Resilience of Forensic Evidence Acquisition Under Database Schema Drift

Afiqah M. Azahari[a], Andrea Oliveri[a], Davide Balzarotti[a]

[a]*Security Department, EURECOM 450 Route des Chappes, Biot, France*

Abstract

The reliability of mobile forensic analysis depends not only on the ability to extract application databases but also on the stability of the structures that organize user data. As Android applications evolve, their databases undergo continual schema modifications, which alter established acquisition workflows.

In this paper, we present the first longitudinal study of schema drift in Android mobile applications, examining 320 versions of 20 popular Android apps released between 2022 and 2025. By systematically extracting and analyzing their databases, we reveal how structural changes, ranging from incremental column additions to the removal of entire tables, shape the evidential landscape. We further assess the resilience of SQL-based forensic queries across versions, showing how even minor schema drift can invalidate extractions or may miss newly introduced artifacts. Our results demonstrate that communication and social media apps exhibit the most volatile schema evolution, while navigation, browser, and note-taking apps remain comparatively stable. These findings reveal a critical yet overlooked threat to evidential completeness, motivating the development of adaptive, drift-aware forensic tools that can anticipate and accommodate ongoing application evolution.

Keywords: application databases, schema drift, digital forensics, Android, APK versions

1. Introduction

Mobile applications have become a necessity platform for personal productivity, communication, and social networking. However, most of the actions performed on the platform, such as sending a message, calling, posting updates, and navigating a route, generates data that is persistently stored in the applications' databases. On Android devices, thanks to the operating system native support, these databases are often implemented using SQLite and provide one of the richest sources of digital evidence available during forensic investigation. Their value is immense: Conversations can be constructed from message histories, movements can be mapped from location records, and online behaviour can be reconstructed from the browser artifact.

The access to these precious pieces of evidence depends on the correct acquisition and interpretation of data records stored in the different databases that each application create and manage according to its own unique schema. However, updated versions of the applications are released at short intervals, sometimes weekly or even daily. Along with new features and bug fixes, these updates might also introduce changes in the database schema. Such changes, referred in this study as **schema drift**, can involve the addition of a new table or field, its removal, or the restructuring of some existing table's columns. To extract information from SQL databases, existing forensic tools use relational queries that are tailored for a specific database schema. As schemas drift, forensic acquisition queries may break, require updates or return incomplete results, ultimately increasing the risk of failing to extract critical forensic evidence.

This problem is not hypothetical. For example, the extraction queries described by Anglano et al. Anglano et al. (2017) for Telegram have become nowadays (2025) obsolete, as the proposed queries to extract blocked users and messages no longer work due to table drops and table renaming. Furthermore, open-source forensic tools absadiki (2025); sepinf-inc (2025) regularly require manual human revisions, updates, and testing of the implemented queries to remain compatible with schema drift. Despite its importance, schema drift in Android applications and its implications for forensic acquisition remain largely unexplored by researchers. Existing studies had focused on extraction methods, case-specific database analysis, or other aspects of an application's evolution (such as permissions and vulnerabilities), but none have tried to assess how often database changes in new versions of applications, nor the impact of schema drifts for the completeness and reliability of forensic evidence extractions.

In this paper, for the first time, we address this gap by presenting a longitudinal study on 20 widely used Android applications across the total of 320 versions released between 2022 and 2025. By extracting and comparing SQL databases created and updated by the different application across versions, we analyze how schemas drift over time, classify the resulting patterns, and assess their impact on forensic acquisition queries. Our findings reveal that some applications have frequent schema drift, particularly communication and social platforms, and therefore present recurring challenges for forensic tools. Other application instead remain largely stable even over long period of time. Through case studies, we demonstrate how acquisition queries require updates, or fail as schemas evolve, exposing

significant risks to the collection of application artifacts. In addition, our study shows that schema drift does not only cause queries to fail outright. Although existing queries may remain functional, they can omit newly introduced forensic data in the schema by ignoring added columns or tables, or by no longer linking critical relationships. Conversely, new schemas may introduce additional fields, which remain under-explored unless a human analyst adapt the existing tools to take advantage from these changes. These results suggest the need for drift-aware forensic tools that can adapt in an automatic way to evolving database schemas, ensuring uninterrupted workflows and the comprehensive extraction of digital evidence.

2. Motivation

The forensic value of application databases is not only determined by the richness of the data they store, but also by the stability of the structures that govern how such data can be retrieved. While we already highlighted the growing dependence of investigations on these databases, here we focus on the specific drivers that make schema drift.

A first driver is the release cadence of modern mobile applications. Many popular apps do not only perform cosmetic improvements: in new versions they frequently introduce new features that can alter the underlying storage structures. A second driver lies in the mismatch between how forensic tools evolve and how applications evolve. Commercial and open-source tools are maintained by human developers who must monitor app releases, inspect database layouts, and rewrite SQL queries when incompatibilities arise. As shown in the case of open-source projects, like the Whatsapp Msgstore Viewer absadiki (2025) and IPED sepinf-inc (2025), the frequency of application updates often overtakes the capacity of maintainers, resulting in tools that quickly become outdated. This creates a dangerous situation in which investigators may unknowingly rely on queries that execute successfully but no longer capture all relevant artifacts. Finally, the reliability of digital evidence is a legal as well as a technical concern. As schemas drift, queries that are not updated may fail to retrieve newly added data fields created by application developers in the application database. Investigators and forensic tool developers must remain aware on the frequent updates across hundreds of applications, a task that is challenging to sustain in practice. Such gaps are often difficult to detect and can undermine both the completeness and the reliability of acquisition queries. Schema drift therefore represents not just a technical inconvenience, but a direct threat to the repeatability and defensibility of forensic practice.

