how data were passed, processed, and returned within the key classes and methods. For complex logics and algorithms, we consulted two Large Language Models (LLMs) to support exploratory interpretation: ChatGPT (OpenAI, 2025) and GraphCodeBERT (Guo et al., 2021). These tools were used only to accelerate initial understanding; all interpretations were manually reviewed against the decompiled code, and no conclusions were drawn solely from LLM output. For critical logic, hypotheses were further validated by reimplementing the operations in Python (Python Software Foundation, 2025) to confirm correctness. These elements were progressively assembled into coherent data flows and decision logics, representing the reconstructed geolocation mechanisms.

We further analyzed the intermediate traces identified during our investigation, focusing on their storage formats, persistence, and content. To this end, we employed several adapted tools, including dfindexeddb (Google, 2025d), ImHex (WerWolv, 2025), and protoc (Google, 2025f). When necessary, we also implemented custom parsers to automate trace extraction.

3.3. Validation

To corroborate our findings and explore operational impacts, we conducted four complementary sets of validation experiments. Due to the high risk of re-identification from geolocation data, we do not release the raw validation dataset; however, we provide aggregated statistics and anonymized data to support reproducibility (see Mombelli (2025)).

3.3.1. Dynamic validation

To corroborate the hypotheses developed during the reverse engineering phase, we hooked 94 methods identified as key nodes of the geolocation stack using Frida. These hypotheses concerned both (1) the reconstruction of geolocation mechanisms, and (2) the generation, population, and semantics of the identified intermediate traces. This allowed us to comprehensively validate execution flows and data propagation under varying request parameters (e.g., priority, granularity), behaviors (e.g., walking, running, driving) and environmental conditions (e.g., indoors, outdoors). The full set of targeted experiments is documented in Mombelli (2025).

3.3.2. Temporal evolution of intermediate traces

To study the lifecycle and the persistence of the discovered intermediate traces, we monitored their creation, evolution, and deletion on four rooted Android devices (Table 1) over 30 days. We kept the devices connected to the Internet and used them in a controlled manner under real-world conditions, with interactions mainly limited to geolocation purposes. To preserve in-the-wild behavior, we did not pin GMS

Table 1
Devices used to monitor the temporal evolution of intermediate traces. GMS versions reflect automatic updates observed during the monitoring period.

Brand and model	Android version	GMS versions
Google Pixel 9a	16	25.31.33 to 25.35.34
Samsung Galaxy A51	13 (One UI 5.1)	25.31.33 to 25.35.34
Xiaomi Redmi Note 11 Pro 5G	11 (MIUI 13.0.10.0)	25.31.33 to 25.35.34
Fairphone 3+	10	25.30.31 to 25.35.34

versions, enabling automatic updates to occur. Additionally, we used MITM Proxy to intercept all network communications, with the aim of capturing requests to Google's remote location APIs and validating the mechanisms of crowdsourced data collection for location enhancement.

3.3.3. Targeted evaluation of forensic tools

We processed the forensic acquisition of our main test device using four leading, up-to-date forensic tools—two commercial (whose names are withheld) and two open-source (i.e., Autopsy (Sleuth Kit Labs, 2025) and ALEAPP (Brignoni, 2025))—to assess their ability to detect and parse the identified intermediate traces.

3.3.4. Operational prevalence of intermediate traces

Using a set of custom parsers developed during the reverse-engineering phase (available at Mombelli (2025)), we analyzed 307 forensic acquisitions of Android devices in which the private application directory `/data/data/com.google.android.gms/` was accessible. These acquisitions, provided by the Vaud Cantonal Police (Switzerland), served to evaluate the operational prevalence of the intermediate traces we identified. The dataset covered a broad device spectrum, spanning Android versions from 7 to 15 and including more than ten vendors (e.g., Samsung, Xiaomi, OPPO, Huawei, Google), ensuring heterogeneity in both hardware and software contexts.

4. Reconstruction of the geolocation process

This section reconstructs how the FLP computes a device's final geolocation fix, as inferred from our empirical analysis and corroborated by our dynamic validation tests. Because FLP is proprietary and largely undocumented, this reconstruction reflects our best interpretation of the results and may be incomplete or imprecise. Further technical details on the underlying calculations, algorithms, and procedures are available at Mombelli (2025) and complement the findings presented here.

4.1. Initialization and location requests

The geolocation process begins with the activation of the *Google Location Manager service*, which instantiates a *Google Location Manager* (responsible for handling requests) and a *Fused Location Provider* (responsible for computing final fixes). Upon creation, the *Fused Location Provider* initializes a *Runtime Cache* that stores the most recent location in memory.

Location computation is handled by a layered location engine that sequentially applies throttling, delaying, and fusion logic. Depending on request parameters and device state, a *Stationary Throttling Engine* may reuse a recent final fix when the device is stationary, while a *Delaying Engine* can temporarily defer recomputation following changes in request settings. The final *Fusion Engine* then combines inputs from multiple location providers to compute a new geolocation fix when required. Once initialized, the system can handle two request types:

- Current Location Request: After permission checks and basic sanitization, a new final fix is computed by the *Fusion Engine* and stored in the *Runtime Cache*.
- Last Location Request: No new computation is triggered; the *Runtime Cache* is queried directly, returning the most recent final fix.

In this process, we identified two notable behaviors relevant for forensic interpretation:

- If a request with a coarse granularity has expired by less than 10 minutes, the last calculated final fix is not discarded. Instead, it is returned with a timestamp refreshed to the current time.
- Requests with coarse granularity are subject to obfuscation: the coordinates of the resulting final fix are perturbed with a random offset (updated hourly), and the horizontal accuracy is downgraded to a minimum of 2000 m.

4.2. Activation of location providers

The first major step in the computation of a new final geolocation fix is the activation of available location providers, coordinated by a *GMS Fusion Scheduler* and a *Location Provider Activator*. This activation depends primarily on the availability of location providers[3] and the priority of the incoming request. The inferred roles of each location provider and the conditions for their activation are summarized in Table 2.

4.3. Network-based geolocation mechanisms

The *Network Location Provider (NLP)* is the first major provider to contribute to geolocation. Its role is to estimate the device's position from surrounding cell towers and Wi-Fi APs. The process follows three main stages:

1. **Data collection:** The system scans nearby Wi-Fi APs and retrieves information about the serving cell tower and its neighbors, storing parameters and timestamps in memory.

Table 2
Overview of location providers available for geolocation.

