Journal of Internet Services and Applications, 2025, 16:1, doi: 10.5753/jisa.2025.5242

INTACT: Compact Storage of Data Streams in Mobile Devices to Unlock User Privacy at the Edge

Rémy Raes [**Inria, Univ. Lille, CNRS, UMR 9189 CRIStAL, France** | ***remy.raes@inria.fr***]
Olivier Ruas [**Pathway, France** | ***olivier.ruas@gmail.com***]
Adrien Luxey-Bitri [**Inria, Univ. Lille, CNRS, UMR 9189 CRIStAL, France** | ***adrien.luxey@inria.fr***]
Romain Rouvoy [**Inria, Univ. Lille, CNRS, UMR 9189 CRIStAL, France**| ***romain.rouvoy@inria.fr***]

Centre Inria de l'Université de Lille, Parc scientifique de la Haute-Borne, 40, avenue Halley - Bât A - Park Plaza, 59650 Villeneuve d'Ascq - France

Received: 29 November 2024 • **Accepted:** 10 April 2025 • **Published:** 30 June 2025

Abstract Data streams produced by mobile devices, such as smartphones, offer highly valuable sources of information to build ubiquitous services. Such data streams are generally uploaded and centralized to be processed by third parties, potentially exposing sensitive personal information. In this context, existing protection mechanisms, such as *Location Privacy Protection Mechanisms* (LPPMs), have been investigated. Alas, none of them have actually been implemented, nor deployed in real-life, in mobile devices to enforce user privacy at the edge. Moreover, the diversity of embedded sensors and the resulting data deluge makes it impractical to provision such services directly on mobiles, due to their constrained storage capacity, communication bandwidth and processing power. This article reports on the FLI technique, which leverages a piece-wise linear approximation technique to capture compact representations of data streams in mobile devices. Beyond the FLI storage layer, we introduce *Divide & Stay*, a new privacy preservation technique to execute *Points of Interest* (POIs) inference. Finally, we deploy both of them on Android and iOS as the INTACT framework, making a concrete step towards enforcing privacy and trust in ubiquitous computing systems.

Keywords: Mobile, data streams, storage, compression, geolocation, privacy, attack

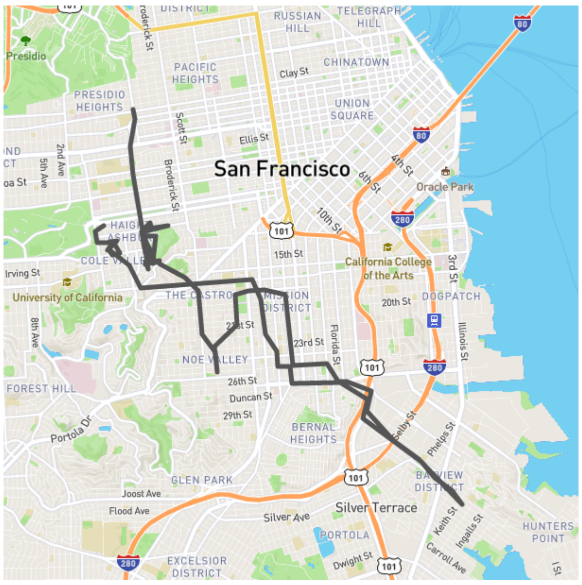
1 Introduction

Mobile devices: usage and data With the advent of smartphones and more generally the *Internet of Things* (IoT), ubiquitous devices are mainstream in our societies and widely deployed at the edge of networks. Such constrained devices are not only consuming data and services, such as content streaming, restaurant recommendations or more generally *Location-Based Service*s (LBSs), but are also key producers of data streams by leveraging a wide variety of embedded sensors that capture the surrounding environment of end-users, including their daily routines. The data deluge generated by a connected user is potentially tremendous: according to preliminary experiments, a smartphone can generate approximately 2 pairs of *Global Positioning System* (GPS) samples and 476 triplets of accelerometer samples per second, resulting in more than 172,800 location and 41,126,400 acceleration samples daily.

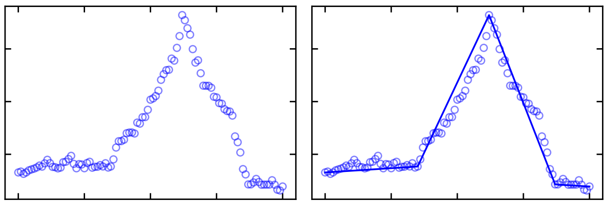
In this context, the storage and processing of such data streams in mobile devices are challenges that cannot only be addressed by assuming that the hardware capabilities will keep increasing. In particular, sustainability issues call for increasing the lifespan of legacy devices, thus postponing their replacement. This implies that software-defined solutions are required to leverage the shortenings of hardware resources.

Privacy Continuous data streams inevitably include *Sensitive Personal Information* (SPI) that jeopardize the privacy of end-users, when processed by malicious stakeholders. While machine learning algorithms are nowadays widely adopted as a convenient keystone to process large datasets and infer actionable insights, they often require grouping raw input datasets in a remote place, thus imposing a privacy threat for end-users sharing their data. This highlights the utility vs. privacy trade-off that is inherent to any data-sharing activity [Cerf *et al.*, 2017]. On the one hand, without crowdsourced GPS traces, it would be hard to model traffic in real-time and recommend itineraries. On the other hand, it is crucial to protect user privacy when accepting to gather SPI.

