Automatically generating digital forensic reference data triggered by mobile application updates

By:
Angelina A. Claij-Swart, Erik Oudsen, Bouke Timbermont, Christopher Hargreaves, Lena L. Voigt

From the proceedings of
The Digital Forensic Research Conference
DFRWS APAC 2025
Nov 10-12, 2025

Forensic Science International: Digital Investigation 54 (2025) 301985

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS APAC 2025 - Selected Papers from the 5th Annual Digital Forensics Research Conference APAC

Automatically generating digital forensic reference data triggered by mobile application updates

Angelina A. Claij-Swart [a,*], Erik Oudsen [a,*], Bouke Timbermont [a,*], Christopher Hargreaves [b], Lena L. Voigt [c]

[a] Netherlands Forensic Institute, The Hague, the Netherlands
[b] Department of Computer Science, University of Oxford, Oxford, United Kingdom
[c] Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany

ARTICLE INFO

Keywords:
Digital forensics
Datasets
Reference data
Data synthesis
Tool validation
Tool testing
Mobile forensics

ABSTRACT

Mobile applications are subject to frequent updates, which poses a challenge for validating digital forensic tools. This paper presents an approach to automate the generation of reference data on an ongoing basis, and how this can be integrated into the overall validation process of a digital forensic analysis platform. Specifically, it describes the architecture of the mobile data synthesis framework Puma, shares its capabilities via an open-source project, and shows how it can be used in a tool testing workflow triggered by application updates. The value of this approach is demonstrated with three example use cases, documenting the use of the approach over six months and reporting insights and experiences gained from this integration. Finally, this work highlights additional contributions the proposed approach and tooling could make to the digital forensics community.

1. Introduction

The field of digital forensics is still facing many of the problems identified by Garfinkel (2010), such as size, complexity, and number of devices. Automation can help in handling these challenges. However, as highlighted in Casey (2002), "when evaluating evidence, its reliability and accuracy are of grave importance both in the investigative and probative stages of a case," and more recently, "the courts have the expectation that the methods to produce the data that an expert bases their opinion on are valid" (UK Forensic Science Regulator, 2020). As automation increases, so does the opportunity to increase uncertainty. Early work by Carrier (2003a) introduces how error can result from translating data through abstraction layers[1] used to facilitate presentation of data in a form that allows analysis. More recent work presents a more complex and specific set of automated processes within modern monolithic forensic tools and documents errors that occur at each stage (Hargreaves et al., 2024b).

Recently, the SOLVE-IT knowledge base (Hargreaves et al., 2025) began to index weaknesses and mitigations in specific digital forensic techniques, and one of the highlighted mitigations, which is used for many of the weaknesses, is tool testing. Tool testing in digital forensics has been discussed since 2000 (Guttman et al., 2011) and a more extensive discussion of existing work can be found in Section 2. However, while there is some existing work, often discussing a need for testing, publication of tangible advances in digital forensic tool testing is less common. This is despite the significant challenges of tool testing in an area where developed tools must process complex, diverse, and even potentially adversarial data.

The dynamic nature of smartphone applications introduces further challenges. According to Statista, as of April 2025, 36 percent of the top 1000 Android applications in Google's PlayStore were updated at least weekly, with 73 percent being updated at least monthly (Statista, 2025). In light of these frequent changes, it becomes crucial to regularly validate forensic tools' ability to correctly extract traces resulting from application use.

This paper aims to advance this area, and to encourage further work developing techniques to ensure quality and correctness of results from automation in digital forensic tools, which will face new challenges with the introduction of AI capabilities into the workflow (Scanlon et al., 2023).

* Corresponding author.
E-mail addresses: a.claij@nfi.nl (A.A. Claij-Swart), e.oudsen@nfi.nl (E. Oudsen), b.timbermont@nfi.nl (B. Timbermont), christopher.hargreaves@cs.ox.ac.uk (C. Hargreaves), lena.lucia.voigt@fau.de (L.L. Voigt).

[1] Abstraction layers are a function of inputs, rules, and outputs.

https://doi.org/10.1016/j.fsidi.2025.301985

Available online 3 November 2025

This paper is applicable to developers of digital forensic tools, but also to forensic labs that require their own internal testing, or researchers interested in tool errors. The paper makes the following contributions:

- An open-source tool, Puma,[2] designed to automate user actions on Android devices, which unlike existing user interface (UI) automation frameworks allows concise and readable code to be written to perform a variety of tasks within supported applications. This includes sending messages, taking pictures, and location-dependent activities. This reduces the time and effort required to generate comprehensive and representative reference data.
- The overall validation workflow used to ensure correctness of results from a complex digital forensic tool, with regard to application updates.
- Demonstration of the use of this workflow employed in practice for testing Hansken, a Digital Forensics as a Service (DFaaS) platform presented in prior work by van Baar et al. (2014) and van Beek et al. (2015, 2020). Our demonstration highlights insights on changes in application artifacts within a time frame of six months.
- A set of reference data capturing different versions of application data that can be used to test other forensic tool implementations, as well as the corresponding scripts used to generate the data.
- Additions to the SOLVE-IT knowledge base, representing the techniques developed in this work and the weaknesses they mitigate.[3]

Both the code for Puma and the dataset are publicly available.[4,5] The remainder of this paper is structured as follows: Section 2 provides background and related work. Section 3 clarifies the scope of the paper, and Section 4 introduces the architecture of the developed data synthesis framework. The value of the approach is demonstrated in Section 5, with examples highlighting three use cases of the automation framework. Section 6 provides a discussion, including considerations on the generalizability of the proposed approach for forensic tool testing. Conclusions are provided in Section 7.

2. Background and related work

This section provides background and related work, specifically covering automation in digital forensic tools, tool testing, reference data, and automated dataset generation.

2.1. Automation in digital forensic tools

Michelet et al. (2023) defines automation as "software or hardware that completes a task more efficiently, reliably, or transparently by reducing or removing the need for human engagement". Automation in digital forensic tools has been discussed as early as 2003, with tools needing to translate data through one or more abstraction layers so it can be understood (Carrier, 2003a) and to solve the 'complexity problem'. It was also highlighted that each abstraction layer does not only have an output, but also a margin of error. There have also been other early explicit warnings about the dangers of automation in digital forensics: "Another common mistake made by inexperienced individuals is over reliance on user-friendly or automated forensic software" (Casey, 2006). More recently Hargreaves et al. (2024b) demonstrated specific problems in digital forensic tool processes including error propagation.

Therefore, automation is necessary to handle the complexity problem, has become increasingly important in digital forensics, and, as described in the introduction, expectations have increased that the results of automation are correct (UK Forensic Science Regulator, 2020). A significant contributor to meeting that expectation is testing digital forensic tools.

2.2. Tool testing

Horsman (2019) describes a survey about tool testing and the results suggested a reliance on vendor tool testing (question 3). This was echoed in the DFPulse Practitioner Survey 2024; Hargreaves et al. (2024a) indicating significant interest in, and reliance on tool testing to validate tools.

However, this remains a challenging area: "developing extensive and exhaustive tests for digital investigation tools is a lengthy and complex process" (Guo et al., 2009). Guttman et al. (2011) describes that the NIST *Computer Forensics Tool Testing (CFTT)* program has been active since 2000 and documents insights of the first ten years of this program. It also highlights that since 2008, test reports for mobile device forensic tools have also been published alongside write blocking and disk imaging tools. The CFTT testing methodology is described as "functionality driven" (NIST, 2019) and the Scientific Working Group on Digital Evidence (2018) proposes minimal testing guidelines for different categories of digital forensic tools, including the type of test that needs to be performed, the recommended testing frequency, and the appropriate entity for carrying out the test (e.g., vendor, lab, third party etc.). Others have also suggested that anti-forensic techniques must also be addressed in tool testing requirements (Wundram et al., 2013).