Name	Inferred role	Activation (if available)
Pressure Provider	Supplies barometric pressure information.	Always.
GMS Weather Provider	Provides contextual weather information.	Always.
Step Detector	Detects user steps to aid in motion estimation.	If priority is HIGH_ACCURACY.
Activity Recognition Provider	Infers user activity (e.g., walking, running, driving).	If priority is HIGH_ACCURACY.
Wi-Fi RTT Continuous Ranger	Measures distances to Wi-Fi APs using Wi-Fi Round Trip Time (RTT) technology.	If priority is HIGH_ACCURACY.
GNSS Provider	GMS provider that supplies location data based on GNSS satellite signals.	If priority is HIGH_ACCURACY.
GNSS Batch Provider	Fallback provider (from Android Location Framework) that delivers GNSS location data in batches.	If priority is HIGH_ACCURACY, and if *GNSS Provider* is not available.
Bluesky Provider	Enhances GNSS accuracy by providing correction data.	If either *GNSS Provider* or *GNSS Batch Provider* is activated.
Network Location Provider (NLP)	Provides location information primarily based on Wi-Fi APs and cell tower data.	If priority is either HIGH_ACCURACY or BALANCED_POWER_ACCURACY.
Network Location Provider Low Power	Lower-power variant of *NLP* that relies exclusively on cell tower data.	If priority is LOW_POWER.
Passive Provider	Receives location updates passively by listening to fixes triggered by other requesters.	If priority if PASSIVE.

[3] Each provider's availability is determined by a combination of factors, including the accessibility of its underlying data sources (e.g., barometric sensor, Wi-Fi connectivity), the device's current state (e.g., detected device speed, GNSS signal quality), and other dynamic conditions (e.g., timing, previously activated providers).

2. **Aggregation:** Wi-Fi and cell scans occurring within a 20-second window are paired into location candidates.
3. **Estimation:** Each candidate produces two independent location estimates, one from Wi-Fi fingerprinting (*Wi-Fi Locator Result*) and one from cell tower data (*Cell Locator Result*). These results are encapsulated in a *Network Location* object, and the most accurate fix is selected.

4.3.1. Cell-based position estimation

Cell-based positioning relies on pre-calculated location fixes associated with the serving cell tower, retrieved either from a local cache (`CFPC1 Cache`, see Section 5) or, if missing, from Google's servers via the private `VoilaTile` API.[4] A request to this API contains identifiers for the target cell tower (Mobile Country Code, Mobile Network Code, Cell ID, plus Location Area Code for GSM cell towers). A response includes, for the queried cell (if known) and for each neighbor enclosed in a map tile of variable size:

1. A `CellPrimaryOnly` estimate: A precomputed geolocation fix based on the current cell (i.e., the queried cell or the neighbor under consideration).
2. Multiple `CellFingerprint` estimates: A list of precomputed geolocation fixes, based on the current cell and multiple subsets of its neighbors.

It's important to note that these intermediate fixes represent the device's position, not the cell tower's position. All central locations are encoded using Google's S2 Geometry format, a spatial indexing system that partitions the Earth's surface into hierarchical cells, each identified by a unique S2 Cell ID encoding both geographic position and level of detail (Google, 2025g). The returned data are decoded by a *Cell Location Calculator*, before being stored in the `CFPC1 Cache`.

As illustrated in Fig. 1a, the cell-based geolocation process follows three steps:

1. **Cache check:** If the serving cell is not present in the `CFPC1 Cache`, a `VoilaTile` request is issued, and the result is cached for future use.
2. **Estimation:** The system compares the current list of neighbors with the neighbor sets in the cached `VoilaTile` data associated with the serving cell tower. If a match is found, the corresponding `CellFingerprint` fix is selected. If not, the system defaults to the fallback `CellPrimaryOnly` fix.
3. **Accuracy adjustment:** The reported horizontal accuracy is bounded to $\geq$100 m before the result is dispatched.

4.3.2. Wi-Fi-based position estimation

Wi-Fi positioning follows a process similar to cell-based estimation. The required geolocation data for detected APs are retrieved from multiple local caches (`FREWLE`[5] `Cache v2/v3` and `VTC Cache`, see Section 5), or, if absent or expired, from Google's `VoilaTile` API. A request includes a list of BSSIDs for the target APs, while a response provides information for each requested AP (if known) and nearby APs (enclosed in a map tile of variable size, also known as FREWLE tile): BSSID, central location (encoded as an S2 Cell), output power, path-loss exponent, elevation, RTT model,[6] associated building and floor. Note that no horizontal accuracy value is included in the response, as these are not precomputed location fixes: the central location reflects the estimated absolute position of each AP. Accuracy is instead computed locally, based on values returned by `VoilaTile` together with the measured RSSI values.

The returned data are decoded by a *Wi-Fi FREWLE Location Manager*, before being stored in the `FREWLE Cache` and `VTC Cache`. Once AP data are available, the positioning process follows four stages, as shown in Fig. 1b:

1. **Cache check:** If the detected APs are not present in cache (`FREWLE Cache v2/v3`, `VTC Cache`), a `VoilaTile` request is issued, and the result is cached for future use.
2. **Preprocessing:** Detected APs are sorted by signal strength; at most the top 20 are used.
3. **Estimation:** Device location, horizontal accuracy, and indoor probability are inferred using three embedded neural networks (packaged with the GMS APK, weights are stored in files `wifi_location.model.uncompressed`, `wifi_accuracy.model.uncompressed`, and `wifi_indoor.model.uncompressed`) taking as input the measured RSSIs and the parameters received from `VoilaTile`. The following constraints, relevant for forensic interpretation, are applied:
   - If a single AP is detected, its coordinates define the central location of the intermediate fix, while accuracy and indoor probability are estimated by the respective models.
   - With 2–3 detected APs spaced more than 300 m apart, the location of the closest AP is returned, with fixed horizontal accuracy (100 m) and indoor probability (0.25).
   - Horizontal accuracy values are finally clamped between 4.2 m and 210 m.
4. **Altitude computation and floor assignation:** The device's altitude and vertical accuracy are derived from AP elevation data and measured RSSI values. The building and floor in which the device is located, both represented by an S2 Cell ID and a unique identifier, are determined by plurality voting among the strongest APs detected.

4.3.3. Crowdsourced location enhancement

Network-based positioning is continuously refined through crowdsourced data contributed by GMS user devices. This process is coordinated via the remote and private `Kollektomat` API,[7] which aggregates environmental and sensor data before uploading them to Google's servers. Our analysis identified three distinct collection mechanisms:

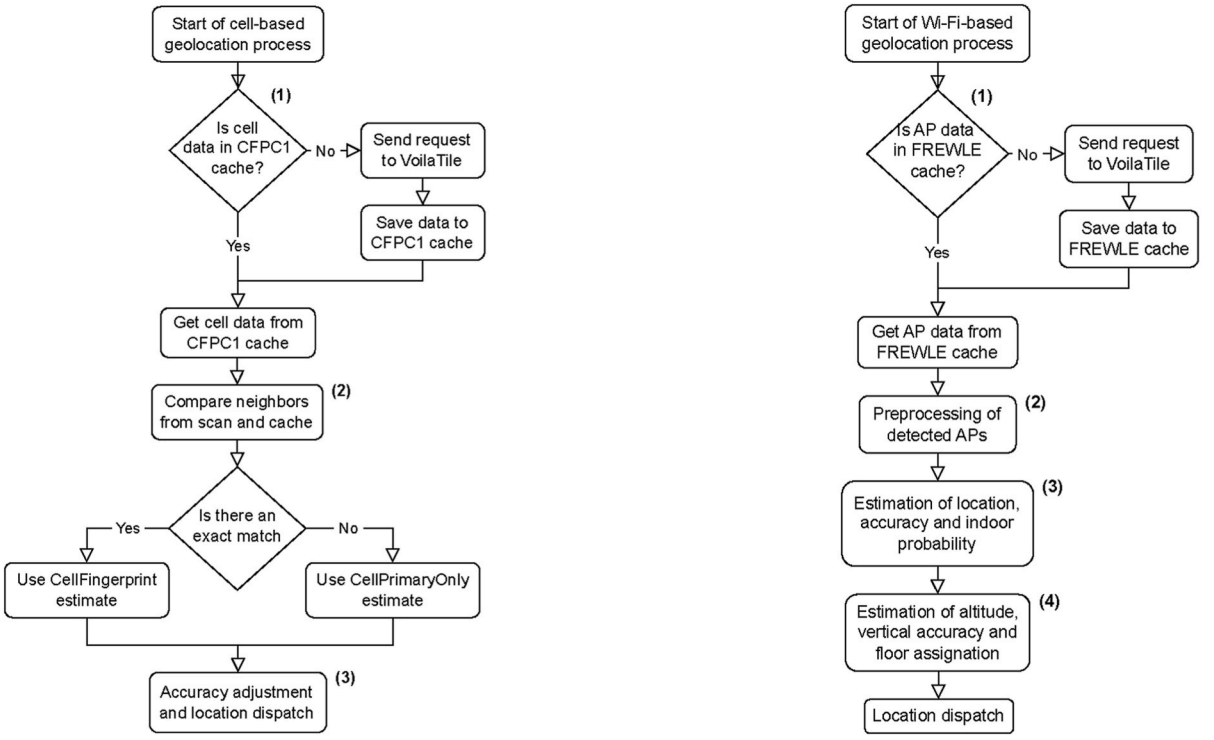
- *Burst Collector*: Actively performs Wi-Fi and cell scans, requests activity recognition results and current final geolocation fixes "on demand." Then, it uploads data immediately. Validation tests showed that this collection occurred on 24 out of 30 days, with a median frequency of three uploads per day.
- *Sensor Collector*: Performs a wide range of sensor readings (accelerometer, gyroscope, calibrated and uncalibrated magnetic field, barometer), Wi-Fi and cell scans, Wi-Fi RTT events, GNSS measurements and status, and current final geolocation fixes, then stores them in a local temporary cache (`NLP_S Cache`, see Section 5) before upload. Validation tests showed that this collection was more irregular, with activity detected on only 10 out of 30 days. Some days showed heavy collection and others very little or none, which makes its pattern difficult to interpret.
- *Passive Collector*: Passively collects Wi-Fi and cell scans, activity recognition results, and current final fixes requested by other components, then uploads data. Validation tests showed that this collection occurred on 23 out of 30 days, with a median frequency of four uploads per day.

[4] https://voilatile-pa.googleapis.com/google.internal.android.location.voilatile.v1.VoilaTileService/FindTiles. See Mombelli (2025) for more details on request and response structure.

[5] The term FREWLE is used in the context of GMS to denote the location estimation technology based on Wi-Fi AP fingerprinting.

[6] If the AP is RTT-enabled, this model contains information to allow positioning via Wi-Fi RTT technology, such as an anchor (AP absolute geolocation), floor, and vertical uncertainty.

[7] https://android-context-data.googleapis.com/google.internal.android.location.kollektomat.v1. KollektomatService/Offer.



(a) Cell-based position estimation process.

(b) Wi-Fi-based position estimation process.

Fig. 1. Sub-flows of network-based geolocation estimation.

See Mombelli (2025) for further details on `Kollektomat` requests, building upon the work of Leith (2025).

4.4. GNSS-based geolocation mechanisms

The second major component involved is the *GNSS Provider*. This provider supplies satellite-based location estimates. Our analysis identified two key stages:

1. **GNSS position estimation:** GNSS-based position estimation is handled by multiple internal providers. Location requests may be forwarded directly to GNSS hardware via the *Chipset GNSS Provider*, or processed by a higher-level *Blue Star Provider*, which combines GNSS measurements with inertial sensor data through the native library `libbluestar_jni_android.so`. In practice, the *Blue Star Provider* is preferred for high-frequency updates, while direct chipset access serves as a fallback or is used for lower-frequency requests. However, the *Blue Star Provider* was inactive during validation for unknown reasons, limiting verification of its precise behavior.
2. **Measurement corrections:** The initial estimate is refined by the *Bluesky Provider*, which applies environmental corrections using raster tiles retrieved from Google's `VoilaTile` API. These tiles encode per-pixel clutter characteristics (e.g., buildings, trees) and are cached locally in the `Bluesky Cache` (see Section 5). A Kalman filter[8] integrates these corrections with GNSS measurements, velocity, and bearing, and the refined fix is then reinjected into the active *GNSS Provider*.

[8] A Kalman filter is an algorithm that estimates the state of a dynamic system by combining predictions with noisy sensor measurements to produce more accurate results over time (Welch and Bishop, 2006).

4.5. Fusion of geolocation sources

After individual providers generate their estimates, the *Fusion Engine* integrates them into a final fix. This process involves three fusion controllers (Table 3) and five refining filters (Table 4). As detailed in Fig. 2, incoming estimates (internally called "evidence" in FLP) are processed in three stages:

1. **Update of fusion controllers and filters:** Each new piece of evidence updates the relevant fusion controllers, primarily according to its source provider. In parallel, the five refinement filters update their state according to their respective requirements.
2. **Determination of updated final geolocation fix:** The *Fusion Coordinator* then applies a hierarchy of decisions:
   (a) If available, prefer a recent injected GNSS fix.
   (b) Otherwise, if activation conditions are met (i.e., pedestrian movement, not in-vehicle/on-bycicle, sufficient amount of recent location evidence), use the *Particle Filter Fusion Controller*.
   (c) Otherwise, delegate to the *GNSS/Wi-Fi/Cell Fusion Controller*, which applies context-dependent logic (Table 5):
      i. If pedestrian activity is detected (via the *Pedestrian Switching Fusion* logic), a fusion of GNSS and Wi-Fi fixes is computed via the *GNSS–Wi-Fi Fusion* logic.
      ii. Otherwise, the 'best' estimate between a GNSS and a Wi-Fi fix is selected by applying the *Failover Fusion* logic.
      iii. The chosen fix is compared with a cell-based fix (via the *Failover Fusion* logic) to allow fallback when GNSS and Wi-Fi fixes are missing or poor.
3. **Refinement:** If it's not GNSS-injected, the chosen fix is refined by the five filters and returned as the output of the *Fused Location Provider*.