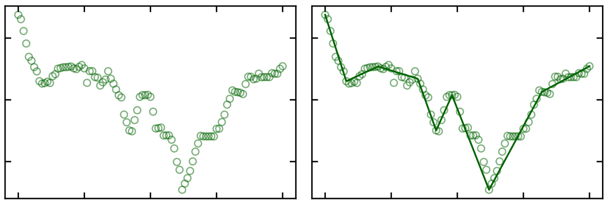
LPPM To address this ethical challenge, privacy-preserving machine learning [Xu *et al.*, 2015] and decentralized machine learning [Bellet *et al.*, 2017; y Arcas, 2018] are revisiting state-of-the-art machine learning algorithms to enforce user privacy, among other properties. Regarding location privacy, several *Location Privacy Protection Mechanisms* (LPPMs) have been developed to preserve user privacy in mobility situations. Location reports are evaluated and obfuscated before being sent to a service provider, hence keeping user data privacy under control. The user no longer automatically shares their data streams with service providers but carefully selects what they share and make sure the data they unveil does not contain any SPI. For example, Geo-Indistinguishability [Andrés *et al.*, 2013] generalizes differential privacy [Dwork, 2008] to GPS traces, while Promesse [Primault *et al.*, 2015] smooths the GPS traces—both temporally and geographically—to erase



(a) Cabspotting mobility sub-trace of `user 0`.



(b) Raw longitude trace for `user 0`. **(c)** Modeled longitude with FLI.



(d) Raw latitude trace for `user 0`. **(e)** Modeled latitude with FLI.

Figure 1. FLI compacts any location stream as a sequence of segments.

POIs from the input trace. LPPMs successfully preserve sensitive data, such as POIs, while maintaining the data utility for the targeted service.

Challenges Despite their reported effectiveness, no LPPM has ever been implemented and deployed on mobile devices: previous works have been simulated on the *Android Debug Bridge* (ADB) [Khalfoun *et al.*, 2021] at best. While the extension of those works to Android and iOS devices may seem straightforward, it is hindered by the scarce resources of edge devices. Storage is notably challenging, as LPPMs generally require the user to access all their GPS traces, and sometimes the ones of additional users. To this day, storing such amounts of data is challenging on constrained mobile devices. Data processing algorithms are additionally not optimized for mobile devices. Furthermore, even if the storage capacity of modern devices keeps increasing, the deluge of data streams produced by all the sensors of a smartphone (e.g., GPS, accelerometer, gyroscope, etc.) makes it impossible to store all the raw data for the applications that require it.

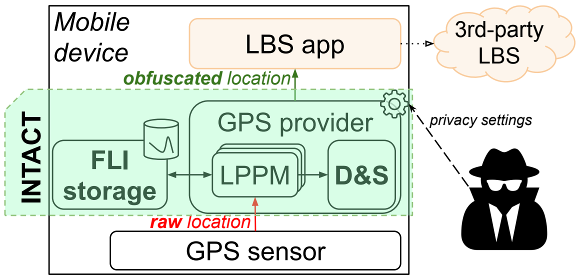


Figure 2. Overview of INTACT contributions.

Content This work, as an extension of the method presented in [Raes *et al.*, 2024], demonstrates that modeling data streams successfully addresses these device-level storage & processing challenges.

In previous work from our team, a novel algorithm, *Fast Linear Interpolation* (FLI), was introduced to model and store data streams under memory constraints, leveraging a *Piece-wise Linear Approximation* (PLA) technique [Raes *et al.*, 2024]. FLI does not store raw data samples (Fig. 1b & 1d) but, instead, models their evolution as linear interpolations (Fig. 1c & 1e)—offering a much bigger storage capacity at the cost of a controlled approximation error.

In this paper, we demonstrate how FLI can be leveraged to implement an LPPM working directly on mobile phones—thanks to the increased GPS storage capacity offered by FLI. To be proven useful, the LPPM's privacy gains must be evaluated *in situ*, before any geolocation trace is shared. We thus also introduce a new POI attack algorithm, dubbed *Divide & Stay* (D&S), running directly on mobile, which goal is to extract POIs from GPS traces. We will use D&S to assert whether the LPPMs effectively protects the privacy of traces modeled with FLI.

Altogether, FLI and D&S constitute the *IN-siTu locAtion proteCTion* (INTACT) framework, pictured in Fig. 2. The figure shows how our two contributions can be combined to enable the deployment of LPPMs on mobile devices to support location-preserving LBS. We report that INTACT can store vast amounts of location data on mobile, and that it can protect end-user privacy while using LBS by using device-local LPPMs.

In the following, we first discuss related works (Sec. 2), before dissecting the INTACT framework (Sec. 3). Next, we present our experimental setup (Sec. 4) and the results we obtained (Sec. 5). Finally, we discuss the limitations of our approach (Sec. 6) and then conclude this paper (Sec. 8).

2 Related Work

2.1 Data Storage

Overcoming the memory constraints of mobile devices to store data streams usually implies the integration of efficient temporal databases. To take the example of Android: few databases are available, such as SQLite and its derivative Drift [Binder, 2019], the cloud-supported Firebase [Tamplin and Lee, 2012], the NoSQL hive, and ObjectBox [Dollinger and Junginger, 2014]. The situation is similar on iOS.

Relational databases (*e.g.*, SQL) are typically designed for *OnLine Transactional Processing* (OLTP) and *OnLine*

Analytical Processing (OLAP) workloads, which widely differ from time-series workloads. In the latter, reads are mostly contiguous (as opposed to the random-read tendency of OLTP); writes are most often inserts (not updates) and typically target the most recent time ranges. OLAP is designed to store big data workloads to compute analytical statistics, while not putting the emphasis on read or write performances. Finally, in temporal workloads, it is unlikely to process writes & reads in the same single transaction [Timescale, 2019].