These reasons motivate the central research questions of this work:

- **Q1: How often are popular Android applications updated, and how are releases distributed across major, minor, and patch versions?** By studying update intervals and the distribution of release types, we measure how actively applications releases their updates. Frequent update suggest that core features of application might be redefined or redesigned, and this may suggest that the database where the evidence reside might be modified as well.
- **Q2: To what extent do these updates alter the underlying database schemas?** In particular, how often are tables renamed or dropped, columns added or restructured, and schema layouts changed across versions? Such changes can break existing acquisition queries, necessitate revisions, or lead to incomplete data extractions within forensic workflows. This is particularly problematic when investigators unknowingly rely on outdated SQL queries, potentially leading to missed evidence or partial reconstructions.
- **Q3: How do schema changes impact the robustness of forensic acquisition queries?** When database schemas are modified, the change can affect the ability to extract evidence from the database. Particularly, if queries must be updated to recover forensic casework information after a version update, does schema evolution preserve or challenge the reliability and completeness of evidence collection?

By addressing these questions, this paper provides a longitudinal perspective on how Android application evolution shapes the durability of forensic workflows. Understanding schema drift at scale is a necessary step toward the development of adaptive, drift-aware tools capable of ensuring evidential integrity despite the rapid pace of mobile application change.

3. Methodology

Our approach of understanding the stability and reliability of SQL-based forensic extraction across different versions of popular applications starts with the process of selecting 20 widely used applications that are widely supported by commercial forensic extraction tools such as Cellebrite Cellebrite (2018, 2020), MSAB MSAB (2014, 2022), and Belkasoft Belkasoft (2018, 2025) and reflect realistic forensic challenges across multiple domains.

For each application in our dataset (Step 1) we performed a preliminary analysis to identify interesting databases, and then (Step 2) we executed them in a controlled environment and interacted with them to populate their databases with realistic data. We proceeded to systematically extract the SQL databases produced by each application version (Step 3), compared their schemas to identify changes (Step 4), and finally we assessed how schema evolution affected the recovery of forensic artifacts (Step 5).

3.1. Environment Setup

In Android systems, each installed application is assigned a private directory (only accessible with root privileges) that it can use to store persistent internal data, including databases, configuration files, shared preferences, and cached content. To access this information, we conducted our experiments in a rooted Android 16.0 (API level 34) `x86_64` emulator. Some

Table 1: Collected Databases and Required Interactions per Application Version

Application	DB Creation/Fill Trigger Actions	Extracted DBs	DB Missing Reason
Communication Applications			
WhatsApp	Load messages	5/16	Forced update
Kakao Talk	Load messages	15/16	Forced update
Telegram	Load messages	16/16	
Viber	Login + startup	16/16	
LINE	Load messages	16/16	
Google Chat	Send a message	16/16	
Discord	Login + message	14/16	DBs not created
GroupMe	Login + message	16/16	
Truecaller	Login + startup	16/16	
Browser Applications			
Firefox	Browse	16/16	
DuckDuckGo	Browse	15/16	Forced update
Brave	Browse	16/16	
Tor Browser	Browse	16/16	
Opera	Browse	16/16	
Social Media Applications			
Twitter/X	Post + message	16/16	
Instagram	Login + message	15/16	Forced update
TikTok	Login + message	16/16	
Note-taking Applications			
Evernote	Login, create notes, organize event	7/16	Forced update
Location/Navigation Applications			
Google Maps	Query + navigate	16/16	
Waze	Query + navigate	16/16	

versions of certain applications, however, were incompatible with this Android version or enforced compatibility checks that prevented them from running on this setup. In these cases, we switched to older emulator images, specifically Android 11.0 (API level 30) or Android 8.1 (API level 27), allowing us to circumvent the restrictions and execute the applications and collect their databases.

3.2. Applications Download (Step 1)

To study the evolution of database schemas across different applications, our approach requires collecting multiple versions of each application. For each application, we download its APK from APKMirror APKMirror (2025), a publicly available repository that archives application packages across different release versions. We retrieved all monthly releases between May 2024 and May 2025. In addition, to capture longer-term changes, we also included one version from six months prior to May 2024 and one version per year for the two preceding years.

This allowed us to track schema drift both in the short term (month-to-month changes) and in the long term (yearly evolution). In months where multiple updates were released, we selected the earliest available version for that month to maintain temporal consistency across the dataset. We managed to collect up to 16 APK versions per application, with a total of 320 APKs for the entire dataset. This structured versioning strategy enables a uniform basis for longitudinal schema comparison and helps ensure that any observed changes in the database structures resulted from the version evolution and not because of inconsistencies in sampling.

Table 1 summarizes the applications we studied, along with their categories and description. For a visual overview of the timeline, versions and application update categories included in this study, check the additional material Azahari et al. (2025).

3.3. Application Installation and Interaction (Step 2)

Each application version was installed on the emulator according to its update timeline, and a predefined sequence of manual interactions was carried out to both trigger database creation and populate it with forensically relevant data. Repeating the same interaction sequence across all versions ensured that the resulting datasets were directly comparable. This step was particularly important for applications like WhatsApp and Telegram, which only instantiate their databases after user activity (such as sending messages or initiating conversations) has occurred. The interaction steps executed prior to data extraction are detailed in Table 1.

3.4. Database Extraction (Step 3)

Next, we copied each application's private data directory from the emulator to the analysis machine to extract the SQL databases generated by each app. However, not all databases containing forensic artifacts were recoverable. In fact, in some rare cases the application refused to start because it required a mandatory update. Despite numerous attempts with compatible emulator images, date/time changes, offline mode, and dynamic instrumentation (Frida) to trigger the required interactions, these versions refused to create the local database until a more recent version was installed.

For our study, we concentrated on databases that contains forensically relevant evidences, such as message histories, posting activity, location queries, and event records. These databases were then examined to detect instances of schema drift. Table 1 summarizes the number of databases successfully collected for each application and highlights the challenges that limited recovery.

3.5. Schema Drift Analysis (Step 4)

To examine how database schemas changed across versions, we relied on the `sqldiff` utility provided with SQLite. This tool outputs DB schema differences as SQL statements (e.g., `CREATE TABLE`, `DROP TABLE`, `ALTER TABLE`), enabling a global structural comparison between databases. To capture finer-grained changes, we complemented this with metadata inspection using `PRAGMA table_info`, which allowed us to analyze column definitions within tables. By combining these approaches, we classified schema drift into table creation, deletion, restructuring, and column-level modifications – such as column removal and addition. Tracking these changes provided a longitudinal view of how evidence storage evolved. Newly introduced tables could expand the scope of forensic artifacts, whereas dropped or restructured tables could result in data loss, relocation, or reinterpretation. This analysis formed the basis for evaluating the stability of database structures and their implications for forensic extraction.