5. Intermediate traces of geolocation activity

During this extended geolocation process, GMS generates multiple intermediate traces that are stored in application-specific storage. A

Table 3
Overview of fusion controllers used to integrate geolocation evidence.

Controller	Description
Injection Sensor Fusion Controller	Passes through GNSS fixes corrected by measurement post-processing. Returns the most recent GNSS-injected location. This controller was never activated during validation testing.
Particle Filter Fusion Controller	Uses a particle-filter[a] algorithm to combine GNSS, inertial sensors (via BluePixel),[b] step detection, activity recognition, Wi-Fi RTT measurements, and network estimates into a refined fix.
GNSS/Wi-Fi/Cell Fusion Controller	Applies decision logic to select or combine GNSS, Wi-Fi, and cell estimates. Depending on accuracy and source availability, it may output: (1) a standalone cell, Wi-Fi, or GNSS fix; (2) a GNSS–Wi-Fi weighted average; or (3) a GNSS fallback after rejecting a poor Wi-Fi fix.

[a] A particle filter is a probabilistic algorithm that estimates the state of a dynamic system by representing possible states with a set of weighted samples (particles), often applied in tracking or localization under uncertainty (ScienceDirect, 2025).

[b] BluePixel, associated with the package `com.google.android.location.fused. bluepixel`, appears to be a nanoapp to process sensor data with minimal power usage.

Table 4
Overview of fusion filters used to refine geolocation fixes.

Filter	Role
Elevation Optimization Filter	Optimizes altitude estimation by integrating sensor inputs (barometric, GNSS, Wi-Fi, and weather) using graph-based optimization and historical filtering techniques.
Indoor Probability Filter	Stores the most recent indoor probability value derived from Wi-Fi-based estimates and applies it to the fused geolocation fix (if the probability estimate is not older than 60 s).
Speed Estimator Filter	Aggregates and smooths GNSS-based speed estimates to produce a refined and validated speed measurement for the current geolocation fix.
Dynamic Location Filter	Continuously refines the device's position using a Kalman filter that combines predicted motion from past state with new inputs (position, speed, bearing).
Accuracy Output Filter	Tracks historical accuracy values (in the last 5 s) and adjusts the final fix's reported accuracy accordingly.

Table 5
Fusion logics.

Logic	Role
Pedestrian Switching Fusion	Selects between a pedestrian-optimized and fallback location source based on recent motion activity and GNSS signal quality. Prioritizes the pedestrian source when conditions suggest on-foot movement.
GNSS–Wi-Fi Fusion	Combines GNSS and Wi-Fi location estimates using a weighted accuracy-based fusion. If the GNSS fix is significantly more accurate (accuracy at least twice as good), it is preferred. Otherwise, a fused fix is computed along the great-circle path between the two.
Failover Fusion	Selects between a primary and fallback location source based on availability, accuracy, recency, and spatial separation. Prefers the primary fix unless it is missing, outdated, or significantly less reliable than the failover source.

subset of these traces is detailed in Table 6 and described below (see Mombelli (2025) for detailed format specifications and parsing tools).

We also identified further traces, such as `/cache/dsm` (daily *NLP* usage statistics) and `/cache/nlp_devices` (Wi-Fi AP observations from *Burst* and *Passive Collector* events). These were only partially parsed and are not discussed here.

5.1. Details on intermediate traces

5.1.1. CFPC1 cache

This cache stores precomputed device geolocation fixes based on cell towers, returned by the `VoilaTile` API. Information include tower identifiers and associated geolocation and temporal data (i.e., timestamps of the associated `VoilaTile` response, in UTC). Each entry persists for approximately seven days, with cleanup performed daily. Validation tests on trace evolution corroborated this lifespan across all four tested devices (listed in Table 1). This cache reveals historical knowledge of nearby cell towers (both those serving the device and neighbors reported by Google remote services), but does not reliably identify which cell towers were actively serving the device.

5.1.2. FREWLE cache v2 (FTC)

This cache contains geolocation data for Wi-Fi APs returned by the `VoilaTile` API. Each response contributes the following to the cache: (1) general information about the `VoilaTile` response (e.g., FREWLE tile, timestamp in UTC, average elevation); (2) geolocation information for each received AP; and (3) BSSIDs of uninformative APs (i.e., detected by the device but unknown to the `VoilaTile` service). Entries persist for approximately seven days, with cleanup performed daily. This could not be corroborated through validation tests, as this cache was not present on any of the four tested devices. Again, this cache reveals historical information about nearby APs, but it does not reliably identify the APs that have been directly observed by the device.

5.1.3. VTC cache

Contains RTT models for Wi-Fi APs returned by the `VoilaTile` API. Information from each response is added to the cache: (1) BSSID aliases for each AP, if present (no aliases were observed in collected `VoilaTile` responses, leaving their nature unclear); (2) a timestamped[9] (in UTC) RTT model for each received RTT-enabled AP; and (3) BSSIDs of requested RTT-disabled APs. Entries remain available for approximately seven days, with daily cleanup routines. This could not be corroborated through validation tests, as this cache was not present on any of the four tested devices. Once again, while the cache preserves historical information about nearby APs, it does not allow a reliable distinction between APs directly observed by the device and additional ones included in the server response. This cache is no longer present in newer versions of GMS (from version 25.18.33 onwards).

5.1.4. FREWLE cache v3

This cache is a more advanced Wi-Fi AP cache that can replace `FREWLE Cache v2` and `VTC Cache`, implemented in LevelDB and composed of key-value pairs. A key includes a type identifier followed by a string, while the corresponding value is either a bytestring or a serialized Protobuf message. The known type identifiers and their associated values are summarized in Table 7.

Entries are retained until size thresholds are exceeded (approximately 10 MB, checked daily), at which point most of the oldest type-0 entries (along with all associated entries of other types) are deleted. Validation tests on trace evolution showed a lifespan of type-0 entries ranging from 7.0 to 10.9 days. Unlike v2, it distinguishes between APs directly observed (‘lookups’) and those only returned by the server (‘additional’).

5.1.5. NLP_S cache

A Protobuf-based cache which captures the output of *Sensor Collector* events and contains the sensor and contextual data listed in Section 4.3.3, along with additional metadata such as the Android build fingerprint, the timezone, and the GNSS chipset model. Each

[9] Due to an apparent implementation issue, the recorded timestamp granularity is approximately 86.4 ms rather than the expected 1-h resolution.