Baggili et al. (2007) also advocated for continuous tool testing, especially in the area of mobile forensic tools, given frequent updates. Lyle et al. (2022) also stated that while general validation of a specific tool version may be conducted centrally and shared, testing needs to be repeated when technologies change.

Marshall and Paige (2018) identified a lack of clearly formulated requirements in digital forensic methods and proposed a publicly available set of requirements for digital forensic methods and tools to improve transparency in tool testing. Horsman (2018) also discussed the need for rigorous and transparent validation techniques for digital forensic tools due to the interaction of tool errors and limitations, with incorrect use by users. Subsequently, Marshall (2021) elaborated on approaches that enable tool vendors to establish transparency and trust in their tool's compliance with specified requirements without revealing tools' inner workings. The introduction of generative AI into digital forensics also raises further concerns over transparency and trust (Webb et al., 2024), and the need for systematic validation gains further significance (Wickramasekara et al., 2025).

Brunty (2023) outlined the steps needed for internal tool testing as: defining the testing scope, obtaining a suitable test dataset, performing tests in a controlled environment, and assessing the test results based on expected results. Dataset acquisition is described as potentially the most challenging part of the validation process as it needs "a variety of different use cases [such as] testing and evaluation of a forensic tool's capabilities such as extracting a specific artifact".

2.3. Reference data

A crucial requirement for effective forensic tool testing is the development of reference data, which "consists of test scenarios (cases) against which a EE (Electronic Evidence) tool or its individual function is validated." (Guo et al., 2009).

Early work on reference data includes the *Digital Forensics Tool Testing Images* by Carrier (2003b), a set of small, synthetic test images created between 2003 and 2010, targeting areas such as partitioning, file system, carving, and memory analysis. Others have created a SQLite3 dataset with 77 databases comprising different corner cases that

[2] Puma stands for Programmable Utility for Mobile Automation.
[3] https://github.com/SOLVE-IT-DF/solve-it/pull/106.
[4] https://github.com/NetherlandsForensicInstitute/puma.
[5] https://doi.org/10.5281/zenodo.16579435.

can be encountered in such databases (Nemetz et al., 2018). Prominent collections of digital forensic datasets are *Digital Corpora*[6] and *Computer Forensic Reference Data Sets (CFReDS)*.[7]

However, since Garfinkel et al. (2009) called for standardized corpora, the need for reference datasets remains an ongoing issue for digital forensic tool testing, as well as digital forensic research and education in general. This has been discussed in multiple works (Yannikos et al., 2014; Grajeda et al., 2017; Horsman and Lyle, 2021; Göbel et al., 2025), with Hargreaves et al. (2024b) emphasizing the importance of modular, error-focused datasets, and Gonçalves et al. (2022) assessing the availability of mobile device datasets "as one of the main fields of missing datasets".

Importantly, Spichiger and Adelstein (2025) urged the preservation of reference data for systems that are constantly changing since "software updates may change basically any aspect of a system" and "these changes could lead to a different interpretation of found traces and therefore negatively impact their potential evidentiary value". They also note that retrospective collection of reference data might be challenging or impossible, so preservation must be timely.

2.4. Automated dataset generation

While real-world data can be valuable for forensic tool testing, it has limitations, such as privacy and legal concerns, restricted availability, or missing ground truth (Garfinkel et al., 2009). Synthetic data is an alternative that can be generated on demand and tailored to specific scenarios (Du et al., 2021). However, the manual creation of synthetic data is time-consuming and error-prone. There has been some work specifically targeting the automated creation of synthetic forensic data for testing forensic tools. Yannikos et al. (2011) and Yannikos and Winter (2013) proposed and later implemented a model-based method for the creation of synthetic test disks that involves the simulation of user activity in a scenario. Visti et al. (2015) introduced ForGe, a tool that automates the generation of test disk images, comprising the creation of NTFS file systems and data-hiding techniques.

Moreover, different approaches and corresponding tools for the automated synthesis of forensic data have been presented for teaching purposes (Moch and Freiling, 2009; Scanlon et al., 2017; Du et al., 2021; Göbel et al., 2022; Schmidt et al., 2023; Wolf et al., 2024; Voigt et al., 2024). Apart from Scanlon et al. (2017), who proposed creating baseline disk images and injecting artifacts into them, they all encompass automating control of a virtual machine, including the simulation of user activity, to reduce the manual effort required for the generation of synthetic data while increasing their resemblance to real-world disks.

While these approaches have focused on teaching scenarios, others have demonstrated alternative uses of such approaches for tool testing. Notably, Rzepka et al. (2025) utilized ForTrace++ (Wolf et al., 2024) to automate the generation of an extensive dataset of 1600 main memory dumps to assess inconsistencies in main memory dumps acquired with four different tools.

There is also limited work on synthesizing mobile device datasets. Ceballos Delgado et al. (2022) introduced FADE, a tool that enables the injection of artifacts, namely text messages, contacts, calls, and files, into rooted Android emulators. Michel et al. (2022) presented the proof-of-concept tool, AutoPoD-Mobile, which works on a restricted set of five specific physical Android devices of different vendors and different Android versions as well as one iOS device, for which they implemented only limited functionality. With this, they proposed an approach for creating a diverse range of artifacts (e.g., contacts, calls, WhatsApp and email communication, pictures) on mobile devices by either injecting them or by approximating user activity on the devices using ADB, public APIs of applications, or web clients (e.g., WhatsApp communication via WhatsApp Web). They also utilized the Appium Settings application on target devices to enable static location spoofing. They demonstrated their approach with a multi-device setup, allowing them to synthesize mobile data for multi-person scenarios.

Demmel et al. (2024) introduced an approach that simulates user activity on Android devices via the device's UI. They provided a proof-of-concept implementation that supports the simulation of gestures, clicks on UI elements, and external events. These included incoming calls or text messages, as well as GPS coordinate changes. To simulate the external events, they established a telnet connection to send commands to the emulator. However, in their current implementation, only rooted Android emulators and single-device settings are supported. The simulation of multiple devices and communication between them is not possible. Also, the use of a rooted device prevented them from native access to Google's Play Store.

2.5. Summary

Increasing use of automation in digital forensic tools requires thorough tool testing programs that can keep up with rapid changes in technology, particularly in the ever-evolving mobile space. There is a strong case to use automation to assist with this task. Some work exists in this area, e.g., Demmel et al. (2024), but there is a significant need for a practical solution that integrates well with tool development and testing workflows, that is responsive to target application updates and beyond the proof-of-concept stage. The following sections describe the development of an automation framework that is focused on generating data to assist with this tool testing.

3. Scope

This section outlines the scope of this paper, covering the aim, the focus on automated test data generation, and the three existing use cases where this automation can be integrated into a broader tool testing architecture. Referring to the tool testing procedure described in Brunty (2023), we focus on the two initial steps: defining the test scope and obtaining the dataset, with emphasis on the latter, which the authors described as the most demanding part of the process.

3.1. Aim and motivation

Due to the rapid pace of updates to mobile applications, a new approach is needed to perform complete validation of forensic tools, preferably for every application version. To ensure full coverage, this process also needs to happen within a certain time span of the version's release, since server-dependent applications might not allow older versions to be used. Once an application version is no longer supported by the server, it will be impossible to create new reference data of that version (Spichiger and Adelstein, 2025). Due to this combination of factors and the resulting effort and challenges of manually creating reference data and validation of forensic tools, we propose an automated solution. Such a system needs to perform several tasks:

- Generate reference data for new application versions
- Process this reference data using the forensic tools under test
- Verify the output of the forensic tools against expected results

For the first step, which is the focus of this paper, the automatically generated reference data needs to be *representative* and *comprehensive*, discussed in the following subsections.