3.6. Query Robustness Evaluation (Step 5)

To evaluate the resilience of SQL queries on the application's databases, we examined how schema drift influenced their effectiveness in extracting evidence across different application versions. Our objective was to estimate how often forensic tools must be updated to remain accurate. For this purpose, we designed a set of SQL queries for each application, targeting the database schema of versions released in May 2024 – a point chosen as the approximate midpoint of our dataset. Query design was guided by the forensic features listed in Table 3.6, which capture evidence types of practical relevance to investigators, such as messaging records, posting activity, event logs, and location history. Once developed on the reference schema, these queries were executed across all earlier and later versions. This approach enabled us to assess both backward compatibility with older releases and forward robustness against newer updates. By recording query failures and their causes, we quantified the impact of schema drift on the reliability of forensic data extraction.

Table 2: Forensic evidence extracted per application

Application	Forensic Evidence
Communication Applications	
WhatsApp	DM; GM; AM; Call Logs; Group Call Logs; Deleted Messages
Kakao Talk	DM; GM; AM; Deleted Messages
Telegram	DM; GM; AM
Viber	DM; GM; Call Logs
LINE	DM; Group Info; AM; Call Logs
Google Chat	DM; GM; Search History; Deleted Messages
Discord	DM
GroupMe	GM; AM; Event Conversations; Deleted Messages
Truecaller	Call Logs; Message Content; AM; Favorite Contacts
Browser Applications	
Firefox	Search Queries; Bookmarks; Recently Closed Tabs
DuckDuckGo	DFM; Bookmarks
Brave	Visited Pages; DFM; Recently Closed Tabs
Tor Browser	Open Tabs; Bookmarks; Recently Closed Tabs
Opera	Visited Pages; DFM; Location Browsing; Search titles and URLs
Social Media Applications	
Twitter/X	DM; Bookmarked Posts
Instagram	DM; Followers List; Recent Searches
TikTok	DM; User Interactions; Deleted Messages
Note-taking Applications	
Evernote	Deleted Notes; All Notes; Reminder List; Calendar Events; Tasks
Location/Navigation Applications	
Google Maps	Current Location History
Waze	List of Favorite Locations; Search Queries; Navigation History

[DM] Direct Messages
[GM] Group Messages
[AM] Attachment Metadata
[DF] Downloaded Files Metadata

4. Results

4.1. Update Frequency and Release Patterns

Table 3 highlights the substantial variation we observed in the update frequency across application categories. Communication and social platforms, including TikTok, WhatsApp, Telegram, and Instagram, update most frequently, with median intervals ranging from one to eight days. These applications also release both minor and major updates at a rapid pace, reflecting a continuous user-driven improvements. This observation is consistent with the high schema drift we will discussed in Section 4.2, as shorter release cycles with significant version changes increase the likelihood of modifications in database evidence storage.

Table 3: Application Update Intervals and Types, Grouped by Category

Application	Update Interval (days)	Major	Minor	Patch
Communication Applications				
WhatsApp	2	4	11	–
Kakao Talk	12	4	8	3
Telegram	8	3	11	1
Viber	8	7	8	–
LINE	8	4	11	–
Google Chat	3	13	2	–
Discord	3	15	–	–
GroupMe	20	2	13	–
Truecaller	8	2	13	–
Browser Applications				
Firefox	10	15	–	–
DuckDuckGo	7	–	15	–
Brave	3	–	13	2
Tor Browser	22	4	11	–
Opera	8	10	5	–
Social Media Applications				
Twitter/X	3	2	13	–
Instagram	5	15	–	–
TikTok	1	8	7	–
Note-taking Applications				
Evernote	5	–	13	2
Location/Navigation Applications				
Google Maps	3	2	13	–
Waze	24	1	13	1

In contrast, browser and navigation applications exhibit more moderate update cycles, typically ranging from 7 to 24 days. Browsers such as Firefox, DuckDuckGo, Tor Browser, and Opera primarily release updates as scheduled minor or major versions rather than frequent patches. Their development strategy emphasizes bundled releases, in which security fixes, bug resolutions, and feature enhancements are integrated into larger, less frequent updates.

Finally, note-taking applications, such as Evernote, update approximately every five days. However, these updates are limited to patches and minor revisions, with no major redesigns observed.

4.2. Schema Drift

Our longitudinal analysis of 16 versions for each application in our dataset reveals that database schemas evolved over time with a applications-depending rate. In Table 4 we summarizes the databases we examined to capture this evolution, according to the criteria presented in Section 3.6.

As shown in the table, several applications exhibited substantial schema drift, including table restructuring through the addition or removal of columns, as well as the creation and deletion of entire tables. In contrast, some databases maintained stable structures, showing no observable changes across versions. We can categorize applications into three groups based on the extent of schema drift.

High-drift applications – Applications such as WhatsApp, KakaoTalk, Truecaller, Viber, LINE, Google Chat, TikTok, and GroupMe exhibited substantial schema changes across application updates. Most of these belong to the communication category, where databases evolve in tandem with the introduction of new features. Messaging applications, in particular, frequently expand functionality to support richer interactions, adding capabilities such as group messaging, community announcements, one-time-view attachments, and deleted messages. Also notable from the query coverage analysis (see Table 5, Telegram row of 'Deleted Message' Query Type and Notes) is that, for now, only Telegram handles deleted messages by physically dropping the corresponding database rows. In contrast, other messaging applications that allow message deletion, such as WhatsApp, Instagram, Google Chat, TikTok, and GroupMe remove only the cell data containing the message text while retaining related metadata such as the timestamp, sender, and receiver. This approach in handling acquisition of deleted evidence may change in the other messaging applications if they want to adopt Telegram's privacy-oriented feature. When that happens, proving that a message existed must rely on version-aware techniques rather than current direct content reads.

As shown in Table 3, new features are often introduced within short release intervals, requiring developers to add new tables or columns to store the associated data. As a result, forensic acquisition queries designed for previous versions may miss newly created data evidence from the database. If the acquisition queries are not updated, reflecting the schema drift, investigator may risk overlooking this available data evidence.