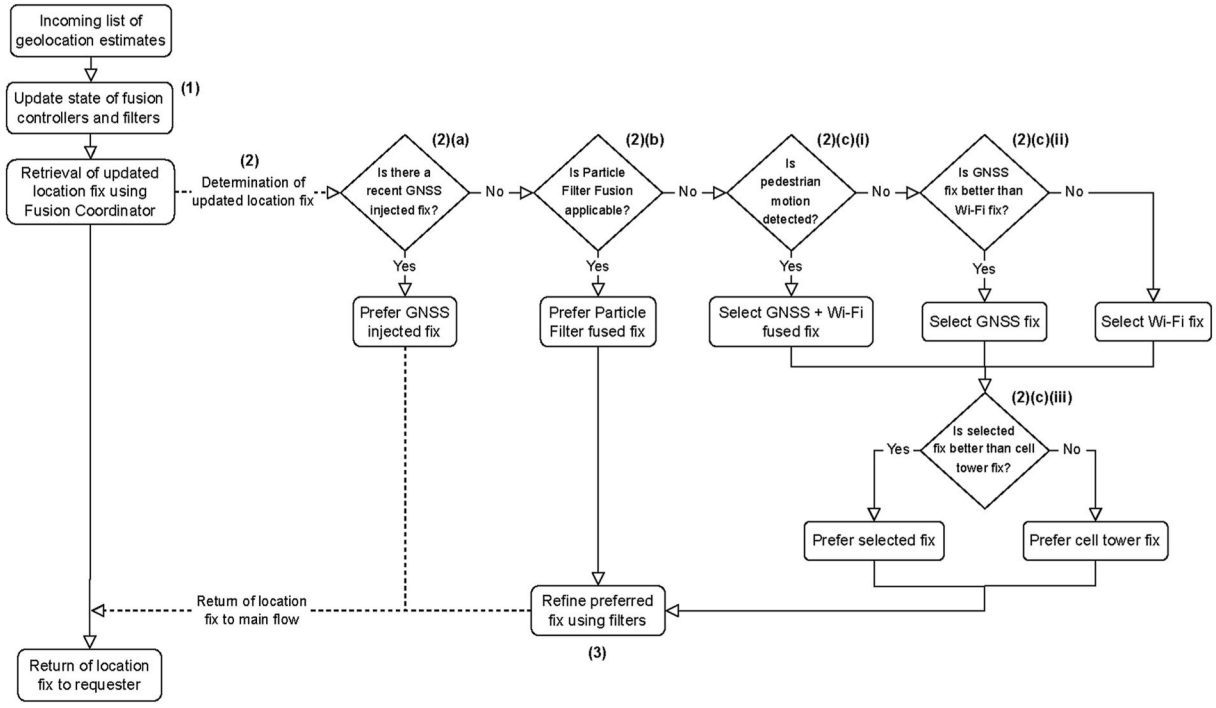


Fig. 2. Geolocation fusion logic.

Table 6
Overview of discovered intermediate traces generated during the geolocation process. Paths are relative to the private app data directory for the GMS package (`/data/data/com.google.android.gms/`).

Name	Path	File/Folder name	Format
CFPC1 Cache	cache/cfpc1/	cfpc1.*	Binary (AES-128-CBC encrypted)[a]
FREWLE Cache v2	cache/ftc/	ftc.*	Binary (AES-128-CBC encrypted)[a]
VTC Cache	cache/vtc/	vtc.*	Binary (AES-128-CBC encrypted)[a]
FREWLE Cache v3	app_frewle_tile_cache_manager_db/	frewle_tile_cache_manager_db/	LevelDB (cleartext)
NLP_S Cache	cache/nlp_s/	<*UTC_TIMESTAMP*>/0000*	Gzip compressed Protobuf (AES-128-CBC encrypted)[a]
Bluesky Cache	cache/bluesky/<*BLUESKY_VERSION*>/	<*TILE_HASH*>[b]	Protobuf (AES-128-CBC encrypted)[c]

[a] The decryption key is found in the NLP Cache Key file (`files/nlp_ck`).
[b] Static code analysis suggests that this value is the result of a MurmurHash3 x64 128-bit hash computed over a concatenated string consisting of the Bluesky version, a Bluesky tile ID, and the Android ID. Attempts to replicate this computation were unsuccessful.
[c] The decryption key is derived from the Android ID, found in the Checkin preference file (`shared_prefs/Checkin.xml`), using PBKDF2-SHA1 (1324 iterations).

Table 7
Identified key-value entries stored in `FREWLE Cache v3`.

Type ID	Key	Value
0	BSSID	Geolocation data for Wi-Fi APs (including floor ordinal ID, building ordinal ID and FREWLE tile ID), insertion timestamp and last lookup timestamp (both hour-level granularity, UTC).
1	Building ordinal ID	Building model (i.e., building coordinates encoded as a S2 Cell ID, building unique ID) and floor model for each floor (i.e., floor coordinates encoded as a S2 Cell ID, floor unique ID, floor label).
2	FREWLE tile ID	S2 Cell ID associated with the FREWLE tile and number of AP entries in the associated VoilaTile response.
3	BSSID	Aliases.
4	Nibble model-masked BSSID	BSSID.
5	Building S2 Cell ID and building unique ID	Building ordinal ID.
6	–	General cache metadata.

measurement is precisely timestamped (UTC). Each file represents a collection event and is deleted once transmitted to Google's servers. Validation tests revealed heterogeneous lifespans, ranging from 0.4 to 7.1 days depending on the device, with upper values consistently reaching approximately 7 days across all tested devices.

5.1.6. *Bluesky cache*

A Protobuf-based cache that stores raster tiles used by the *Bluesky Provider* for GNSS measurement correction. Each file in the cache corresponds to a single tile retrieved from the `VoilaTile` endpoint. No explicit timestamps are stored within these files. Instead, the cache mechanism uses the file's creation time (in UTC) in filesystem's metadata to reflect the original `VoilaTile` request timestamp. The modification and access times (in UTC) are updated only the first time a tile is looked up in a session, after which the tile is loaded into memory. The default retention appears to be of approximately 30 days (checked every 12 h), though this was not conclusively validated.

5.2. *Operational validation*

As described in Section 3.3, we conducted two validation tests to assess the operational prevalence of the identified intermediate traces.

Table 8
Operational prevalence of the discovered intermediate traces in real-world forensic acquisitions. For `FREWLE v3`, only type-0 entries are considered.

	Devices ($n = 307$)		Number of entries		Persistence (days)	
	#	%	Mdn	Max.	Mdn	Max.
CFPC1	172	56.0	2319	4096	7.0	18.6
FTC (informative)	45	14.7	298	10000	6.3	746.3
FTC (uninformative)	45	14.7	0	71	3.0	6.7
VTC (RTT-enabled)	140	45.6	1	16	8.1	132.0
VTC (RTT-disabled)	140	45.6	334	1000	–	–
FREWLE v3 (lookups)	173	56.4	261	9040	5.2	17.2
FREWLE v3 (additional)	173	56.4	5560	125574	5.1	17.0
NLP_S	45	14.7	Variable[a]		3.3	6.9
Bluesky	183	59.6	30	100	29.8	854.1

[a] The number of entries varies depending on the data type. Median number of entries values are: 1700 for acceleration, 3405 for gyroscope, 0 for calibrated magnetic field, 1537 for uncalibrated magnetic field, 6 for Wi-Fi scan, 0 for cell tower scan, 22 for GNSS status, 24.5 for GNSS measurements, 20 for final geolocation fixes.