3.1.1. Representative reference data

Voigt et al. (2025) formally defined a restricted form of *realism* in synthetic forensic data as being indistinguishable from real-world data with respect to a restricted set of allowed features. This mirrors intuitive definitions from previous literature. In the context of our tool testing

[6] https://digitalcorpora.org/.
[7] https://cfreds.nist.gov.

approach, *realistic* reference data would, therefore, not entail the synthetic data being indistinguishable in every aspect from data generated by users in real-world settings, but only in aspects relevant to our aim. To avoid confusion with more broadly defined concepts of realism in synthetic data, we refer to the quality of the reference data being indistinguishable from real-world data in selected aspects as *representative* data.

It is critical that the data is created by interacting with the application in question, simulating user activity, rather than using alternative methods such as injecting entries to a database or appending log files. While these methods might be suitable for other use cases, they contradict the goal of our approach to identify changes in application artifacts, as injection techniques necessitate knowledge about the storage data structure in the first place.

In contrast, to validate the output of forensic tools, it is not necessary for the content of the reference data to follow a coherent narrative. Additionally, some traces of the synthetic nature of the data are not relevant to the approach we propose, e.g., traces of the automation framework on the target device in general or a flag[8] hinting that a location trace stems from spoofing, as found by Demmel et al. (2024).

3.1.2. Comprehensive reference data

Comprehensive reference data in this context relates to the supported features of the forensic tool under test. For complete validation of its extraction features for a specific application, the reference data used for validation should contain artifacts for each feature supported by the forensic tool. For example, if a forensic tool supports recovering messages and pictures from a chat application, but not videos, the reference data is comprehensive when it contains only messages and pictures.

Notably, some features may have a multiplicative effect on the required volume of reference data. For example, a forensic tool supporting chat messages and pictures requires reference data with at least four artifacts: a message and a picture, both sent as well as received. When the forensic tool adds support for group chats, those four artifacts are needed again in the context of a group chat, doubling the required artifacts to eight. With the rapid growth of features in mobile applications, the number of artifacts needed in comprehensive reference data can create substantial scalability challenges, reinforcing the need for an automated approach.

In addition to the generated data itself, the provision of ground truth is also advised (Breitinger and Jotterand, 2023). Besides conventional ground truth logs in machine-readable format, this can also include making the generated test data *self-describing*, e.g., a message might be "This is a WhatsApp message from Alice to Bob, sent on 2025-05-20 at 13:00". This makes analysis and manual verification of tool results easier.

3.2. Applying automated reference data creation

To address the need for automated creation of reference data, we have developed Puma, a tool that enables and simplifies the automated execution of actions on mobile devices, and the architecture is discussed in Section 4. Puma forms a central part of the Hansken validation workflow, see Fig. 1. This shows that the complete process can be automated: applying application updates, creating reference data, extracting the data, processing with the forensic tool under test, and verifying its outputs.

Within this workflow, there are three use cases in which Puma can be used to create reference data for applications, including handling application updates. The three use cases are described in the following subsections and demonstrated in the context of tool testing for Hansken in Section 5.

[8] If this flag is not covered by the forensic tool's supported features, see Section 3.1.2.

3.2.1. Initial data population

The first use case is the initial creation of reference data for a mobile application for which support will be added to the forensic tool in question. This reference data can then be used to investigate the files in which the application stores data, helping the initial forensic tool development. After this development, the reference data will be used in a Continuous Integration and Continuous Deployment (CI/CD) pipeline to ensure that future code changes do not introduce regression in support.

Creating such data requires planning of the actions to be performed within the application, as well as precise execution. This often involves multiple devices, since interaction between users is a key characteristic of many applications, e.g., messaging applications. As discussed in Section 3.1.2, the number of required actions depends on the range of features supported by the application, and can increase rapidly due to multiplicative effects. Furthermore, the execution of actions must be logged to establish ground truth, which should capture the time at which actions are executed and any deviation from the planned actions.

The automation framework presented in this paper aims to address these problems by allowing researchers and developers to populate applications with data through script execution. We have used Puma to generate reference data for multiple applications and present the results in Section 5.1.

3.2.2. Post-update data population: application launch only

The second use case involves validating a forensic tool after an application update, assuming that the application is already supported. In this scenario, the application has been populated with data on the target device prior to the update. The update may alter the way the application stores data, e.g., by updating the database schema, and also migrate existing contents. This may therefore prevent a forensic tool from correctly parsing the data. On Android, the popular messaging application WhatsApp behaves in this way: when an update introduces a new database schema, the existing database on the device is migrated without requiring user interaction.

The validation step in this use case is straightforward: since the application was populated prior to the update and no further actions were performed within the application, the output of the forensic tool should remain unchanged after applying the update. We have applied this approach to validate our tools against new versions of the WhatsApp messaging application, which is demonstrated in Section 5.2.

3.2.3. Post-update data population: extended user actions required

The third use case extends the second by incorporating UI actions executed by Puma. While the process described in the second use case is effective for applications that automatically migrate stored data to a new format upon updating, some applications behave differently. In these cases, data is not migrated during an update. Instead, the new format is used only for data created after the update.

One such application is Telegram, which stores user data (e.g., messages and contacts) as Binary Large Objects (BLOBs) in a database (Jaeckel et al., 2025). The structure of these BLOBs can change between versions, but only new entries will use the new structure while existing data remains unchanged. Consequently, actions must be performed within the application post-update to generate reference data reflecting the latest version. This is illustrated in Section 5.3.

The validation approach here is mostly consistent with that in the second use case, but in this case, as new actions are carried out which supplement the existing data, the forensic tool's output is expected to change. Therefore, the overall validation approach must be aware of the new executed actions and verify whether the forensic tool correctly detected them, while still correctly processing the old data.