Moderate-drift applications – These includes applications such as Telegram, X (Twitter), Waze, one database of DuckDuckGo, Brave, Instagram, and two of the databases in Opera. While these applications did not experience extensive changes, they still showed a noticeable structural evolution.

Within this category, communication and social platforms displayed more frequent schema changes than non-communication applications. In contrast, applications not primarily focused on communication or social interaction, that probably does not strictly depends on new user eye-capturing features, such as Waze, DuckDuckGo, Brave, and Opera, demonstrated only minor schema drift, typically occurring once a year or approximately every six months.

Low-drift applications – No schema drift was observed in the database of Evernote, Google Maps, and Discord, as well as in one DuckDuckGo database, a Tor Browser database, a LINE database, two Instagram databases, and an Opera database. We observed that the core database structures that store essential artifacts, such as tab history, downloaded data, or search queries in browsers, and searching location, have remained constant and have not required structural modification, even as these applications continue releasing updates. As a result, we observed no drift in this category of applications.

Drift Direction and Nature of Changes – With respect to table and column additions or removals, our analysis shows that schema drift across all studied applications was predominantly additive, with far fewer instances of table or column deletion. This trend likely reflects a common development practice: preserving backward compatibility for existing user data while extending database functionality.

When removals did occur, they were limited to a small subset of applications. Table drops were observed in Telegram, Waze, Viber, LINE, TikTok, Opera, DuckDuckGo, and Brave, ranging from a single table removal to a maximum of three in a single update. However, three-table drops were exceptionally rare, occurring only twice across all studied versions, one instance even within a minor update. Two-table drops occurred instead three times, all associated with major releases.

For example, in TikTok a major update removed the tables `IM_RES_CACHE` and `USER_EXTRA`. The first table dropped stored media cache information, while the second one contained metadata about user-interaction features, such as logging interaction prompts between profiles. These tables not only reflects a change in feature design but also dropped the availability of data related forensic artifacts, including cached media traces and records of user-to-user interaction events.

Cases of columns dropped were observed in X (Twitter), Truecaller, Viber, Brave, GroupMe, Opera, and WhatsApp, with instances ranging from a single dropped column to a maximum of two. For example, X dropped the column `users.has_nft_avatar`; Truecaller dropped `msg_im_quick_actions.action_value`; Viber dropped `conversations.to_number` as well as `hidden_gems.countries` and `hidden_gems.monetized_phrase`; GroupMe dropped `messages.has_image_url` and `chats.is_muted`; and WhatsApp dropped `optimised_delivery_info.msg_timestamp`.

We observes that these table and column drops did not occur in monthly incremental updates but were observed primarily in yearly or six-month snapshots. This pattern suggests that destructive schema changes are more likely to occur during major feature overhauls or redesigns rather than routine updates. The removal of tables or columns containing forensic evidence has direct implications for forensic practice, as it necessitates adapting tools and queries to ensure consistent data collection across different application versions.

More details on the specific versions and databases affected by tables and columns removals are provided in Tables on the additional material repository Azahari et al. (2025).

5. Stability of Forensic Extraction Under Drift

After we categorized the types of schema drift observed across application versions, we now evaluate the stability of forensic data extraction in the presence of such drift. Our objective is to determine how robust forensic queries remain when applications introduce schema changes and whether extraction tools can continue to function without modification.

To this end, we constructed a sequence of SQL queries targeting the forensic features defined in Table 3.6 within each application's database. For example, in Telegram we extracted

Table 4: Schema drift across applications and time ranges (Nov 21 – May 25)

Application	Database	11/21–11/22	11/22–11/23	11/23–05/24	05/24–06/24	06/24–07/24	07/24–08/24	08/24–09/24	09/24–10/24	10/24–11/24	11/24–12/24	12/24–01/25	01/25–02/25	02/25–03/25	03/25–04/25	04/25–05/25
WhatsApp	msgstore.db												4/0/3 (6/0)	3/0/8 (26/0)	6/0/10 (34/0)	4/0/6 (11/1)
KakaoTalk	KakaoTalk.db								0/0/1 (8/0)						0/0/1 (2/0)	
	KakaoTalk2.db						1/0/2 (3/0)		1/0/0 (0/0)			1/0/0 (0/0)		1/0/0 (0/0)	0/0/1 (2/0)	
Telegram	cache4.db	14/0/4 (16/0)	12/1/6 (14/0)	7/1/4 (10/0)	2/0/0 (0/0)				2/0/0 (0/0)	0/0/1 (2/0)	0/0/2 (6/0)					
Viber	viber_messages	3/0/7 (14/0)	5/0/11 (12/0)	2/1/3 (7/1)	0/0/2 (4/0)		1/0/1 (2/0)	0/0/3 (8/0)		0/0/2 (4/0)		0/0/1 (2/0)	0/0/1 (4/0)		3/0/0 (0/0)	1/0/1 (0/2)
LINE	naver_line	0/0/2 (13/14)	0/2/2 (6/0)	0/0/1 (2/0)	2/1/2 (4/0)							0/0/2 (4/0)				0/0/1 (4/0)
Google Chat	dynamite.db	0/0/7 (32/0)	0/0/5 (26/0)	0/0/4 (18/0)		0/0/2 (4/0)	0/0/1 (4/0)						0/0/1 (2/0)	0/0/1 (4/0)	0/0/2 (16/0)	0/0/1 (2/0)
GroupMe	groupme.db	2/0/4 (12/1)	1/0/4 (12/0)	1/0/2 (4/0)		0/0/1 (4/0)				1/0/2 (0/2)		0/0/2 (14/0)	0/0/1 (0/0)			1/0/2 (3/0)
Truecaller	tc.db	3/0/3 (12/0)	2/0/11 (29/1)	0/0/3 (4/0)			0/0/1 (1/0)			0/0/1 (8/0)	0/1/0 (0/0)	0/5/2 (3/0)		0/0/2 (3/0)		0/0/1 (2/0)
DuckDuckGo	app.db	2/0/0 (0/0)	2/3/1 (2/0)	2/0/0 (0/0)						1/0/0 (0/0)	0/0/1 (2/0)			1/0/0 (0/0)	0/1/0 (0/0)	1/0/0 (0/0)
Brave	History	2/0/6 (48/1)	2/2/5 (14/0)	0/0/1 (2/0)												
Tor Browser	places.sqlite		0/0/3 (6/0)													
Opera	History	4/0/4 (30/1)	2/1/5 (14/0)	1/1/1 (4/0)												
	newsfeed.db	0/3/1 (2/0)				0/0/1 (2/0)				0/0/1 (4/0)				0/0/1 (2/0)		
Twitter/X	0-66.db			0/0/4 (9/1)						0/0/1 (1/0)					0/0/1 (4/0)	0/0/1 (0/1)
Instagram	direct.db			1/0/0 (0/0)												0/0/1 (3/0)
TikTok	{account_id}_im.db	0/0/4 (9/0)		0/0/3 (6/0)			3/0/2 (6/0)			0/0/1 (2/0)			0/0/1 (1/0)	0/0/1 (2/0)	0/0/1 (1/0)	
	db_im_xx	3/2/1 (10/0)	0/0/1 (3/0)									0/1/0 (0/0)				
Waze	user.db		0/0/1 (2/0)	0/1/0 (0/0)	0/0/1 (2/0)											