The first evaluated the ability of four widely used and up-to-date forensic tools to detect and parse these traces. None of the tools were able to do so. The second test examined a set of real-world forensic acquisitions provided by the Vaud Cantonal Police (Switzerland). Out of 307 devices, 254 (82.7 %) contained at least one non-empty cache. The analyzed acquisitions spanned GMS versions from 21.09.15 (released around April 2021) to 25.34.34 (released around August 2025).[10] A detailed breakdown is provided in Table 8. The *persistence (at acquisition)* metric was assessed as the time between an entry's recorded timestamp and the moment of acquisition, and therefore reflects trace availability rather than full cache lifespan.

6. Discussion

This section discusses the key findings of the study in relation to the two research questions (RQs) presented in the Introduction. Our in-depth analysis focused on GMS versions 25.06.32 to 25.30.31, while validation experiments covered a broader range from 21.09.15 to 25.35.34. Importantly, GMS versions are not tied to Android OS releases, meaning that even older devices may run recent GMS builds and thus exhibit similar behaviors. While Android fragmentation and manufacturer customizations typically complicate the generalization of forensic findings (Kröger and Creutzburg, 2012), GMS seems to operate largely independently from vendor modifications. This suggests that our results are broadly applicable, though they must still be interpreted with caution.

6.1. RQ1: How are intermediate and final geolocation fixes computed on Android?

Our analysis shows that geolocation computation in the FLP is a modular, multi-phase process. Incoming requests first undergo permission checks and sanitization before being directed to the *Fusion Engine*, where data sources are selected primarily based on request priority and provider availability. These sources include a network-based provider that estimates position using Wi-Fi access points and cell towers as references, GNSS-based providers with optional environmental corrections, and sensor and contextual inputs that help determining the best geolocation strategy and refine the final fix. Final location fixes are produced through conditional fusion and post-processing filters, and may, under certain conditions, be served from a runtime cache rather than recomputed.

This adaptive architecture corroborates the implementation of multiple geolocation sources and techniques inferred in prior work, while also exposing sophisticated post-processing logics not previously documented in detail. Still, some mechanisms—notably Assisted GNSS (A-GNSS), cellular triangulation techniques, and Bluetooth-based positioning—were not clearly observed, but this absence could also be caused by limited testing conditions or encapsulation in other layers of the Android location stack.

From a forensic perspective, reported final fixes (which may later be transformed to application-level traces) should not be attributed to a single provider: they are the outcome of layered decisions and filters shaped by request parameters and context. Each component influences both the central estimate and the reported accuracy, while introducing its own limitations. For example, network-based positioning depends on crowdsourced reference data whose freshness and coverage may vary (e.g., stale, sparse, or imprecise entries). Post-processing such as accuracy coarsening can intentionally reduce spatial resolution or shift coordinates and timestamps. Stationary throttling may reuse earlier final fixes with updated metadata.

Overall, the FLP's fused architecture enhances resilience by cross-validating inputs, but this approach also introduces more potential failure points, particularly when data are missing, outdated, or biased. This is important to keep in mind in a forensic context, *a posteriori*, where metadata describing which sources were used and how they were weighted are not necessarily preserved, complicating the assessment of provenance and uncertainty of the resulting traces.

Understanding the underlying geolocation mechanisms enables practitioners to qualify the investigative or evidential weight of a location trace by reasoning about its possible sources, context of generation, and inherent limitations, rather than treating reported coordinates as ground truth. This mechanism-aware perspective helps to avoid overconfidence in apparent accuracy, supports the formulation of alternative hypotheses, and assists in identifying plausible explanations for anomalous or contradictory location data. For example, apparent movement before and after a critical event (e.g., a device briefly appearing to move away from and then return to a car crash location) may result from post-processing effects such as dynamic refinement or filtering rather than actual physical displacement (Meylan et al., 2025). Similarly, analyzing an application's location request parameters can help infer which providers were potentially activated (e.g., NLP, GNSS, Bluesky), allowing investigators to contextualize a trace in terms of expected accuracy and uncertainty even when such metadata is absent. More broadly, knowledge of the geolocation pipeline supports better-structured forensic research and clearer communication of uncertainty in investigative and judicial contexts.

6.2. RQ2: What local traces are generated during geolocation?

We identified multiple caches that record intermediate stages of the geolocation process (see Section 5). This shows that nearly every stage of the Android geolocation pipeline generates intermediate traces; each one reflects a distinct subprocess and is mostly independent from the others. As a result, partial information can often be recovered even in the absence of a complete final geolocation fix.

From an evaluative perspective, however, the probative value of these traces must be approached with caution. For instance, the `Bluesky Cache` records only the creation and lookup times of raster tiles, with repeated lookups overwriting earlier values. The `CFPC1 Cache` stores pre-computed cell-based fixes, while Wi-Fi caches retain AP estimations, both derived from opaque server-side source data. Because many caches contain data retrieved directly from Google's servers, their presence does not necessarily imply direct observation by the device, and their real-world accuracy cannot be independently verified.

Some caches are volatile, subject to dynamic deletion or overwriting

[10] Release dates are based on the APKMirror upload timeline (APKMirror, 2025).

in response to new location requests or changes in connectivity. This highlights the importance of timing and contextual awareness when seizing and acquiring data from devices. Nevertheless, since geolocation services are invoked routinely by both system processes and user applications, these traces are likely to be present in practice. These are hidden from ordinary users, and when correlated they provide valuable insights on the device's position at a certain moment in time.

From an investigative perspective, intermediate traces should not be treated as substitutes for final location fixes, but as complementary data that can contextualize, corroborate, or challenge location hypotheses. In practice, these traces can support forensic reasoning in several ways. First, they can help corroborate or falsify hypotheses derived from application-level location traces or other contextual information. For example, Wi-Fi AP lookups or sensor activity may support or contradict a hypothesized presence inferred from messaging or navigation data. Second, when final location fixes are absent or sparse, intermediate traces can help formulate hypotheses about presence or movement, provided their limitations are carefully considered. For example, when reconstructing a car's route, the `CFPC1 Cache` may provide coarse indications of presence and movement based on the timing and sequence of cell-based entries. Third, some traces provide value beyond geolocation *per se*, such as temporal markers, cellular of Wi-Fi connectivity information, sensor activity, or traces of environmental interaction (e.g., detected APs), which may be relevant to broader event reconstruction or hypothesis testing. Additional illustrative use cases and examples are provided in Mombelli (2025).

An analysis of 307 real-world forensic acquisitions showed the operational prevalence of these caches (see Table 8): 82.7 % of devices contained at least one non-empty intermediate trace, across a wide range of Android versions (7–15), vendors (Samsung, Xiaomi, Google, among others), and GMS releases spanning 2021 to 2025. The large number of entries observed, together with their persistence at acquisition,[11] corroborates the forensic prevalence of these data. While specific implementations may evolve, the consistent presence of these traces across devices and GMS versions indicates that they may originate from stable architectural properties of FLP's geolocation pipeline rather than transient implementation details.