4. Puma – a framework for automated synthesis of mobile data

In this section, we describe the requirements, architecture, and

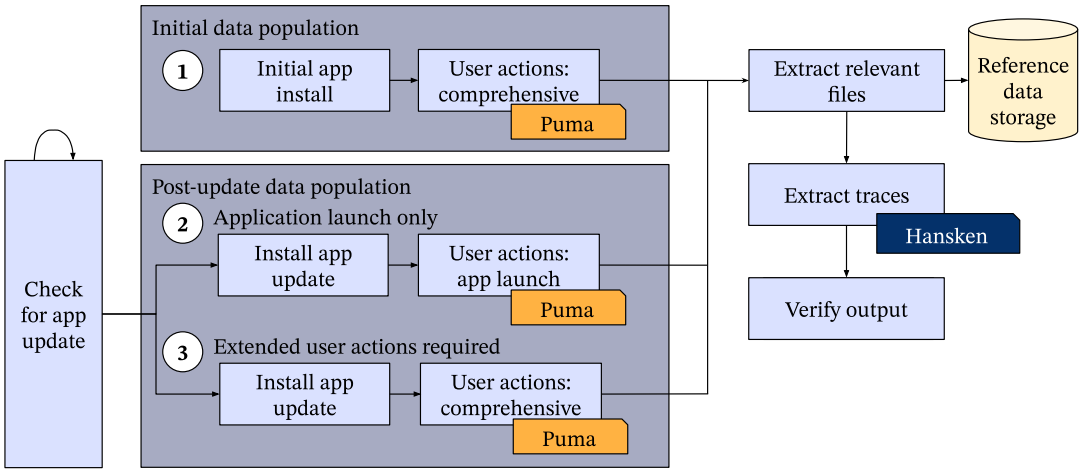

Fig. 1. Tool validation workflow triggered by application updates. ①, ② and ③ correspond to the use cases of Puma within the workflow discussed in this paper.

```
000 42 52 34 94 02 03 00 00 00 00 00 00 38 02 00 00 22 BR4.........8..."
011 17 51 59 3D B1 D4 C0 01 00 00 00 00 22 17 51 59 0F 94 .QY=........".QY..
022 8B AD 01 00 00 00 B0 A8 63 67 47 48 65 6C 6C 6F 20 ........cgGHello
033 42 6F 62 2C 20 74 68 69 73 20 69 73 20 61 20 6D 65 Bob, this is a me
044 73 73 61 67 65 20 66 72 6F 6D 20 41 6C 69 63 65 20 ssage from Alice
055 73 65 6E 74 20 61 74 20 32 30 32 34 2D 31 32 2D 31 sent at 2024-12-1
066 39 20 30 35 3A 30 31 3A 32 37 20 55 54 43 20 63 ED 9 05:01:27 UTC c.
077 3D 01 20 00 00 =. ..

000 E9 BB FD 96 02 01 00 00 00 00 00 00 47 02 00 00 22 ............G..."
011 17 51 59 3D B1 D4 C0 01 00 00 00 00 22 17 51 59 0F 94 .QY=........".QY..
022 8B AD 01 00 00 00 D3 9B 76 67 47 48 65 6C 6C 6F 20 ........vgGHello
033 42 6F 62 2C 20 74 68 69 73 20 69 73 20 61 20 6D 65 Bob, this is a me
044 73 73 61 67 65 20 66 72 6F 6D 20 41 6C 69 63 65 20 ssage from Alice
055 73 65 6E 74 20 61 74 20 32 30 32 35 2D 30 31 2D 30 sent at 2025-01-0
066 32 20 31 33 3A 35 39 3A 33 38 20 55 54 43 01 20 00 2 13:59:38 UTC. .
077 00 .
```

Fig. 2. BLOBs of a regular message with similar content in Telegram v11.5.5 (above) and v11.6.1 (below). The first four bytes (orange) are a header used by Hansken to determine how to parse the BLOB. When this header changed, Hansken needed to be updated. The message text is highlighted in blue.

currently implemented features of our open-source mobile data synthesis framework, Puma.

4.1. Requirements

The development of Puma was driven by a set of requirements. First, the tool must be capable of automating actions on Android devices, both emulated and physical, and must be operable from host systems running macOS, Windows or Linux. Second, the tool must support the generation of comprehensive test data, as explained in Section 3.1.2. This means that the tool should focus on supporting a wide range of application features within selected apps, rather than the creation of complex narrative-driven scenarios that are coherent across a broad range of diverse applications. Third, the tool must ensure that all application data is created by the application itself, and must not rely on alternative methods such as artifact injection. Finally, the tool should enable its users to easily specify what actions should be performed without requiring them to detail exactly how those actions are carried out on the target device.

4.2. System design and capabilities

Puma focuses on combining atomic UI actions into high-level activities such as sending a message or creating a group chat. This reduces the complexity involved in writing automation scripts and eliminates the need for detailed knowledge about the target device's UI. The target device can be either a mobile device emulator running on the host computer or a physical mobile device connected to it via USB, with ADB access enabled. Puma itself does not require root access. However, root access may be required to facilitate the subsequent acquisition of forensic artifacts, as it enables access to all system files.

The framework also supports ground truth logging during simulation. When an action is performed, it can record the time it was initiated. Script authors may include additional logging, e.g., by using Puma to create screenshots.

Furthermore, Puma provides *general-purpose* and *application-specific* functionality. General features include application launch and termination, capturing screenshots and screen recordings, scrolling, and finding text in non-text elements through Optical Character Recognition (OCR). Moreover, Puma provides dynamic location spoofing capabilities, enabling the simulation of movement along a route between two predefined locations, thereby improving upon previous work which allows for spoofing static locations only (Michel et al., 2022; Demmel et al., 2024). The design of Puma emphasizes modularity and extensibility. Each supported application is represented by a dedicated class encapsulating the methods required to perform application-specific actions, while also providing access to general functionality.

At the time of writing, Puma supports eight applications, with supported features varying per application (see Table 1). The main objective is the automated generation of reference data for applications frequently encountered in real-world investigations (e.g., WhatsApp, Telegram, and Snapchat), and therefore necessitating support in forensic tools.

Table 1
Applications and their features currently supported by Puma.

Application	Supported features
Google Camera	Taking pictures
Google Chrome	Visiting URLs, Google search via omnibar, managing and bookmarking tabs
Google Maps	Searching location, navigating (including location spoofing)
Open Camera	Taking pictures and videos
Snapchat	Sending messages and snaps
Teleguard	Sending messages, pictures, sending and accepting invites
Telegram	Sending messages and pictures, making calls
WhatsApp	Sending messages, pictures and videos, location sharing, making calls, creating and managing groups

Some additional applications are supported for demonstration purposes, for instance, the messaging application TeleGuard can be used to easily demonstrate Puma's functionality, as it does not mandate registration with a phone number. Similarly, Google Maps was supported to visually demonstrate the location spoofing feature, but it is also a mainstream application worth monitoring. Although this paper focuses on single-application scenarios, Puma supports concurrent control of multiple devices and applications, enabling more complex simulation scenarios.

4.3. Implementation overview

This section provides an overview of the most relevant parts of the Puma implementation. Further details on the implementation as well as extensive documentation can be found in the Puma repository.[9]

4.3.1. An Appium-based implementation

Puma is implemented in Python and is designed as a high-level abstraction layer built on top of Appium, an open-source automation framework for UI testing that supports various types of platforms, such as mobile, web browser, and desktop (OpenJS Foundation, 2025a). Appium is widely used by developers to test the UIs of their mobile applications for different platforms (e.g., iOS/Android), using the same API. Instead of its typical use in application testing, Puma uses Appium to generate reference data by automatically simulating user activity on a target device.

To understand Puma's implementation and the implications for its forensic data synthesis approach, it is crucial to understand how Appium works (OpenJS Foundation, 2025b). Appium itself operates on a client-server architecture. The Appium server runs on a host workstation and communicates with a connected target device over ADB to install Appium's helper application on it. Puma uses the Appium Python library to send commands to the Appium Server, which then executes them on the device. Appium automates application actions through its helper application hooking into the accessibility framework of the target device to interact with the UI elements, simulating user actions such as tapping buttons, entering text, and swiping.

While Appium offers a straightforward approach for defining atomic actions (i.e., locating UI elements, simulating clicks and swipes, or entering text), to represent higher–level activities that comprise multiple actions, several lines of code are required. This code can become verbose, reducing readability. Puma's abstraction from these atomic actions reduces the complexity involved in writing higher–level activities needed for automation scripts, making Puma code notably more concise and maintainable (see Appendix A).

4.3.2. Adding app-specific actions

As mentioned in Section 4.2, each supported application is implemented as a separate class with methods to perform app-specific actions. These app-specific classes extend the `AndroidAppiumActions()` base class, which provides Puma's general-purpose methods through inheritance. Further details of implementing an app-specific actions in Puma are provided in the project repository documentation and below is a short description. Note that this detailed understanding of Appium is needed for development of new app-specific support, but not for general use of Puma.

In brief, development involves first identifying the corresponding UI element before it can be interacted with. The former can be done by calling Appium's `find_element()` method, which searches for the element in an XML representation of the UI. This method allows selecting the element by different attributes, such as class name and accessibility ID.

Apart from single attributes, elements can also be selected using the XML Path Language (XPath), a query language allowing to query elements by their attributes in an XML document (World Wide Web Consortium (W3C), 2025). This enables more complex queries, such as selecting elements based on multiple attributes, their parent, or sibling elements. Subsequently, the element can be interacted with in different ways depending on the type of element. For instance, buttons are typically engaged through the execution of a click action, denoted by the method `click()`; text fields are populated with text input using the `send_keys()` method.