Time series databases (TSDB). Despite these deep differences, several relational databases offer support for temporal data with industry-ready performance—*e.g.*, TimescaleDB [Timescale Inc, 2018] is a middleware that exposes temporal functionalities atop a relational PostgreSQL foundation. InfluxDB [InfluxData, 2013] is one of the most widely used temporal databases. Unfortunately, when facing memory constraints, its retention policy prevents the storage from scaling in time: the oldest samples are dumped to make room for the new ones. Furthermore, on mobile, memory shortages often cause the operating system to kill the TSDB process to free the memory, which is opposed to the very concept of in-memory databases.

Moving objects databases (MOD). Location data storage is an issue that has also been studied in the MOD community, where a central authority merges trajectory data from several sensors in real-time. To optimize storage and communication costs, it does not store the raw location data, but rather trajectory approximations. *Linear Dead Reckoning* (LDR) [Wolfson *et al.*, 1998] limits data exchange between sensors and server by sending new location samples only when a predefined *accuracy bound* ϵ (in meters) is exceeded. A mobility prediction vector is additionally shared every time a location sample is sent. Even so, this class of solutions requires temporarily storing modeled locations to ensure they fit the ϵ bound and exclusively focuses on modeling location data streams, while we aim at storing any type of real-valued stream.

Modeling data streams. While being discrete, the streams sampled by sensors represent inherently continuous signals. Data modeling does not only allow important memory consumption gains, but also flattens sensors' noise, and enables extrapolation between measurements. In particular, *Piece-wise Linear Approximation* (PLA) is used to model the data as successive affine functions. An intuitive way to do linear approximation is to apply a bottom-up segmentation: each pair of consecutive points is connected by interpolations; the less significant contiguous interpolations are merged, as long as the obtained interpolations introduce no error above a given threshold. The bottom-up approach has low complexity, but usually requires an offline approach to consider all the points at once. The *Sliding Window And Bottom-up* (SWAB) algorithm [Keogh *et al.*, 2001], however, is an online approach that uses a sliding window to buffer the latest samples on which a bottom-up approach is applied. emSWAB [Berlin and Van Laerhoven, 2010] improves the sliding window by adding several samples at the same time instead of one. Instead of interpolation, linear regression can also be used to model the samples reported by IoT sensors [Grützmacher *et al.*, 2018]. For example, Greycat [Moawad *et al.*, 2015] adopts polynomial regressions with higher degrees to further compress the data. Unfortunately, none of those works have been implemented on mobile devices to date.

Sprintz [Blalock *et al.*, 2018] proposes a mobile lossless compression scheme for multi-modal integer data streams, along with a comparison of other compression algorithms. They target streaming of the compressed data to a centralized location from IoT devices with minimal resources. This work is orthogonal to ours, as FLI intends to model floating-point unimodal streams on one's devices for further local computation, instead of streaming it to a third-party server.

Closer to our work, FSW [Liu *et al.*, 2008] and the ShrinkingCone algorithm [Galakatos *et al.*, 2019] attempt to maximize the length of a segment while satisfying a given error threshold, using the same property used in FLI. FSW is not a streaming algorithm as it considers the dataset as a whole, and does not support insertion. The ShrinkingCone algorithm is a streaming greedy algorithm designed to approximate an index, mapping keys to positions: it only considers monotonic increasing functions and can produce disjoints segments. FLI models non-monotonic functions in a streaming fashion, while providing joint segments.

Limitations. To the best of our knowledge, state-of-the-art storage solutions for unbounded data streams either require storing raw data samples or triggering *a posteriori* data computations, which makes them unsuitable for mobile devices.

2.2 Location Privacy Attacks

Raw user mobility traces can be exploited to model the users' behavior and reveal their *Sensitive Personal Information* (SPI). In particular, the POIs are widely used as a way to extract SPI from mobility traces. In a nutshell, a POI is a place where the user comes often and stays for a significant amount of time: it can reveal their home, workplace, or leisure habits. From revealed POIs, more subtle information can also be inferred: sexual orientation from attendance to LGBT+ places, for instance. The set of POIs can also be used as a way to re-identify a user in a dataset of mobility traces [Primault *et al.*, 2014; Gambs *et al.*, 2014]. The POIs can be extracted using spatiotemporal clustering algorithms [Zhou *et al.*, 2004; Hariharan and Toyama, 2004]. Alternatively, an attacker may also re-identify a user directly from raw traces, without computing any POI [Maouche *et al.*, 2017].

2.3 Protecting Mobility Datasets

When data samples are gathered in a remote server, one can expect the latter to protect the dataset as a whole. In particular, *k-anonymity* [Sweeney, 2002] is the property of a dataset guaranteeing that whenever some data leaks, the owner of each data trace is indistinguishable from at least $k - 1$ other users contributing to the dataset. Similarly, *l-diversity* [Machanavajjhala *et al.*, 2007] extends *k-anonymity* by ensuring that the l users are diverse enough not to infer SPI about the data owner. Finally, *differential privacy* [Dwork, 2008] aims at ensuring that the inclusion of a single element in a dataset does not alter significantly an aggregated query on the whole dataset. However, all these techniques require personal samples to be grouped to enforce user privacy.

2.4 Protecting Individual Traces

Rather than protecting the dataset as a whole, each data sample can also be protected individually. In the case of location data, several protection mechanisms—called *Location Privacy Protection Mechanisms* (LPPMs)—have been developed. They may be deployed in a remote server where all data samples are gathered, or run directly on the source device before any data exchange.