At the same time, none of the tested leading forensic tools was able to recognize or parsed these traces. This gap highlights how substantial sources of location traces may remain invisible in current forensic practice.

7. Limitations

Our findings are subject to multiple limitations. First, we examined only two request types (*Current Location* and *Last Location*), leaving others (e.g., location updates, geofencing) unexplored. Second, the analysis focused solely on GMS, excluding alternative frameworks such as the Android Framework APIs, which remain relevant in restricted regions, GMS-free distributions, or older Android versions. Similarly, less common technologies like BLE sightings were only briefly considered. Third, results were based on a limited set of GMS builds; while GMS appears consistent across manufacturers, fragmentation and regional variants may still introduce differences. Finally, this study does not include ground-truth validation of intermediate traces or final geolocation fixes. The opacity of Google's server-side logic leaves elements like Wi-Fi- and cell-based estimates as black boxes, and assessing their real-world accuracy would require a different experimental methodology, which is beyond the scope of this work. As a consequence, a formal forensic interpretation framework for these traces is also currently lacking, as such interpretation must build on statistically grounded knowledge of their real-world accuracy. Our focus is therefore on reconstructing trace generation mechanisms and provenance, which is a necessary prerequisite for accuracy-focused studies, an aspect we are currently investigating.

[11] This metric should be interpreted with caution, as it reflects the time between an entry's creation and the forensic acquisition of the device. Acquisition practices and clock errors can inflate values beyond realistic lifespans; median values are nonetheless informative.

8. Conclusion

This study examined the internal mechanisms and forensic relevance of geolocation trace generation on Android devices, focusing on Google Mobile Services (GMS). While geolocation traces are among the most exploited forms of digital evidence, the processes by which they are produced and stored remain largely opaque, particularly within proprietary systems. Such opacity creates risks for forensic interpretation, from overestimating accuracy to misattributing device presence.

Through reverse engineering, behavioral testing, and trace extraction, we reconstructed the logic of the Fused Location Provider (FLP) across multiple GMS versions. We showed that geolocation computation is a multi-stage process combining GNSS, Wi-Fi, and cellular inputs with sensor and contextual data, filtered and fused by adaptive algorithms. A key contribution is the identification of undocumented intermediate traces—such as encrypted caches for Wi-Fi and cell positioning, sensor data collections, and GNSS correction tiles—which are widely present and persist for extended periods but remain undetected by existing forensic tools. Practically, this work enables more robust forensic interpretation of Android geolocation traces by exposing how final location fixes are generated and by providing additional system-level traces that can contextualize, corroborate, or challenge application-level location data. Building on this foundation, future research should pursue deeper analysis of GNSS subsystems and the NLP stack; extend validation across a wider range of devices, environments, and GMS versions; and develop forensic tools capable of parsing intermediate caches (e.g., ALEAPP or Hansken plugins). In addition, case-based or investigative studies would be valuable to assess how intermediate traces can be interpreted and contextualized in real-world forensic scenarios. Investigating the evolution of GMS and assessing the integrity of Google's crowdsourced databases would further broaden both the forensic and societal relevance of this line of research, and this is work we are currently pursuing.

In a digital age where location is both highly informative and highly contested, forensic practitioners must be equipped with both the tools and the understanding to interpret geolocation traces accurately. Our study aims to support that mission by revealing the mechanisms beneath the surface, questioning black-box assumptions, and laying the groundwork for scientifically sound digital trace analysis.

CRediT authorship contribution statement

Samuele Mombelli: Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Writing - Original Draft, Visualization. **Thomas R. Souvignet:** Conceptualization, Methodology, Resources, Writing - Review & Editing, Supervision.

Declaration of generative AI and AI-assisted technologies usage

The authors used ChatGPT (OpenAI) and DeepL Write to assist in improving the language quality and readability of this manuscript. All suggestions generated by these tools were carefully reviewed, edited, and validated by the authors, who retain full responsibility and accountability for the content of this work.

Declaration of 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

The authors would like to thank their colleagues Cléo Berger, Céline Vanini, Bénoît Meylan, and Antoine Desbiolles for their support, discussions, feedback, and reviews throughout the development of this work. We are also grateful to the Vaud Cantonal Police, and in particular Jérôme Arn, for their availability and support in providing validation data. Finally, we thank the anonymous reviewers for their constructive feedback, which helped greatly improve the quality of this paper.

Data availability

We do not redistribute decompiled source code from Google Mobile Services. The identifier mappings generated during decompilation—linking obfuscated names to researcher-assigned descriptive labels—are considered derivatives of proprietary work and are therefore available only upon request under a research data-sharing agreement.

References

Amundsen, A.E., Ovens, K.M., 2017. Forensics analysis of Wi-Fi communication traces in mobile devices. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 3632–3637. https://doi.org/10.1109/BigData.2017.8258357.

AOSP, 2025a. Android.location | API reference. URL: https://developer.android.com/reference/android/location/package-summary. (Accessed 21 May 2025).

AOSP, 2025b. android/platform-samples. URL: https://github.com/android/platform-samples. (Accessed 25 April 2025).

AOSP, 2025c. LocationRequest | API reference. URL: https://developer.android.com/reference/android/location/LocationRequest. (Accessed 18 September 2025).

APKMirror, 2025. Google play services APK. URL: https://www.apkmirror.com/apk/google-inc/google-play-services/. (Accessed 14 December 2025).

Bays, J., Karabiyik, U., 2019. Forensic analysis of third party location applications in android and iOS. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1–6. https://doi.org/10.1109/INFOCOMWKSHPS47286.2019.9093781.

Berger, C., Meylan, B., Souvignet, T.R., 2024. Uncertainty and error in location traces. Forensic Sci. Int.: Digit. Invest. 51, 301841. https://doi.org/10.1016/j.fsidi.2024.301841.

Brignoni, A., 2025. abrignoni/ALEAPP. https://github.com/abrignoni/ALEAPP. (Accessed 25 May 2025) [Version 3.4.0].

Casey, E., Jaquet-Chiffelle, D.-O., Spichiger, H., Ryser, E., Souvignet, T., 2020. Structuring the evaluation of location-related Mobile device evidence. Forensic Sci. Int.: Digit. Invest. 32, 300928. https://doi.org/10.1016/j.fsidi.2020.300928.

Cortesi, A., Hils, M., Kriechbaumer, T., et al., 2010. Mitmproxy: a free and open source interactive HTTPS proxy. https://mitmproxy.org/. (Accessed 22 September 2025) [Version 12.1.2].

Dabove, P., Di Pietra, V., 2019. Towards high accuracy GNSS real-time positioning with smartphones. Adv. Space Res. 63, 94–102. https://doi.org/10.1016/j.asr.2018.08.025.