4.3.3. Location spoofing

A specific feature, 'dynamic location spoofing' is implemented in the `RouteSimulator()` class. Puma offers this feature to facilitate the simulation of movement along a requested route at a specified speed, as Appium's location spoofing is restricted to static locations. The speed at which the route is traversed can be specified at any time during the movement simulation to mimic more realistic movement patterns.

5. Demonstration

In the following section, we report on experiences and observations of using Puma from November 2024 to May 2025, as part of the tool testing process for Hansken. Puma enables us to validate that Hansken supports the latest version of an application the day it becomes publicly available. By using Puma actions on each new version, we can assess whether Hansken's parsing of the application data is still correct, or if updates are needed. It allows the exact versions of applications supported to be specified.

To illustrate the use of Puma, we revisit the three use cases introduced in Section 3.2 and discuss both outcomes and insights gained from its real-world integration.

5.1. Initial data population

The first use case involves extending forensic tool support to extract traces of a new application. This process starts with generating reference data specific to the target application, which serves as a basis for validation. Incorporating a wide range of user actions during data generation is desirable to capture the diverse traces that these actions produce.

An example reference dataset created by Puma for WhatsApp can be found in the repository, including the script used to create it. Listing 1 displays an excerpt of the script used for data synthesis. It includes messages and pictures being sent between two users, with *self-describing* message contents.

We observed that leveraging Puma for the purpose of initial data population reduces errors associated with manual data generation and enhances the reproducibility of reference data generation. This is particularly valuable when it becomes necessary to expand the dataset to cover additional application behaviors. For example, if an additional feature becomes available in WhatsApp to send stickers, the code outlined in Listing 2 can be integrated into our existing scenario. Lastly, it

```
alice_wa = WhatsappActions(device_udid='emulator-5554')
bob_wa = WhatsappActions(device_udid='emulator-5556')
alice_wa.create_new_chat(
    contact="Bob",
    first_message="This is the first message from Alice to Bob,"
                  "sent on 2023-04-12 at 13:35")
bob_wa.send_message(
    message_text="This is the first message from Bob to Alice,"
                 "sent on 2023-04-12 at 13:37",
    chat="alice")
alice_wa.send_media(directory_name="tiger",
                    caption="This is an image of a tiger")
bob_wa.send_media(directory_name="elephant", view_once=True)
```

Listing 1. Excerpt of a Puma script used for initial data population of Whatsapp.

[9] https://github.com/NetherlandsForensicInstitute/puma.

```
alice_wa.send_sticker(chat="Bob")
bob_wa.send_sticker(chat="Alice")
```

Listing 2. Expansion of Listing 1 with additional functionality for WhatsApp.

significantly reduces the time required for data generation as the process is automated and the scenarios can be reused.

5.2. Post-update data population: application launch only

The second use case concerns the handling of updates to applications that migrate their data to new formats following an update. Again using WhatsApp as a case study, we observed it to be updated frequently. In the aforementioned time frame of six months, there have been version releases on 78 days (an average of three releases per week). This was checked by automatically downloading the Android Application Package (APK) from the website of WhatsApp[10] daily, and recovering the version from it using the Android Asset Packaging Tool for Python3 (HuMoran, 2022).

In this scenario, the application was first updated on the test device to the most recent version using the ADB command `adb install whatsapp.apk`. Subsequently, the application was launched and then closed, after which the database was extracted from the device using `adb pull database.db`. This enabled the examination of whether the underlying database schema had changed between versions.

In our tool validation workflow for Hansken, each step mentioned in this scenario was performed automatically, requiring no manual intervention. The interaction with the target application was scripted using Puma, with only a single statement needed to simulate a user tapping the application icon on the home screen, thereby launching WhatsApp on the target device (see Listing 3).

Using the tool validation workflow, we have observed a high degree of stability for the WhatsApp database schema. During a monitoring period exceeding three years, from March 2022 to May 2025, the schema had undergone only two notable changes that broke the support of Hansken (see Table 2).

The schema update observed in version 2.22.14.70 was notable, as it was applied at multiple different application versions on other devices we use. This observation suggests that the change was not solely triggered by the installed update, but was also pushed from the WhatsApp servers. Further investigation was beyond the scope of our current study.

We also noticed that WhatsApp occasionally retracted previously released versions of its application. In some cases, the revoked version was replaced by an earlier one that had been available on the official website. In other instances, it was substituted with a release that had a lower build version number but a higher revision version number. These insights are difficult to obtain without the automated framework.

5.3. Post-update data population: extended user actions required

The third use case addresses application updates for Telegram, which we observed to be less frequent than those for WhatsApp. Still, the updates occur regularly, with 22 updates recorded in the time frame mentioned earlier (an average of approximately one update per week). As with WhatsApp, this was determined by downloading the APK daily from the Telegram website[11] and extracting version information.

```
alice_wa = WhatsappActions(device_udid='emulator-5554')
```

Listing 3. Initialization of the WhatsApp application.

[10] https://www.whatsapp.com/android.
[11] https://telegram.org/android.

Table 2
Example changes in Whatsapp detected by the tool validation workflow.

Date/version	Details
2022-03-22, version 2.22.7.73	The default value for timestamps changed from null to −1. This caused Hansken to interpret these as timestamps from the date 1969-12-31.
2022-06-28, version 2.22.14.70	A major database schema change migrated the content of the messages table to the message table with different columns. The result was that Hansken was unable to extract any messages[a].

[a] An example of two versions of WhatsApp databases containing this schema change is provided in the repository.

Despite the lower update frequency, each Telegram update was observed to pose a greater risk of disrupting support in Hansken: there were two breaking changes in six months. The broken support was primarily due to the structural differences in how Telegram stores data in its internal databases compared to WhatsApp. Specifically, when a new version of Telegram is released, the database schema remains unchanged and existing data is maintained in its original format. However, new user data (e.g., a new message) is stored using a new BLOB layout, with existing messages using the old format. This somewhat mirrors changes in Messages from iOS 10 to iOS 11 retaining both old and new timestamp formats within a single database (Barnhart, 2017). Scenarios like these can be handled with the use of Puma to expand the data post-update.

In this use case, the application was first updated to the latest version. Subsequently, a series of Puma actions were executed to simulate user interactions, such as sending messages or images and initiating voice calls to populate the application's database with data of usage under the updated version. Once these interactions were complete, the database was extracted from the device and analyzed to assess whether the forensic tool continues to correctly parse the updated Telegram data structures.

Fig. 2 illustrates a change between Telegram version `11.5.5` to `11.6.1` (released on 2025-01-02), which prevented parsing by Hansken due to the updated header at the start (the first 4 bytes). This was detected using Puma within 24 h of the update had been released.

5.4. General observations

In summary, the adoption of Puma has enabled a shift from manual execution of validation scenarios on mobile devices to fully automated, script-driven workflows. Previously, scenarios were documented as textual instructions requiring manual execution. These have now been replaced with clear, maintainable Python scripts, allowing efficient and reproducible generation of reference datasets.

We have manually performed the process of generating reference data in this context for years and found it to be both error-prone and labor-intensive. Due to the need for multiple devices, thorough documentation, and growing complexity of application features, it often required several hours of work by at least two people. With Puma, this process can now be carried out in less than an hour by a single person, with a reduced risk of error.

This transition has not only streamlined the creation of reference data, but has also facilitated daily automated testing of Hansken, particularly in relation to detecting application updates. It is difficult to quantitatively assess the impact of this automation on the tool's reliability, due to the lack of systematic failure tracking prior to implementation. However, it is evident that this approach has improved our ability to detect application updates breaking support of our forensic tool. It has also improved our ability to respond to them in a timely and consistent manner and provide clear indications of which versions of an application are tested and supported.

6. Discussion, limitations, and further work

This paper has described the automated dataset generation tool Puma and demonstrated its use. It has also situated Puma within a broader forensic tool testing workflow. While the demonstrations clearly show its benefits, there are still limitations. This section discusses limitations and future work of Puma, the tool testing workflow, and the broader applicability.