***Geo-Indistinguishability* (GeoI) [Andrés *et al.*, 2013]** implements differential privacy [Dwork, 2008] at the trace granularity. In particular, GeoI adjusts mobility traces with two-dimensional Laplacian noise, making POIs more difficult to infer. *Heat Map Confusion* (HMC) [Maouche *et al.*, 2018] aims at preventing re-identification attacks by altering all the traces altogether. The raw traces are transformed into heat maps, which are altered to look like another heat map in the dataset, and then transformed back to a GPS trace.

Promesse [Primault *et al.*, 2015] smooths the mobility traces, both temporally and geographically, to erase POIs from the trace. Promesse ensures that, between each location sample, there is at least a given time and distance interval. In the resulting mobility trace, the user appears to have a constant speed. While Promesse blurs the time notion from the trace—*i.e.*, the user never appears to stay at the same place—it does not alter their spatial characteristics. Yet, while POIs may be still inferred if the user repeatedly goes to the same places, it will be harder to distinguish such POIs from more random crossing points.

It is also possible to combine several LPPMs to improve the privacy of users [Meftah *et al.*, 2019; Khalfoun *et al.*, 2021]. Because of potential remote leaks, the user should anonymize their trace locally before sharing it, which is how Eden [Khalfoun *et al.*, 2021] operates. However, Eden has not been deployed: it has only been simulated on ADB. Even more so: despite their validity and to the best of our knowledge, no LPPM has been implemented in mobile devices. This is partly due to the tight constraints of mobile devices, memory-wise notably: HMC [Maouche *et al.*, 2018], for instance, requires locally loading a large set of GPS traces to operate.

3 The INTACT framework

In order to present the *IN-siTu locAtion proteCTion* (INTACT) framework, our two proposals will be presented in order: first, our time series modelisation tool *Fast Linear Interpolation* (FLI), and then, the POI attack algorithm *Divide & Stay* (D&S).

3.1 FLI: Online Time Series Modeling

To overcome the memory constraints of mobile devices, we claim that efficient temporal storage solutions must be ported onto ubiquitous environments. In particular, we advocate the use of data modeling, such as *Piece-wise Linear Approximation* (PLA) [Keogh *et al.*, 2001; Grützmacher *et al.*, 2018] or Greycat [Moawad *et al.*, 2015], to increase the storage capacity of mobile devices. Therefore, we introduce FLI, an online time series modeling algorithm based on an iterative and continuous PLA to store approximate models of data streams on memory-constrained devices, instead of storing all the raw data samples as state-of-the-art temporal databases do.

The intuition behind FLI is that time series generally do not vary abruptly but rather follow linear tendencies, whether their value increases, decreases or remains the same over time. Linear segments being suitable to represent linear tendencies, one can imagine representing a given time series as interconnected segments, of which each segment models as much original data points as possible, while keeping an error below a predefined threshold. FLI does just that: given a configuration parameter $\epsilon \in \mathbb{R}^{+*}$, it models any univariate time series as a series of segments (or interpolations), dropping original data points (or samples) as long as the following invariant is preserved:

All samples modeled by an interpolation maintain an error below the maximum threshold ϵ.

FLI is an online algorithm: each of the original time series' data points are inserted in order. During this process, FLI decides whether the latest sample can be modeled with the latest interpolation, or if a new segment needs to be created. We will first present FLI's representation of the time series, then how reads are performed, before diving into the insertion algorithm.

FLI's mathematical formulation Consider an ever-growing univariate real-valued time series $\mathcal{P} \subset \mathbb{R}^2$, such that its i-th point is expressed as: $\mathcal{P}[i] = (t_i, x_i)$, with $i \in \mathbb{N}^*$, where $t_i \in \mathbb{R}^+$ is the point's timestamp, and $x_i \in \mathbb{R}$ is its value. Let $\mathcal{F}$ be the data structure containing FLI's representation of $\mathcal{P}$ under an error bound of ϵ. We say that $\mathcal{F}$ *models* $\mathcal{P}$ under ϵ, or: $\mathcal{F} \models_\epsilon \mathcal{P}$.

The data structure $\mathcal{F}$ is composed of *i)* a series of historical points $\mathcal{H}$ selected from $\mathcal{P}$, *ii)* a triplet of gradients $(g_M, g_{\text{min}}, g_{\text{max}})$ that are used for reading and insertion, and *iii)* the last inserted point p_{last}. How these elements are used is displayed in Fig. 3.

$$\mathcal{F} = (\mathcal{H}, g_M, g_{\text{min}}, g_{\text{max}}, p_{\text{last}}) \text{ s. t.}$$
$$\begin{cases} \mathcal{H} \subset \mathcal{P} \subset \mathbb{R}^2 \\ (g_M, g_{\text{min}}, g_{\text{max}}) \in \mathbb{R}^3 \\ p_{\text{last}} \in \mathbb{R}^2 \end{cases}$$

Any *segment* $\vec{s}$ in $\mathcal{H}$ *models* one or more points in the original time series $\mathcal{P}$:

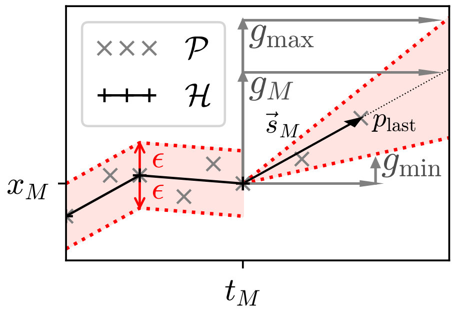


Figure 3. A time series $\mathcal{P}$ being modeled with FLI, displaying every component of FLI's data structure $\mathcal{F} = (\mathcal{H}, g_M, g_{\min}, g_{\max}, p_{\text{last}})$. $\mathcal{H}$ models every point in $\mathcal{P}$ while keeping their error below ϵ. $\mathcal{H}$'s last entry, $\mathcal{H}[M] = (t_M, x_M)$, is the origin of the latest model $\vec{s}_M$, with slope g_M. $\vec{s}_M$ can notably be used to read past $\mathcal{H}[M]$, into the future. Gradients $g_{\min}$ and $g_{\max}$ bound the next sample insertions so as to keep their modeling error below ϵ. The three slopes are updated upon insert. The last inserted point p_{last} is also saved for next iterations.

$$\begin{cases} \forall k \in [\![1, |\mathcal{H}|[\![\\ \exists (i,j) \in [\![1, |\mathcal{P}|]\!]^2 \\ i < j \end{cases} \text{s. t.} \begin{cases} \mathcal{H}[k] = \mathcal{P}[i] \\ \mathcal{H}[k+1] = \mathcal{P}[j] \\ \vec{s}_k = \langle \mathcal{H}[k], \mathcal{H}[k+1] \rangle \end{cases} \text{then}$$

$$\forall l \in [\![i, j[\![, \vec{s}_k \models_\epsilon \mathcal{P}[l] \iff \text{dist}(\vec{s}_k, \mathcal{P}[l]) < \epsilon \quad (1)$$

The latest (or *current*) segment $\vec{s}_M$ is treated differently: as it starts from the last point in $\mathcal{H}$, FLI stores its slope g_M in $\mathcal{F}$:

$$\begin{cases} M = |\mathcal{H}| \\ \exists i \in [\![1, |\mathcal{P}|]\!] \end{cases} \text{s. t.} \begin{cases} \mathcal{H}[M] = \mathcal{P}[i] \\ \vec{s}_M = \langle \mathcal{H}[M], g_M \rangle \end{cases} \text{then}$$

$$\forall l \in [\![i, |\mathcal{P}|]\!], \vec{s}_M \models_\epsilon \mathcal{P}[l] \iff \text{dist}(\vec{s}_M, \mathcal{P}[l]) < \epsilon \quad (2)$$

Reading data streams In FLI, reading a value at time t is achieved by estimating its image using the appropriate interpolation, as detailed in Algorithm 1.

If t is lower than the last timestamp stored in the history (t_M for short), then a position on an interpolation stored in $\mathcal{H}$ will be retrieved. Line 3 selects the appropriate index k such that $\vec{s}_k = \langle \mathcal{H}[k], \mathcal{H}[k+1] \rangle$ contains the queried timestamp: $\mathcal{H}[k].t \leq t < \mathcal{H}[k+1]$. In practice, the lookup is done with a dichotomy search. Lines 4-5 return the image of t on the segment $\vec{s}_k$, after having computed the gradient g between the consecutive historical points $\mathcal{H}[k]$ and $\mathcal{H}[k+1]$.

If t is ulterior or equal to t_M, line 7 returns the image on the current interpolation $\vec{s}_M = \langle \mathcal{H}[M], g_M \rangle$.

Inserting data samples Let us now go through the most important algorithm in FLI, the sequential insertion, which is in charge of maintaining the invariant: *all modeled points in $\mathcal{P}$ have an approximation error below ϵ.* Data samples are inserted sequentially: the current interpolation is adjusted to fit new samples until it cannot satisfy the invariant. A naive solution to maintain the invariant while updating the current model would be to memorize every sample between $\mathcal{H}[M]$ and the last observed sample, to check their error against the model—which would be costly. Instead, FLI only maintains $g_{\min}$ and $g_{\max}$, which are cost-effectively updated at each sample insertion.

Algorithm 1 Approximate read using FLI

```
Require: { $\mathcal{F} = (\mathcal{H}, g_M, g_{\min}, g_{\max}, p_{\text{last}})$
         { $M = |\mathcal{H}|$
         { $(t_M, x_M) = \mathcal{H}[M]$
1: function Read($t \in \mathbb{R}$)
2:     if $t \leq t_M$ then                                   ▷ Historical read
3:         find $k \in [\![1, M[\![$ s. t. { $\mathcal{H}[k].t \geq t$
                                           { $\mathcal{H}[k+1].t < t$
4:         $g \leftarrow \frac{\mathcal{H}[k+1].x - \mathcal{H}[k].x}{\mathcal{H}[k+1].t - \mathcal{H}[k].t}$      ▷ Compute img. on $\vec{s}_k$
5:         return $g \times (t - \mathcal{H}[k].t) + \mathcal{H}[k].x$
6:     else                                                   ▷ Forward read
7:         return $g_M \times (t - t_M) + x_M$
8:     end if
9: end function
```