Values indicate database schema drift over time: new / dropped / restructured (columns added / columns removed).
We have studied also the following DBs that however do not show any modification along the timeline:
LINE: `call_history`, Discord: `{account_id}/a`, Firefox: `places.sqlite`, DuckDuckGo: `downloads.db`, Tor Browser: `tab_collections`, Opera: `reading.db`, Instagram: `ranked_user_{account_id}` and `recent_searches.db`, Evernote: `UDB-{account_id}+RemoteGraph.sql`, GoogleMaps `gmm_sync.db`.

Table 5: Query Coverage – Communication Applications

Application	Query Type	Ver. Matched	Notes
Telegram	Direct Messages	15/16	1 update (schema change)
	Group Messages	13/16	2 updates (schema change)
	Deleted Message	–	No row available
LINE	Direct Messages	4/16	1 update (schema change)
	Attachment Data	4/16	1 update (schema change)
Discord	Direct Message	14/16	No DB in later versions
GroupMe	Events/Conversations	13/16	1 update (schema change)
Truecaller	Call Logs	15/16	1 update (schema change)
	Message Content	14/16	1 update (schema change)
	Attachment Data	14/16	1 update (schema change)

Table 6: Query Coverage – Social Media Applications

Application	Query Type	Ver. Matched	Notes
Twitter/X	Direct Messages (DM)	14/16	2 updates (database name change)
	Bookmarked Posts	14/16	2 updates (database name change)
Instagram	Deleted Message	–	No row available

Table 7: Query Adaptations Across Applications and Timeline Versions

Application	Query Type	11/21	11/22	11/23	05/24	06/24	07/24	08/24	09/24	10/24	11/24	12/24	01/25	02/25	03/25	04/25	05/25
Telegram	Direct Messages	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Group Messages	★	•	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Twitter/X	Direct Message	★	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Bookmarked Post	★	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Truecaller	Call Logs	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Message Content	•	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Attachment Data	•	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LINE	Direct Messages	✓	✓	✓	✓	•	•	•	•	•	•	•	•	•	•	•	•
	Attachment Data	✓	✓	✓	✓	•	•	•	•	•	•	•	•	•	•	•	•
Discord	Direct Message	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GroupMe	Events Conversation	•	•	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓ = query (built at 05/24) works unchanged on this version; • = query does not work and requires adaptation; ★ = query was updated at this version to handle a schema change (new baseline); ✗ = not applicable / no query possible.

direct message information, including sender and receiver identifiers, timestamps, and, where available, geolocation data. All queries were initially developed for the May 2024 version, which served as our baseline.

The baseline queries were subsequently applied across all other collected versions to evaluate extraction stability under both forward drift (monthly updates from May 2024 to May 2025) and backward drift (yearly and six-month snapshots preceding May 2024, specifically November 2021, November 2022, and November 2023). This methodology allowed us to assess not only whether queries continued to execute successfully, but also how the scope and content of available evidence changed over time.

For backward drift, any query failures were addressed by rewriting the queries to restore functionality, with the results then compared to the May 2024 baseline. This comparison revealed whether earlier versions provided equivalent evidence, reduced evidence, or, in some cases, additional evidence.

For forward drift, we similarly evaluated whether the baseline queries remained valid or required modification, and whether the extracted evidence differed from that of May 2024. Through this process, we were able to determine whether schema evolution over time led to an expansion, reduction, or consistency in forensic evidence.

The results of our experiments show that messaging applications frequently experienced query failures due to schema changes, as summarized in Table 5. For instance, Telegram, LINE, GroupMe, and Truecaller all required at least one query update across the 16 versions.

In contrast, non-communication applications, including browsers, note-taking tools, and navigation apps, showed no query failures. This finding aligns with their classification in the moderate- and low-drift categories: applications for which the database schemas changed more often over time were also more likely to require adjustments in the forensic extraction queries.

Social media applications exhibited mixed levels of extraction stability. For example, Instagram and TikTok required no modifications across all collected versions, despite frequent minor schema changes. This indicates that the database structures storing forensic-relevant data remained unchanged.

Conversely, Twitter/X proved more fragile: queries targeting direct messages and bookmarks failed in 2/16 versions due to database name changes, as shown in Table 6. Although these changes were relatively minor, they were sufficient to disrupt otherwise valid queries, necessitating updates to maintain forensic support.

Table 7 summarizes for all the applications and versions when forensic queries required modifications to remain functional. A clear pattern emerges: most query failures occurred during backward drift, particularly in the yearly and six-month snapshots preceding May 2024.