6.1. Puma

While the benefits of using Puma have been shown in Section 5, there are additional examples that have not been presented in this paper. To illustrate these broader capabilities, an example script presenting the interaction between two emulators, using multiple applications, is provided in Appendix B. It includes the use of Google Maps to demonstrate the location spoofing functionality mentioned in Section 4.2 and shows how Puma may facilitate the creation of arguably realistic, multi-application scenarios, making it a valuable tool for training and educational purposes. Its scripts are aimed at having good readability and are easily adaptable, thereby reducing the barrier to entry for simulating complex mobile interactions, even for users with limited technical expertise.

Despite the benefits offered by Puma, it is important to acknowledge that it may not always be the optimal choice for dataset generation. The time required to develop Puma scripts should be weighed against potential gains in efficiency and reproducibility. Table 3 outlines scenarios where Puma offers advantages, as well as cases where manual approaches may be more suitable.

Furthermore, there are also specific limitations and further work for Puma. Its interaction with applications is fundamentally dependent on Appium, which leads to some inherited limitations. First, it requires the development of custom automation scripts for each individual application, preventing the use of generalized scripts for common actions across multiple applications, such as sending messages in various chat applications, even when those applications exhibit similar UI patterns. There are also occasional issues with the underlying Appium implementation, e.g., due to differing loading times of application elements. Like all UI-based automation frameworks, it is sensitive to differing "user journeys" through applications, e.g., presenting a user agreement to accept, popups describing new features, or the requirement to create accounts. These currently need to be handled either with bespoke code or manually bypassed.

Additionally, the use of Appium leaves traces on the target device. In the use cases presented in this paper, with its focus on trace extraction validation, additional traces left by Puma can be disregarded (see Section 3.1.1). However, other use cases, such as those discussed in Section 6.3, will need to identify, document, and potentially scrub those artifacts if that is part of the alternative use case's 'realism' requirement. For example, in case of utilizing the location spoofing mechanisms, it might be necessary to ensure that the application under test treats spoofed locations equivalently to actual location changes, or to identify differences (as done by Demmel et al. (2024)) and assess their relevance for the use case in which the synthetic data is employed.

Other limitations include that, while Puma handles updates to stored data well, some aspects of frequent mobile application updates remain a challenge. In particular, any modification to an application's UI, even subtle or non-visible changes, can disrupt existing Puma scripts. While this can cause initial issues, the code can evolve to handle multiple versions of an application's UI.

To enhance the robustness of Puma, we plan to incorporate a finite state machine capable of recognizing the current state of the application under test. This would allow the framework to automatically detect and respond to unexpected interface elements (e.g., pop-ups), or recover from anomalous states by restarting the application or adjusting the execution flow. The integration of such a state-aware mechanism would improve Puma's fault tolerance, particularly in complex or long-running scenarios.

Finally, we are exploring the potential for integrating artificial intelligence (AI) into Puma's architecture. AI-based techniques could assist in navigating application UIs by reducing reliance on hard-coded UI element identifiers, which are often subject to change during application updates. Furthermore, this may enable the development of generalized automation scripts capable of performing common tasks across multiple applications, thereby overcoming one of the current limitations of Puma's design.

Other future work will focus on enhancing Puma's functionality, broadening its platform and application support, and increasing its robustness and flexibility. The selection of additional applications developed internally will be driven by practical requirements, particularly those aligned with expanding the capabilities of Hansken. Application support will be prioritized based on forensic relevance and demand from investigative contexts. However, as the project is open-source, with detailed documentation, other application support can be added by the broader digital forensics community for any use case.

6.2. Forensic tool testing

Puma's area of application lies within a broader tool testing architecture, as illustrated earlier in Fig. 1. The use case demonstrated in Section 5.2 showed that for some applications, such as WhatsApp, additional user actions are not required to update the application data before extracting it for validation of a new application version. However, there are benefits to incorporating automated data generation in this situation as well. For example, if a WhatsApp update introduces a change such that UI interaction is newly required to trigger post-update behavior, the approach that omits UI actions would falsely report that the forensic tool supports the latest version. In contrast, an approach incorporating UI actions would correctly identify the breaking change.

There are other considerations within the broader architecture. An example is identifying when an update is necessary. This is currently a scheduled check every 24 h, triggering the re-evaluation process if the application version is different. However, some applications may retain the same version but require revalidation between releases if they heavily utilize server-side data. This is particularly true when trace extraction depends on API access records or internally cached artifacts. Techniques for detecting these server-side changes would also be beneficial.

Another aspect that was not explored was the synchronization of user data of a user logged onto multiple devices. Since Puma can use any

Table 3
Comparison of conditions favoring Puma versus manual data generation.

Consideration	Puma-based data generation	Manual data generation
Support	Puma already supports the application or feature	Requires weighing effort against Puma support development
Repetition	Dataset will need to be generated repeatedly, e.g., across different testing cycles or application versions	Dataset will only be created once
Data Volume	Large volume of data is required, potentially involving multiple applications and device configurations	Only a limited number of simple actions are required
Intrusiveness	Additional traces left by underlying automation framework (i.e., Appium) are not a concern	The dataset must closely mimic real-world conditions; automation framework are unacceptable

number of devices, in future work Puma can be used to investigate how apps store such synchronized artifacts.

Puma is focused on generating reference data for Android applications. There is therefore an obvious need for a complementary approach for iOS. Appium, which underpins Puma, also supports iOS, so it is possible for new scripts to be created to perform automated data generation here too. We anticipate the reuse of substantial portions of the existing Android-focused implementation, which would facilitate a more efficient development process and enable Puma to support a wider range of mobile devices and ecosystems.

While Puma is integrated into the validation workflow for Hansken, it can be used to generate reference data for any forensic tool. In addition, the overall tool validation workflow can apply to any forensic tool, but with caveats. For example, the ability to automatically extract traces from the reference data is possible here because of direct access to the internal capabilities of Hansken. Other tool developers would likely have similar capabilities and the overall approach can be generalized. However, if the validation needs to be performed in a forensic lab or by another third-party, programmatic access to forensic tools is not always available. However, with innovative approaches such as the use of ForTrace++ for UI control of digital forensic tools as presented by Rzepka et al. (2025), it is possible to test any GUI-based tool or plugins developed for such tools. However, this solution is far from ideal, and programmatic access (either API or command line) to tools would preferably be available to perform validation. For output verification, the output should be available in a standardized representation such as the CASE ontology (Casey et al., 2017). Validation efforts would benefit from demanding such requirements from all digital forensic tool vendors.

The proposed approach of automated test data generation represents a starting point for a much bigger potential research area: automating digital forensic research. For example, it would be possible for the overall system to detect schema changes in the database used by an application and promptly disseminate this knowledge to the broader community, rather than keeping it exclusively for tool testing. With machine-readable ground truth, locating known items within a modified database or dataset of other artifacts could also be possible, beginning the process of automatically inferring trace locations. The Aardwolf project offers a platform designed to facilitate the sharing of such knowledge (Boztas et al., 2025).

6.3. Broader discussion

Section 4.2 described how high-level activities such as sending a message are recorded as ground truth. This provides a further opportunity to enhance timeline analysis research involving the inference of 'high-level events' from lower-level timeline entries (e.g., Hargreaves and Patterson (2012)). Combining the ground-truth logs with a timeline generated from the test device could facilitate techniques such as machine learning to identify the expected artifacts for a given activity.

There is also the potential to use Puma beyond its current tool testing-focused use case, such as for automated data generation for teaching. However, it would be necessary to either determine the acceptability of Puma's automation artifacts in the concrete teaching context or to minimize their presence, e.g., by sharing only selected portions of the mobile device datasets with students.