Algorithm 2 Insertion using $\epsilon \in \mathbb{R}^{+*}$

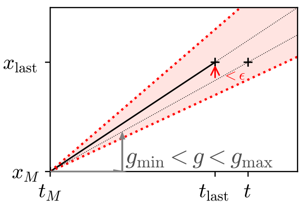
```
Require: { $\mathcal{F} = (\mathcal{H}, g_M, g_{\min}, g_{\max}, p_{\text{last}})$
         { $M = |\mathcal{H}|$
         { $(t_M, x_M) = \mathcal{H}[M]$
1: function insert($p = (t, x) \in \mathbb{R}^2$, $\epsilon$)
2:     $(t_\Delta, x_\Delta) \leftarrow (t - t_M, x - x_M)$
3:     $g \leftarrow x_\Delta / t_\Delta$                                   ▷ Compute $g$
4:     if $g_{\min} < g < g_{\max}$ then
5:         $g_M \leftarrow g$                                   ▷ Update model (Fig. 4)
6:         $(g^p_{\min}, g^p_{\max}) \leftarrow \left(\frac{x_\Delta - \epsilon}{t_\Delta}, \frac{x_\Delta + \epsilon}{t_\Delta}\right)$
7:         $g_{\min} \leftarrow \max(g_{\min}, g^p_{\min})$
8:         $g_{\max} \leftarrow \min(g_{\max}, g^p_{\max})$
9:     else
10:        $\mathcal{H} \leftarrow \mathcal{H} \cup [(t_{\text{last}}, x_{\text{last}})]$        ▷ New model (Fig. 5)
11:        $(t'_\Delta, x'_\Delta) \leftarrow (t - t_{\text{last}}, x - x_{\text{last}})$
12:        $g_M \leftarrow x'_\Delta / t'_\Delta$
13:        $g_{\min} \leftarrow (x'_\Delta - \epsilon) / t'_\Delta$
14:        $g_{\max} \leftarrow (x'_\Delta + \epsilon) / t'_\Delta$
15:    end if
16:    $(t_{\text{last}}, x_{\text{last}}) \leftarrow (t, x)$                 ▷ Update last sample
17: end function
```

Notation-wise, we will consider than the original series $\mathcal{P}$ is ever-growing, and that at each insertion, FLI handles sample $\mathcal{P}[|\mathcal{P}|]$, noted $p = (t, x)$. The last point inserted in FLI's datastructure $\mathcal{F}$ is noted $p_{\text{last}} = (t_{\text{last}}, x_{\text{last}})$. Before insertion, p and p_{last} coexist. After insertion, p_{last} takes the value of p.

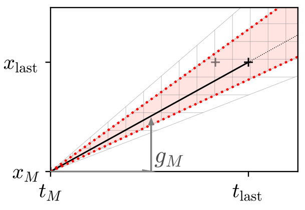
Following along Algorithm 2, we will first cover the insertion of samples that fall within the current interpolation (represented in Fig. 4), before explaining how a new segment is generated once the last point breaks the invariant (displayed in Fig. 5).

Upon insertion of a new sample p, lines 2-3 first compute the slope g of the segment $\vec{s} = \langle \mathcal{H}[M], p \rangle$. On line 4, g is compared to the interval $]g_{\min}, g_{\max}[$, to know whether p falls within the bounds of the current interpolation $\vec{s}_M$ or not.

If it falls within (cf. Fig. 4), p is added to the current interpolation: g_M is updated to g on line 5, g_M being used for reading ahead (see Alg. 1). Two new slopes $g^p_{\min}$ and $g^p_{\max}$ are computed on line 6. They respectively represent the lines $\langle \mathcal{H}[M], (t, x - \epsilon) \rangle$ and $\langle \mathcal{H}[M], (t, x + \epsilon) \rangle$. Together,

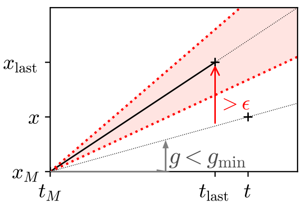


(a) p's gradient g remains within $[g_{\min}, g_{\max}]$.

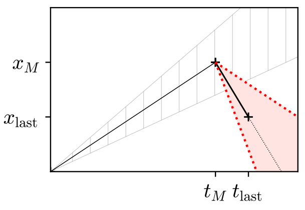


(b) The model is updated to encompass p.

Figure 4. When a new sample fits within $[g_{\min}, g_{\max}]$, it is added to the current model by updating $g_{\min}$ and $g_{\max}$. g_M is also updated for read queries (see Alg. 1).



(a) The error of p exceeds ϵ.



(b) A new model is created from the last two points.

Figure 5. When an error $> \epsilon$ is reported, a new model is created using p_{last} as its origin.

those two lines materialise p's *allowed cone* for future values, so as to keep $\mathcal{H}[M]$ as the origin of the model while preserving the invariant for p and the future insert. On line 7, $g_{\min}$ takes the maximum between its previous value and $g^p_{\min}$; on line 8, the converse operation happens to $g_{\max}$. What these two lines do is *shrinking* the interval $]g_{\min}, g_{\max}[$ so as to encompass p's allowed cone in the next iteration. Recursively, $g_{\min}$ and $g_{\max}$ represent the current model's allowed cone, which is *the intersection of all modeled points' allowed cones*—similarly to Galakatos *et al.* [2019], but in an online fashion. The process is pictured in Fig. 4b: vertical hatches represent the previous point's allowed cone, horizontal ones show the last point's, while the model's allowed cone is represented in red.

If g falls outside $]g_{\min}, g_{\max}[$ (that is: if p falls outside the model's allowed cone), then a new interpolation must begin from the two last points (see Fig. 5). It begins from the last two observed samples: the new $\vec{s}_M$ becomes $\langle p_{\text{last}}, p \rangle$. The penultimate sample p_{last} is thus persisted to the series of selected points $\mathcal{H}$ at line 10, and new values for $g_M, g_{\min}$ and $g_{\max}$ are derived based on the new $\vec{s}_M$ (lines 11–14). Recognize here how the computation of the new gradients is equivalent to the computation of p's allowed cone on line 6 (with a different origin).

In any case, the penultimate sample p_{last} is updated on line 16.