Forward drift testing, covering monthly updates from May 2024 to May 2025, showed far fewer failures, with the majority of queries remaining operational across consecutive versions. An exception was observed in LINE, where queries targeting direct messages and attachment data failed from June 2024 onward. This failure resulted from a table drop in that update, which removed the `contacts` table containing essential mes-

```sql
SELECT c.last_message,
       c.last_created_time,
       c.owner_mid,
       con.name
FROM chat c
LEFT JOIN chat_member cm
    ON c.owner_mid = cm.mid
LEFT JOIN contacts con
    ON cm.mid = con.m_id;
```

(a) Baseline query valid for 11/2021 to 05/2024, used to retrieve the last message, creation time, owner, and contact name.

```sql
SELECT c.last_message,
       c.last_created_time,
       c.owner_mid,
       ch.from_mid,
       cm.mid,
       cm.created_time
FROM chat_history ch
LEFT JOIN chat c
    ON c.chat_id = ch.chat_id
LEFT JOIN chat_member cm;
```

(b) Updated query valid from 06/2024 to 05/2025: the `contacts` table was dropped and replaced by `chat_history`, allowing extraction of message and participant data but no longer providing contact names.

Figure 1: LINE direct message acquisition query adaptation due to schema drift.

sage and attachment information.

Overall, these findings emphasize the importance of designing forensic support tools that balance two strategies: maintaining resilience against routine incremental updates while remaining flexible enough to handle major schema changes that may occur unexpectedly during both backward and forward evolution.

5.1. Case Study: Query Update and Fallback

Now we examine how query updates and fallback strategies influence the richness and completeness of evidential data. Schema changes do not always cause queries to completely fail, but they can return results which does not contain newly added fields that can contain relevant evidence. This highlight that forensic tools must do more than simply maintain functional queries, they must also monitor schema modifications that could introduce new sources of evidence or, if overlooked, result in data loss.

By analyzing examples of query adaptation, we illustrate both the risks of reduced evidential value and the opportunities to capture additional artifacts as databases evolve across version updates.

5.1.1. Query Update

Consider the messaging application LINE, the baseline SQL query developed for the May 2024 version, shown in query 1a, allows to extract direct messages by joining the `chat`, `chat_member`, and `contacts` tables. However, following the June 2024 app update (version 14.4.5), this query start to fail because the `contacts` table, which previously stored user information, was removed from the schema and its usage of on more recent application version (forward query adaptation) result impossible.

A revised query is then required (Figure 1b). Although the updated query still retrieves key elements such as the last message text, message owner, and identifiers, the removal of the `contacts` table prevents the extraction of contact names. This reduces the amount of directly linkable identity information available in the results. For forensic acquisition, this illustrates that schema changes may not entirely remove evidence, but can limit the range of data points that can be retrieved for analysis.

5.1.2. Query Fallback

Now we illustrate how applications databases evolved to include richer data over time introducing more detailed structures and then increasing the extractable evidence. As shown in the queries adaptation in query 2, schema drift in Telegram required progressive fallbacks to support group message extraction in older versions. The baseline query for May 2024-2025 (refer to Figure 2a) relied on `messages_topics` and `topics`, which provided both message content and contextual fields such as group labels and edit timestamps. However, in the database schemas from November 2022 and November 2023, the baseline query is no longer functional, as it fails to return the same results as the acquisition query on the baseline version. This is because the `edit_date` field did not yet exist in those versions of the application. In this version of the application's database, a forensic investigator must adjust the baseline query to extract only the message data, as the date of group creation or modified is missing (Refer to Figure 2b). We executed the queries designed for the November 2022 and November 2023 versions on the Telegram database from November 2021. The query failed on this version with an error of missing `messages_topic` and `topics`. Upon examining the database schema, we observed that the table uses for storing group message data was not yet present in the November 2021 version.

On top of that, if a forensic investigator or developers is unaware of schema drift as the application evolves, newly introduced columns that may contain important evidence may be overlooked. Take acquisition query presented in 2b as an example. If the same query is applied to extract group message data in 05/2025, it will still execute successfully but does not extract another possible evidence that is important for forensic that is timestamp. This case shown that if the forensic investigator or tool developer is unaware of new columns introduced through schema drift during an application update, the outdated acquisition may miss newly available data.

In this case study, the schema drift shows the need for practitioners to monitor newly introduced features, as in this case where schema drift in the form of additional columns provides more data for forensic acquisition.

6. Related Work

Mobile application forensics has been widely studied from different angles, with researchers examining how evidence can

```sql
SELECT
  m.date,
  t.data AS topic_data,
  m.data AS message_data,
  t.edit_date,
  m.uid,
  u.name,
  u2.phone
FROM messages_topics m
LEFT JOIN topics t
  ON t.topic_id = m.
      topic_id
LEFT JOIN users u
  ON u.uid = m.uid
LEFT JOIN user_phones_v7
    u2
  ON u.uid = u2.key;
```

(a) Baseline query valid from 05/2024 to 05/2025, extracting message content, topic data, edit date, user ID, name, and phone number.

```sql
SELECT
  m.date,
  t.data AS topic_name,
  m.data AS message_data,
  m.uid,
  u.name,
  u2.phone
FROM messages_topics m
LEFT JOIN topics t
  ON t.topic_id = m.
      topic_id
LEFT JOIN users u
  ON u.uid = m.uid
LEFT JOIN user_phones_v7
    u2
  ON u.uid = u2.key;
```

(b) Fallback query valid from 11/2022 to 11/2023, extracting required data but losing `edit_date`, which records the group creation date.

```sql
SELECT
  m.date,
  t.data AS message_data,
  m.uid,
  u.name,
  u2.phone
FROM messages_v2 m
LEFT JOIN channel_users_v2
    t
  ON t.uid = m.uid
LEFT JOIN users u
  ON u.uid = m.uid
LEFT JOIN user_phones_v7
    u2
  ON u.uid = u2.key;
```

(c) Fallback query valid for 11/2021. Extracting data from messages_v2 + channel_users_v2 data format, with no group name data entirely.

Figure 2: Telegram group messages query adaptations due to schema drift

be extracted, how applications evolve over time, and how these changes affect the reliability of forensic tools.