Another important direction involves the development of long-duration scenarios and testing. These scenarios would simulate user activity over extended periods, spanning multiple days, in order to generate richer, more temporally diverse datasets. Such datasets could better reflect real-world usage patterns and improve the ecological validity of forensic validation procedures.

7. Conclusions

This paper has highlighted the need for digital forensic research to provide practical solutions to the problems described as early as Garfinkel (2010) such as the dramatic increase in digital forensic tool scope and complexity, and therefore the cost of their development and updates. This is compounded by the update cycles and sheer number of modern mobile applications.

To contribute towards addressing this challenge, this paper has described the development of an automated testing workflow for a major digital forensic tool implementation, and the creation of an automation component to overcome many of the challenges in ensuring that rapidly changing mobile applications are processed reliably by automated forensic tools. It has also outlined some areas to further enhance quality. The hope is that the digital forensic research community can build on this work and develop practical solutions to improve the overall results of automated digital forensic tools.

CRediT authorship contribution statement

Angelina A. Claij-Swart: Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Visualization, Writing - Original Draft, Writing - Review & Editing. **Erik Oudsen:** Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Visualization, Writing - Original Draft, Writing - Review & Editing. **Bouke Timbermont:** Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Visualization, Writing - Original Draft, Writing - Review & Editing. **Christopher Hargreaves:** Methodology, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision. **Lena L. Voigt:** Methodology, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision.

Acknowledgments

We thank Felix Freiling for his valuable comments on an earlier draft of this paper. Thanks to the anonymous reviewers for their constructive feedback and improvement suggestions. We thank the Netherlands Forensic Institute for funding the development of Puma and the validation tool and for providing us with the opportunity to dedicate time to innovation. The work by Lena L. Voigt was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393 541 319/GRK2475/2-2024).

Appendix A. Comparison of Python Appium and Puma code for sending a Telegram message

Appium code

```
driver = webdriver.Remote(command_executor=appium_server,
                          options=options)
driver.activate_app(app_id='org.telegram')
chat_row_elem = driver.find_element(
    by=AppiumBy.XPATH,
    value="//android.view.ViewGroup['Alice']"
)
chat_row_elem.click()
message_text_field = driver.find_element(
    by=AppiumBy.XPATH,
    value="//android.widget.EditText[@Text='Message']"
)
message_text_field.send_keys(value="Hello Alice!")
send_button_elem = driver.find_element(
    by=AppiumBy.XPATH,
    value="//android.view.View[@content-desc='Send']"
)
send_button_elem.click()
```

Puma code

```
bob_telegram = TelegramActions(device_udid='emulator-5554')
bob_telegram.send_message(message='Hello Alice!', chat='Alice')
```

Appendix B. Puma script for a more complex scenario

```
# Alice uses 2 applications on device ‘emulator-5554’
alice_wa = WhatsappActions(device_udid='emulator-5554')
alice_maps = GoogleMapsActions(device_udid='emulator-5554')

# Bob uses 1 application on device ‘emulator-5556’
bob_wa = WhatsappActions(device_udid='emulator-5556')

# Alice and bob send each other a message
bob_wa.send_message(
    message_text='Hey Alice, are you still at the Eiffel Tower?',
    chat='Alice'
)
alice_wa.activate_app()
alice_wa.send_message(
    message_text='I was just leaving!',
    chat='Bob'
)

# Alice starts navigation
alice_maps.activate_app()

# Navigation from point A to point B at 35 kmph
alice_maps.start_route(
    from_query='Eiffeltower',
    to_query='Notre Dame',
    speed=35
)
alice_wa.activate_app()
alice_wa.send_message(message_text="I'm leaving now!")
```

References

Baggili, I.M., Mislan, R., Rogers, M., 2007. Mobile phone forensics tool testing: a database driven approach. Int. J. Digital Eviden. 6, 168–178.

Barnhart, H., 2017. Time is Not on our Side when it Comes to Messages in iOS 11. https://smarterforensics.com/2017/09/time-is-not-on-our-side-when-it-comes-to-messages-in-ios-11/. (Accessed 13 May 2025).

Boztas, A., De Jong, J., Hadjigeorghiou, C., 2025. Argus: a new approach for forensic analysis of apps on mobile devices. Forensic Sci. Int.: Digit. Invest. 53, 301938.

Breitinger, F., Jotterand, A., 2023. Sharing datasets for digital forensic: a novel taxonomy and legal concerns. Forensic Sci. Int.: Digit. Invest. 45, 301562.