The case of ϵ The value of the error bound parameter has an important impact on the performances of FLI. If ϵ is too small, none of the inserted samples fit the current model at that time, thus initiating a new model each time. In that case, there will be one model per sample, imposing an important memory overhead. The resulting model overfits the data. On the other hand, if ϵ is too large, then all the inserted samples fit, and a single model is kept. While it is the best case memory-wise, the resulting model simply connects the first and last points and underfits the data.

The choice of ϵ is highly dependent on the input data. For instance, accelerometer data—that generally has a high sampling rate and little variance—are modeled efficiently even with small values of ϵ. Whereas GPS data—slow sampling rate and high variance—require finer tuning. We propose a parameter tuning process in Section 4.2.

On the other hand, the cautious reader will have observed how ϵ is only a parameter of the Insert function in Alg. 2. In future works, our team intends to dynamically adapt this error bound depending on the data and its freshness.

3.2 D&S: Attacking Location Privacy

Time series modeling being out of the way, we will now present our mobile-ready geolocation privacy attack tool: *Divide & Stay* (D&S). The purpose of such a tool is to audit the privacy of mobility traces directly where they are generated: on resource-constrained devices such as smartphones. Only then would it be reasonable privacy-wise to upload one's location traces to a third-party. D&S constitutes an enhancement of the original POI-Attack algorithm proposed by Primault *et al.* [2014]. The original contribution being too costly to run on mobile, D&S proposes a mobile-ready variation. Just like FLI, D&S fits into the broader INTACT framework, the mobile privacy system that constitutes our whole contribution.

In POI-Attack, the POI disclosure is done by a two-steps algorithm: potential candidates for POIs (dubbed *stays*) are first extracted, then *stays* are merged to avoid duplication of similar POIs. A *stay* is defined as a circle with a radius lower than d_{max} where a user spent a time higher than a set time t_{min}. A *stay* is represented by its center. The two thresholds t_{min} and d_{max} have an important impact on the type of POI extracted. *Short stays* will identify day-to-day patterns, such as shopping preferences, while *long stays* will identify *e.g.* travel preferences. In the second step, the *stays* whose centroids are close enough are merged to obtain the final list of POIs. POI-Attack [Primault *et al.*, 2014] iterates linearly over the mobility trace and compute *stays* as they appear. This approach is expensive for denser mobility traces—*i.e.*, with high frequency sampling. It is prohibitively long to execute on constrained devices like mobile phones.

Our contribution *Divide & Stay* (D&S) instead proposes a divide-and-conquer strategy that scales with the data density. The intuition behind D&S is to avoid wasting time looking for *stays* in portions of the trace where they are impossible, *i.e.* where more than d_{max} has been traveled in less than t_{min}, *e.g.* a car trip at high speed in a straight line. While the regular approach would consider each location until the end of the trace, D&S skips it entirely. The key idea of *Divide & Stay* is to recursively divide the trace until either such a *stay*-less segment is found and discarded, or until a fixed size segment is found on which the regular way to extract *stays* is performed.

Algorithm 3 depicts the pseudo-code of D&S. It scrutinizes the GPS trace $T \in (\mathbb{R} \times \mathbb{G})^n$, composed of n samples. For each $i \in [\![0, n-1]\!]$, $T[i].t$ represents the i_{th} sample's timestamp, while $T[i].g$ is its position in whatever geographical space $\mathbb{G}$ equipped with a distance function $\text{dist}_{\mathbb{G}}$. D&S has three configuration parameters: the aforementioned t_{min} and d_{max} representing the time and space limits of a POI, and

Algorithm 3 *Divide & Stay* (D&S) using parameters $(t_{min}, d_{max}, s_{max}) \in \mathbb{R}^{3+}$

```
Require: $T \in (\mathbb{R} \times \mathbb{G})^n$
 1: function D&S($(i_{\text{first}}, i_{\text{last}}) \in [\![0, n-1]\!]^2$)
 2:     if $i_{\text{last}} - i_{\text{first}} \le s_{max}$ then          ▷ Iterative case
 3:         return getStays $(T[i_{\text{first}}..i_{\text{last}}])$
 4:     end if
 5:     $S \leftarrow \emptyset$
 6:     $i_{\text{split}} \leftarrow \lfloor (i_{\text{first}} + i_{\text{last}})/2 \rfloor$
                                         ▷ Left sub-trace recursion
 7:     $t_\Delta \leftarrow T[i_{\text{split}}].t - T[i_{\text{first}}].t$
 8:     $d_\Delta \leftarrow \text{dist}_{\mathbb{G}}(T[i_{\text{first}}].g, T[i_{\text{split}}].g)$
 9:     if $\neg(d_\Delta > d_{max} \wedge t_\Delta \le t_{min})$ then
10:         $S \leftarrow$ D&S $(i_{\text{first}}, i_{\text{split}})$
11:     end if
                                         ▷ Right sub-trace recursion
12:     $t_\Delta \leftarrow T[i_{\text{last}}].t - T[i_{\text{split}}].t$
13:     $d_\Delta \leftarrow \text{dist}_{\mathbb{G}}(T[i_{\text{split}}].g, T[i_{\text{last}}].g)$
14:     if $\neg(d_\Delta > d_{max} \wedge t_\Delta \le t_{min})$ then
15:         $S \leftarrow S \cup$ D&S $(i_{\text{split}}, i_{\text{last}})$
16:     end if
17:     return $S$
18: end function
```

s_{max}, the sub-trace size threshold below which the divide-and-conquer approach for POI inference is abandoned in favor of the iterative one. Three indices are manipulated, all called i with a self-explanatory subscript. The D&S function takes i_{first} and i_{last} as arguments, being the bounds of the sub-trace under study. On the first call, the whole input space is provided: i_{first} is 0 and i_{last} takes $n-1$. Subsequent recursive calls provide either the first half of the input sub-trace, or the second, until an iterative search is preferred.