Forensic Extraction Tools and Techniques – Numerous research works have focused on the development of different techniques to extract forensic evidence from mobile applications databases and their recoverability in function of changes in databases' layout. An initial work in this direction was performed in 2015 by Sgaras et al. Sgaras et al. (2015) that has shown how to practically acquire forensic evidence from popular communication apps like WhatsApp and Viber, but without studying structural changes over time. At the same time, several open-source tools has been proposed to extract evidences from applications such as ALEAPP Brignoni (2023) works with plugins to parse evidence from Android file-system dumps or archives and generates reports on user and application activity, Chatistics SkSumit (2023) that parses exported chat logs, and Whapa B16f00t (2023) that decrypts and parsed evidence from WhatsApp databases.

A first work that tries to address the problem of apps' database schema evolution has been proposed by Shimmi et al. Shimmi et al. (2020). In their study, they analyzed schema evolution in iOS backups, proposing automated comparison of table and column structures to keep updated with where the evidence can be extracted. While they showed that schema drift disrupts query reliability, their scope was limited to native iOS applications. Another interesting methodology was developed by Lee et al. Lee et al. (2023) that presented a predictive approach to identify schema layouts in messaging apps, without however, study how that layout changes in different application versions. Recently, AnForA Anglano et al. (2020), Argus Boztas et al. (2025) and Puma Institute (2025) tools have been proposed to automate detection of app-generated artifacts through snapshotting and scripted app interactions. Two of these tools, internally use `sqldiff` to highlight database changes but did not evaluate how schema changes impacts forensic evidence extraction. In contrast, our study evaluates widely used third-party Android apps and focuses on how schema changes directly affect the completeness of forensic evidence.

Application Evolution – Beyond forensics, researchers have examined how Android applications evolve from different point of view. Prior works investigated code complexity and maintainability Gao et al. (2019a), accessibility features Dos Santos et al. (2023); Oliveira et al. (2024), and topic modeling of release changes Thomas et al. (2014). Hecht et al. Hecht et al. (2015) highlighted how poor design choices in updates can degrade performance and quality. Security- and privacy-focused works have shown that permission requests often evolve dangerously Calciati and Gorla (2017); Taylor and Martinovic (2017), while large-scale analyses of millions of app records have revealed persistent issues such as third-party tracking and malicious behaviors Wang et al. (2019). Vulnerability research further showed that insecure dependencies can persist across multiple updates Gao et al. (2019b).

While these studies collectively highlights that mobile applications evolve rapidly, however, they mostly focus on code, permissions, or functionality. To the best of our knowledge, our study is the first empirical work to investigate application version evolution through the lens of database schema drift and its direct consequences on forensic extraction and acquisition. By combining longitudinal schema analysis with query testing, we demonstrate not only how schemas change as the versions evolve but also how this changes alters the reliability and completeness of digital evidence available to investigators.

7. Discussion, Limitations and Way Forward

Our study provides one of the first longitudinal examinations of schema drift in application databases across multiple Android apps and versions. The results reveal a clear stratification between categories of applications. Communication and social networking platforms demonstrate high schema volatility, consistent with their fast-paced release cycles and constant

feature additions. In contrast, navigation, note-taking, and certain browser applications remain relatively stable, making them less susceptible to drift-induced forensic losses.

A central implication present in this paper is that schema drift not only disrupts existing queries but also demands continual revision so that newly introduced data evidence in the schema does not remain unqueried. This is particularly problematic in forensic contexts, where investigators may unknowingly rely on outdated SQL queries that continue to return partial data without signaling new possible forensic evidence introduced because we also show that schema drift can also expand the evidential landscape by introducing new data sources. Drift-aware acquisition frameworks, therefore, need to move beyond mere query maintenance and incorporate adaptive mechanisms for detecting newly added structures and surfacing them to the investigator.

From a broader perspective, these findings highlight that the current forensic ecosystem remains reactive. Open-source and commercial tools alike depend on constant human intervention to rewrite extraction logic. Our results suggest that future directions should emphasize automation: schema-diff based monitoring, query template generation, and machine learning models capable of predicting evidential table mappings. Such approaches could substantially reduce the latency between schema change and forensic tool update, preserving evidential reliability in real-world investigations.

While comprehensive, our study is subject to some limitations. The dataset of 20 applications, though diverse and representative of forensic interest, cannot fully capture the heterogeneity of the Android ecosystem. Other application categories may display different drift behaviors. Some applications enforced mandatory updates or exhibited anti-emulation features, limiting our ability to consistently initialize and extract their databases. This restriction may have reduced the number of SQL queries broken due to schema drift and led to missing or incomplete coverage of certain versions. Finally, also if out of scope of the paper, our analysis focused primarily on structural schema drift (tables, columns, relationships) rather than semantic drift. Even when schemas remain stable, the meaning of fields, encodings, or application-level semantics may change in ways that alter evidential interpretation. Detecting such semantic drift requires deeper, content-aware analysis beyond the scope of this study.

Future work should extend this analysis in different directions such as broadening the corpus of applications to include additional application categories and the development of prototype drift-aware forensic tools, capable of automated schema monitoring and adaptive query regeneration, that will be key in moving the field from reactive maintenance toward proactive, resilient evidence extraction.

8. Conclusion

We demonstrate that application database schemas are not static but evolve with different release of the application. Analyzing 320 versions of 20 Android applications, we showed that schema drift directly affects the stability and reliability of forensic acquisition queries. The study further reveals that communication and social platforms perform the highest levels of drift, forcing updates to forensic aquisation queries, whereas navigation, note-taking, and some browser applications remain largely stable with no significant schema changes. As schemas drift, queries developed for a specific database version require continual updates, as current acquisitions may fail due to dropped columns or tables, and also to avoid missing newly introduced evidential data. This requires investigators to remain aware of any schema changes during version updates to preserve evidential completeness and acquisition reliability. Finally, our study highlights the need for forensic tools to incorporate adaptive query generation and schema-agnostic capabilities to mitigate the impact of schema drift on evidence extraction.

References

absadiki, 2025. whatsapp-msgstore-viewer: Free, open-source and cross-platform tool to decrypt, read, and view whatsapp `msgstore.db` database. `https://github.com/absadiki/whatsapp-msgstore-viewer`. Accessed: 15 July 2025.

Anglano, C., Canonico, M., Guazzone, M., 2017. Forensic analysis of telegram messenger on android smartphones. Digital Investigation 23, 31–49. doi:10.1016/j.diin.2017.09.002.