Froklage, P., 2024. Not all geolocation data is created equal. URL: https://www.magnetforensics.com/blog/not-all-geolocation-data-is-created-equal/. (Accessed 24 September 2025).

Google, 2013. Android at google I/O 2013: keynote Wrapup. URL: https://android-developers.googleblog.com/2013/05/android-at-google-io-2013-keynote-wrapup.html. (Accessed 21 May 2025).

Google, 2023. Migrate to google play services location and context APIs. URL: https://developer.android.com/develop/sensors-and-location/location/migration. (Accessed 27 April 2025).

Google, 2024a. FusedLocationProviderClient. URL: https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient. (Accessed 27 April 2025).

Google, 2024b. Location Request | google play services. URL: https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest. (Accessed 18 September 2025).

Google, 2025a. Android – certified play protect partners. URL: https://www.android.com/certified/partners/. (Accessed 23 September 2025).

Google, 2025b. Fused location provider API. URL: https://developers.google.com/location-context/fused-location-provider. (Accessed 4 June 2025).

Google, 2025c. Google play supported devices. URL: https://storage.googleapis.com/play_public/supported_devices.html. (Accessed 21 May 2025).

Google, 2025d. Google/dfindexeddb. https://github.com/google/dfindexeddb. (Accessed 25 May 2025) [Version 20241105].

Google, 2025e. Location and context APIs. URL: https://developers.google.com/location-context. (Accessed 21 May 2025).

Google, 2025f. Protocolbuffers/protobuf. URL: https://github.com/protocolbuffers/protobuf. (Accessed 2 May 2025) [Version 3.21.12].

Google, 2025g. S2 geometry. URL: http://s2geometry.io/. (Accessed 4 June 2025).

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S.K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D., Zhou, M., 2021. GraphCodeBERT: pre-training code representations with data flow. http://arxiv.org/abs/2009.08366 arXiv:2009.08366 [cs].

Konloch, 2025. Konloch/bytecode-viewer. https://github.com/Konloch/bytecode-viewer. (Accessed 27 April 2025) [Version 2.13.0].

Kröger, K., Creutzburg, R., 2012. Forensics of location data collected by Google android mobile devices. In: Multimedia on Mobile Devices 2012; and Multimedia Content Access: Algorithms and Systems VI, 8304. SPIE, pp. 201–210. https://doi.org/10.1117/12.909891.

Leith, D., 2025. doug-leith/android-protobuf-decoding. URL: https://github.com/doug-leith/android-protobuf-decoding. (Accessed 2 June 2025).

Macarulla Rodriguez, A., Tiberius, C., van Bree, R., Geradts, Z., 2018. Google timeline accuracy assessment and error prediction. Forensic Sci. Res. 3, 240–255. https://doi.org/10.1080/20961790.2018.1509187.

Maghdid, H.S., Lami, I.A., Ghafoor, K.Z., Lloret, J., 2016. Seamless outdoors-indoors localization solutions on smartphones: implementation and challenges. ACM Comput. Surv. 48 (53), 1–53:34. https://doi.org/10.1145/2871166.

Maus, S., Hoefken, H., Schuba, M., 2011. Forensic analysis of geodata in android smartphones. In: International Conference on Cybercrime, Security and Digital Forensics. URL: https://www.researchgate.net/profile/Hans-Hoefken/publication/260320851_Forensic_Analysis_of_Geodata_in_Android_Smartphones/links/0046353148e8fccc09000000/Forensic-Analysis-of-Geodata-in-Android-Smartphones.pdf.

Merry, K., Bettinger, P., 2019. Smartphone GPS accuracy study in an urban environment. PLoS One 14, e0219890. https://doi.org/10.1371/journal.pone.0219890. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0219890. Publisher: Public Library of Science.

Meylan, B., Desbiolles, A., Labidi, I., Souvignet, T.R., 2025. When Location Traces (Almost) Tell the Story: Reconstructing Road Accidents from Position-based Smartphone Traces. Manuscript under review.

Moayeri, N., Li, C., Shi, L., 2019. Indoor localization accuracy of major smartphone location apps. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–8. https://doi.org/10.1109/WCNC.2019.8885639 iSSN: 1558-2612.

Mombelli, S.. FLP Geolocation Analysis. doi:10.5281/zenodo.17920907. Reserved Zenodo DOI; final public release available upon publication. Actively developed repository available at. https://github.com/mombesam-unil/FLP-geolocation-analysis.

Obeidat, H., Shuaieb, W., Obeidat, O., Abd-Alhameed, R., 2021. A review of indoor localization techniques and wireless technologies. Wirel. Pers. Commun. 119, 289–327. https://doi.org/10.1007/s11277-021-08209-5.

OpenAI, 2025. ChatGPT. https://chatgpt.com. (Accessed 8 October 2025) [Models o3, 4o, 4o-mini-high, 5].

Python Software Foundation, 2025. Download python. https://www.python.org/downloads/. (Accessed 25 May 2025) [Version 3.12.3].

Ryser, E., Spichiger, H., Jaquet-Chiffelle, D.-O., 2024. Geotagging accuracy in smartphone photography. Forensic Sci. Int.: Digit. Invest. 50, 301813. https://doi.org/10.1016/j.fsidi.2024.301813. URL: https://www.sciencedirect.com/science/article/pii/S2666281724001379.

Salamh, F.E., Mirza, M.M., Hutchinson, S., Yoon, Y.H., Karabiyik, U., 2021. What's on the horizon? An In-Depth forensic analysis of android and iOS applications. IEEE Access 9, 99421–99454. https://doi.org/10.1109/ACCESS.2021.3095562. Conference Name: IEEE Access.

ScienceDirect, 2025. Particle filter - an overview | ScienceDirect topics. URL: https://www.sciencedirect.com/topics/computer-science/particle-filter. (Accessed 11 September 2025).

skylot, 2025. Skylot/jadx. https://github.com/skylot/jadx. (Accessed 25 April 2025) [Version 1.5.1].

Sleuth Kit Labs, 2025. Autopsy - Download. https://www.autopsy.com/download/. (Accessed 25 May 2025) [Version 4.21.0].

Spichiger, H., 2022. The Evaluation of Mobile Device Evidence Under Person-Level, Location-Focused Propositions. University of Lausanne Lausanne. Doctorate thesis. https://serval.unil.ch/resource/serval:BIB_5949ECA240E7.P002/REF.

Vadla Ravnås, O.A., 2025. Frida/frida. https://github.com/frida/frida. (Accessed 28 April 2025) [Versions 16.6.6 to 17.3.0].

Welch, G., Bishop, G., 2006. An introduction to the kalman filter. URL: https://www.cs.unc.edu/welch/media/pdf/kalman_intro.pdf.

WerWolv, 2025. WerWolv/ImHex. https://github.com/WerWolv/ImHex. (Accessed 25 May 2025) [Version 1.37.4].