Brunty, J., 2023. Validation of forensic tools and methods: a primer for the digital forensics examiner. Wiley Interdisciplin. Rev.: Forensic Sci. 5, e1474.
Carrier, B., 2003a. Defining digital forensic examination and analysis tools using abstraction layers. Int. J. Digital Eviden. 1, 1–12.
Carrier, B., 2003b. Digital Forensics Tool Testing Images. https://dftt.sourceforge.net. (Accessed 13 May 2025).
Casey, E., 2002. Error, uncertainty and loss in digital evidence. Int. J. Digital Eviden. 1.
Casey, E., 2006. Cutting corners: trading justice for cost savings. Digit. Invest.: Int. J. Digital Forensics Incident Response 3, 185–186.
Casey, E., Barnum, S., Griffith, R., Snyder, J., van Beek, H., Nelson, A., 2017. Advancing coordinated cyber-investigations and tool interoperability using a community developed specification language. Digit. Invest. 22, 14–45.
Ceballos Delgado, A.A., Glisson, W.B., Grispos, G., Choo, K.K.R., 2022. Fade: a forensic image generator for Android device education. Wiley Interdisciplin. Rev.: Forensic Sci. 4, e1432.
Demmel, M., Göbel, T., Gonçalves, P., Baier, H., 2024. Data synthesis is going mobile – on community-driven dataset generation for Android devices. Digital Threats: Res. Pract. 5, 1–19.
Du, X., Hargreaves, C., Sheppard, J., Scanlon, M., 2021. TraceGen: user activity emulation for digital forensic test image generation. Forensic Sci. Int.: Digit. Invest. 38, 301133.
Garfinkel, S.L., 2010. Digital forensics research: the next 10 years. Digit. Invest. 7, 64–73.
Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. Digit. Invest. 6, 2–11.
Göbel, T., Maltan, S., Türr, J., Baier, H., Mann, F., 2022. ForTrace – a holistic forensic data set synthesis framework. Forensic Sci. Int.: Digit. Invest. 40, 301344.
Göbel, T., Breitinger, F., Baier, H., 2025. Optimising data set creation in the cybersecurity landscape with a special focus on digital forensics: principles, characteristics, and use cases. Forensic Sci. Int.: Digit. Invest. 52, 301882.
Gonçalves, P., Dološ, K., Stebner, M., Attenberger, A., Baier, H., 2022. Revisiting the dataset gap problem – on availability, assessment and perspective of mobile forensic corpora. Forensic Sci. Int.: Digit. Invest. 43, 301439.
Grajeda, C., Breitinger, F., Baggili, I., 2017. Availability of datasets for digital forensics – and what is missing. Digit. Invest. 22, 94–105.
Guo, Y., Slay, J., Beckett, J., 2009. Validation and verification of computer forensic software tools–searching function. Digit. Invest. 6, 12–22.
Guttman, B., Lyle, J.R., Ayers, R., 2011. Ten years of computer forensic tool testing. Digital Evidence & Elec. Signature L. Rev. 8, 139.
Hargreaves, C., Patterson, J., 2012. An automated timeline reconstruction approach for digital forensic investigations. Digit. Invest. 9, 69–79.
Hargreaves, C., Breitinger, F., Dowthwaite, L., Webb, H., Scanlon, M., 2024a. DFPulse: the 2024 digital forensic practitioner survey. Forensic Sci. Int.: Digit. Invest. 51, 301844.
Hargreaves, C., Nelson, A., Casey, E., 2024b. An abstract model for digital forensic analysis tools – a foundation for systematic error mitigation analysis. Forensic Sci. Int.: Digit. Invest. 48, 301679.
Hargreaves, C., van Beek, H., Casey, E., 2025. SOLVE-IT: a proposed digital forensic knowledge base inspired by MITRE ATT&CK. Forensic Sci. Int.: Digit. Invest. 52, 301864.
Horsman, G., 2018. "I couldn't find it your honour, it mustn't be there!" – tool errors, tool limitations and user error in digital forensics. Sci. Justice 58, 433–440.
Horsman, G., 2019. Tool testing and reliability issues in the field of digital forensics. Digit. Invest. 28, 163–175.
Horsman, G., Lyle, J.R., 2021. Dataset construction challenges for digital forensics. Forensic Sci. Int.: Digit. Invest. 38, 301264.
HuMoran, 2022. Android Asset Packaging Tool for Python3. https://github.com/HuMoran/aapt. (Accessed 20 May 2025).
Jaeckel, L., Spranger, M., Labudde, D., 2025. Forensic analysis of Telegram messenger on iOS smartphones. Forensic Sci. Int.: Digit. Invest. 52, 301866.
Lyle, J.R., Guttman, B., Butler, J.M., Sauerwein, K., Reed, C., Lloyd, C.E., 2022. Digital Investigation Techniques: a NIST Scientific Foundation Review. NIST Internal Report NIST IR 8354. National Institute of Standards and Technology, Gaithersburg, MD.
Marshall, A.M., 2021. Digital forensic tool verification: an evaluation of options for establishing trustworthiness. Forensic Sci. Int. Digit. Invest. 38, 301181.
Marshall, A.M., Paige, R., 2018. Requirements in digital forensics method definition: observations from a UK study. Digit. Invest. 27, 23–29.
Michel, M., Pawlaszczyk, D., Zimmermann, R., 2022. Autopod-mobile – semi-automated data population using case-like scenarios for training and validation in mobile forensics. Forensic Sci. 2, 302–320.
Michelet, G., Breitinger, F., Horsman, G., 2023. Automation for digital forensics: towards a definition for the community. Forensic Sci. Int. 349, 111769.
Moch, C., Freiling, F., 2009. The forensic image generator generator (Forensig2). In: International Conference on IT Security Incident Management and IT Forensics, IEEE, pp. 78–93.
Nemetz, S., Schmitt, S., Freiling, F., 2018. A standardized corpus for SQLite database forensics. Digit. Invest. 24, 121–130.
NIST, 2019. Computer Forensics Tool Testing Program (CFTT). https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt. (Accessed 17 May 2025).
OpenJS Foundation, 2025a. Appium Documentation. https://appium.io/docs/en/2.0/. (Accessed 13 May 2025).
OpenJS Foundation, 2025b. Introduction to Appium. https://appium.io/docs/en/2.0/intro/. (Accessed 17 May 2025).
Rzepka, L., Ottmann, J., Stoykova, R., Freiling, F., Baier, H., 2025. A scenario-based quality assessment of memory acquisition tools and its investigative implications. Forensic Sci. Int. Digit. Invest. 52, 301868.
Scanlon, M., Du, X., Lillis, D., 2017. EviPlant: an efficient digital forensic challenge creation, manipulation and distribution solution. Digit. Invest. 20, 29–36.
Scanlon, M., Breitinger, F., Hargreaves, C., Hilgert, J.N., Sheppard, J., 2023. ChatGPT for digital forensic investigation: the good, the bad, and the unknown. Forensic Sci. Int.: Digit. Invest. 46, 301609.
Schmidt, L., Kortmann, S., Hupperich, T., 2023. Improving trace synthesis by utilizing computer vision for user action emulation. Forensic Sci. Int.: Digit. Invest. 45, 301557.
Scientific Working Group on Digital Evidence, 2018. Minimum Requirements for Testing Tools Used in Digital and Multimedia Forensics. https://www.nist.gov/system/files/documents/2023/08/11/SWGDE. (Accessed 13 May 2025).
Spichiger, H., Adelstein, F., 2025. Preserving meaning of evidence from evolving systems. Forensic Sci. Int.: Digit. Invest. 52, 301867.
Statista, 2025. Frequency of Mobile App Updates on the Google Play Store as of April 2025. https://www.statista.com/statistics/1404434/google-play-store-app-updates-by-frequency/. (Accessed 11 May 2025).
UK Forensic Science Regulator, 2020. Forensic Science Regulator Guidance: Method Validation in Digital Forensics. FSR-G-2018 Issue 2. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/921392/218_Method_Validation_in_Digital_Forensics_Issue_2_New_Base_Final.pdf. (Accessed 13 May 2025).
van Baar, R.B., van Beek, H.M., Van Eijk, E., 2014. Digital forensics as a service: a game changer. Digit. Invest. 11, 54–62.
van Beek, H.M., Van Eijk, E., van Baar, R.B., Ugen, M., Bodde, J., Siemelink, A., 2015. Digital forensics as a service: game on. Digit. Invest. 15, 20–38.
van Beek, H.M., van den Bos, J., Boztas, A., Van Eijk, E., Schramp, R., Ugen, M., 2020. Digital forensics as a service: stepping up the game. Forensic Sci. Int.: Digit. Invest. 35, 301021.
Visti, H., Tohill, S., Douglas, P., 2015. Automatic creation of computer forensic test images. In: International Workshop on Computational Forensics. Springer, pp. 163–175.
Voigt, L.L., Freiling, F., Hargreaves, C., 2024. Re-imagen: generating coherent background activity in synthetic scenario-based forensic datasets using large language models. Forensic Sci. Int.: Digit. Invest. 50, 301805.
Voigt, L.L., Freiling, F., Hargreaves, C., 2025. A metrics-based look at disk images: insights and applications. Forensic Sci. Int.: Digit. Invest. 52, 301874.
Webb, H., Fitzroy-Dale, N., Aqeel, S., Piskopani, A.M., Stafford-Fraser, Q., Nikolaou, C., Dowthwaite, L., Mcauley, D., Hargreaves, C., 2024. Responsible ai in policing. In: Proceedings of the Second International Symposium on Trustworthy Autonomous Systems, pp. 1–5.
Wickramasekara, A., Densmore, A., Breitinger, F., Studiawan, H., Scanlon, M., 2025. AutoDFBench: a framework for AI generated digital forensic code and tool testing and evaluation. In: Proceedings of the Digital Forensics Doctoral Symposium, pp. 1–7.
Wolf, D., Göbel, T., Baier, H., 2024. Hypervisor-based data synthesis: on its potential to tackle the curse of client-side agent remnants in forensic image generation. Forensic Sci. Int.: Digit. Invest. 48, 301690.
World Wide Web Consortium (W3C), 2025. XML Path Language (XPath) 3.1. https://www.w3.org/TR/xpath-31/. (Accessed 14 May 2025).
Wundram, M., Freiling, F., Moch, C., 2013. Anti-forensics: the next step in digital forensics tool testing. In: International Conference on IT Security Incident Management and IT Forensics. IEEE, pp. 83–97.
Yannikos, Y., Winter, C., 2013. Model-based generation of synthetic disk images for digital forensic tool testing. In: International Conference on Availability, Reliability and Security. IEEE, pp. 498–505.
Yannikos, Y., Franke, F., Winter, C., Schneider, M., 2011. 3LSPG: forensic tool evaluation by three layer stochastic process-based generation of data. In: Intl. Workshop on Computational Forensics. Springer, pp. 200–211.
Yannikos, Y., Graner, L., Steinebach, M., Winter, C., 2014. Data corpora for digital forensics education and research. In: IFIP International Conference on Digital Forensics. Springer, pp. 309–325.