Anglano, C., Canonico, M., Guazzone, M., 2020. The android forensics automator (anfora): A tool for the automated forensic analysis of android applications. Computers & Security 88, 101650. URL: `https://doi.org/10.1016/j.cose.2019.101650`, doi:`10.1016/j.cose.2019.101650`.

APKMirror, 2025. Apkmirror âĂŠ free and safe android apk downloads. `https://www.apkmirror.com/`. Accessed: 6 August 2025.

Azahari, A., Oliveri, A., Balzarotti, D., 2025. Dataset and additional tables. URL: `https://github.com/eurecom-s3/resilience-dataset/`. accessed: 2025-05-16.

B16f00t, 2023. Whapa: Whatsapp parser tool. `https://github.com/B16f00t/whapa`. Accessed: 2025-05-16.

Belkasoft, 2018. WhatâĂŹs new in bec v.9.0. `https://belkasoft.com/whats_new_in_version_9_0`. Accessed: 6 August 2025.

Belkasoft, 2025. WhatâĂŹs new in belkasoft x v.2.8. `https://belkasoft.com/new`. Accessed: 6 August 2025.

Boztas, A., De Jong, J., Hadjigeorghiou, C., 2025. Argus: A new approach for forensic analysis of apps on mobile devices. Forensic Science International: Digital Investigation 53, 301938. URL: `https://doi.org/10.1016/j.fsidi.2025.301938`, doi:`10.1016/j.fsidi.2025.301938`.

Brignoni, A., 2023. ALEAPP: Android logs events and protobuf parser. `https://github.com/abrignoni/ALEAPP`. Accessed: 2025-05-16.

Calciati, P., Gorla, A., 2017. How do apps evolve in their permission requests? a preliminary study, in: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), IEEE. pp. 37–41.

Cellebrite, 2018. Ufed ultimate & ufed infield and ufed physical analyzer, ufed logical analyzer & cellebrite reader v7.5 and analytics desktop v7.0 release notes. May 2018.

Cellebrite, 2020. Ufed, ufed infield, ufed physical analyzer, ufed logical analyzer and cellebrite reader v7.32 release notes. April 2020.

Dos Santos, P.S.H., Oliveira, A.D.A., De Jesus, T.B.N., Aljedaani, W., Eler, M.M., 2023. Evolution may come with a price: analyzing user reviews to understand the impact of updates on mobile apps accessibility, in: Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems, pp. 1–11.

Gao, J., Li, L., Bissyandé, T.F., Klein, J., 2019a. On the evolution of mobile app complexity, in: 2019 24th international conference on engineering of complex computer systems (ICECCS), IEEE. pp. 200–209.

Gao, J., Li, L., Kong, P., Bissyandé, T.F., Klein, J., 2019b. Understanding the evolution of android app vulnerabilities. IEEE Transactions on Reliability 70, 212–230.

Hecht, G., Benomar, O., Rouvoy, R., Moha, N., Duchien, L., 2015. Tracking the software quality of android applications along their evolution (t),

in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 236–247.

Institute, N.F., 2025. Puma - programmable utility for mobile automation. https://github.com/NetherlandsForensicInstitute/puma. Accessed: 6 August 2025.

Lee, W.C., Chen, H.Y., Lin, T.N., 2023. A predictive classification model for identifying conversation-related mobile forensic data in instant messaging applications, in: 2023 International Symposium on Digital Forensics and Security (ISDFS), IEEE. pp. 1–8. doi:10.1109/ISDFS58141.2023.10131853.

MSAB, 2014. Xry offers support for new encrypted version of whatsapp. https://www.msab.com/updates/xry-offers-support-for-new-encrypted-version-of-whatsapp. Accessed: 6 August 2025.

MSAB, 2022. Recover even more digital evidence from apps with xry 10.0.1. https://www.msab.com/updates/released-today-recover-even-more-digital-evidence-from-apps-with-xry-10-0-1. Accessed: 6 August 2025.

Oliveira, A.D.A., dos Santos, P.S.H., Aljedaani, W., Eler, M.M., 2024. Exploring the influence of software evolution on mobile app accessibility: Insights from user reviews. Journal of the Brazilian Computer Society 30, 584–607.

sepinf-inc, 2025. IPED: Digital forensic tool to process and analyze digital evidence. https://github.com/sepinf-inc/IPED. Accessed: 15 July 2025.

Sgaras, C., Kechadi, M.T., Le-Khac, N.A., 2015. Forensics acquisition and analysis of instant messaging and voip applications, in: Proceedings of the 13th ADFSL Conference on Digital Forensics, Security and Law (CDFSL), ADFSL. URL: https://www.cdfsl.org.

Shimmi, S.S., Dorai, G., Karabiyik, U., Aggarwal, S., 2020. Analysis of ios sqlite schema evolution for updating forensic data extraction tools, in: 2020 IEEE International Conference on Intelligence and Security Informatics (ISI), IEEE. pp. 1–8. URL: https://doi.org/10.1109/ISI49825.2020.9280529, doi:10.1109/ISI49825.2020.9280529.

SkSumit, 2023. Chatistics: Parse and visualize chat logs from messenger, hangouts, whatsapp, and telegram. https://github.com/SkSumit/Chatistics. Accessed: 2025-05-16.

Taylor, V.F., Martinovic, I., 2017. To update or not to update: Insights from a two-year study of android app evolution, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA. p. 45âĂŞ57. URL: https://doi.org/10.1145/3052973.3052990, doi:10.1145/3052973.3052990.

Thomas, S.W., Adams, B., Hassan, A.E., Blostein, D., 2014. Studying software evolution using topic models. Science of Computer Programming 80, 457–479. URL: https://www.sciencedirect.com/science/article/pii/S0167642312001621, doi:https://doi.org/10.1016/j.scico.2012.08.003.

Wang, H., Li, H., Guo, Y., 2019. Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play, in: The World Wide Web Conference, Association for Computing Machinery, New York, NY, USA. p. 1988âĂŞ1999. URL: https://doi.org/10.1145/3308558.3313611, doi:10.1145/3308558.3313611.