On lines 2 to 4, the size of the input sub-trace is checked against the threshold s_{max}. If the trace is smaller, then a linear search for *stays* is performed *à la* POI-Attack [Primault *et al.*, 2014]. On l. 6, the indices space is split: i_{split} is set to the midpoint between i_{first} and i_{last}. Lines 7-11 check whether the left sub-trace $T[i_{\text{first}}..i_{\text{split}}]$ is susceptible to contain *stays*, in which case D&S is recursively called. Its output fills the list of stays S. As already mentioned, a sub-trace cannot contain any *stay* if a distance of more than d_{max} was traveled in less than t_{min}. Lines 11 to 16 perform the same check for the right sub-trace $T[i_{\text{split}}..i_{\text{last}}]$, in which case the result of the recursive call is added to S. Finally, S is returned. POI-Attack's merge of *stays* into POIs must be subsequently performed.

The more discarded segments, the faster compared to the regular approach. *Stays* around the midpoints i_{split} could be missed, but D&S ignores them because a POI is a cluster of several stays: it is very unlikely to miss them all. D&S can be implemented sequentially or concurrently, to leverage multi-core processors.

4 Experimental Setup

4.1 Key Performance Metrics

Memory footprint. The key objective of FLI is to reduce the memory footprint required to store an unbounded stream of samples. We explore two metrics: *(i)* the number of 64-bit variables required by the model and *(ii)* the size of the model in the device memory. To do so, we compare the size of the persistent file with the size of the vanilla SQLite database file. We consider the number of 64-bit variables as a device-agnostic estimation of the model footprint.

I/O throughput. Another key system metric is the I/O throughput of the temporal databases. In particular, we measure how many write and read operations can be performed per second (IOPS).

POI quality. Measuring the quality of inferred POIs is difficult, as there is no acknowledged definition of how to compute POIs. We consider as our ground truth the POIs inferred by the state-of-the-art POI-attack [Primault *et al.*, 2014], which we refer to as the 'raw' POIs. The existence of such a 'ground-truth' is however debatable, as two different—but close—POIs can be merged by the algorithm into a single POI. As an example, if a user visits two different shops separated by a road, but their distance is lower than d_{max}, those will be merged into a single POI located at the center of the road. For that reason, we need two metrics to compare the sets of POIs returned in the different cases: the distance between POIs, and the sets' sizes.

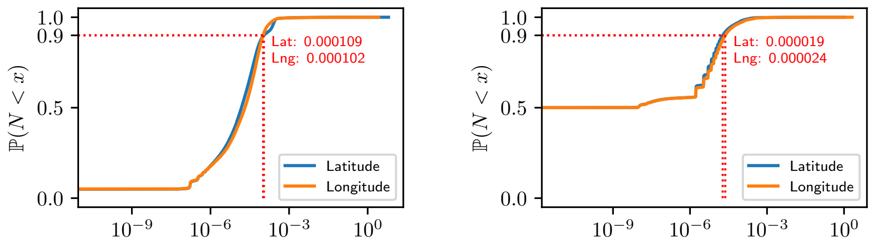
Distance between POIs. As the POI definition is mainly algorithmic, we compute the distance of each obtained POI to its closest raw POI as the metrics assessing the quality of new POIs. These distances are reported as a *Cumulative Distribution Function* (CDF). If FLI does not alter significantly the locations of the mobility traces it captures, the computed distances should be short.

Number of POIs. In addition to the distances between POIs, we are also considering their returned quantity as a metric. In our previous example, visiting the two shops may result in two different POIs because they have been slightly shifted by FLI. Beyond the numbers, we expect that Promesse successfully anonymizes mobility traces by returning a total of zero POI.

4.2 Input Datasets & Parameter Tuning

FLI was built to allow the storage of user-generated data series, such as GPS and accelerometer streams (although it is readily applicable to other types of real-valued streams, as shown in Section 5.5). However, the choice of the ϵ parameter depends on the underlying data distribution. Towards that end, we propose a semi-automated ϵ parameter selection routine.

In this section, we present the two datasets that will be used throughout our evaluation, before outlining and applying our parameter selection algorithm to the both of them.



(a) CDF of Cabspotting locations' drifts. **(b)** CDF of PrivaMov locations' drifts.

Figure 6. *Cumulative Distribution Function* (CDF) of latitude and longitude drifts of successive location samples in Cabspotting and PrivaMov datasets. One can observe that, from one location sample to the next, latitude or longitude deviations are small.

Location datasets. We use two real-world mobility datasets that display different characteristics:

- Cabspotting [Piorkowski *et al.*, 2009] is a mobility dataset of 536 taxis in the San Francisco Bay Area. The data was collected during a month and is composed of 11 million records, for a total of 388 MB. It is composed only of car trips in a dense urban environment.
- PrivaMov [Mokhtar *et al.*, 2017] is a multi-sensor mobility dataset gathered during 15 months by 100 users around the city of Lyon, France. It contains several transportation modes, most notably pedestrian. PrivaMov displays a much higher sampling rate that Cabspotting. We use the full GPS dataset, which includes 156 million records, totaling 7.2 GB.

Parameter tuning. The choice of an ϵ value is of major importance and plays a central role in FLI's performance: a poorly-chosen value has a strong impact on FLI's underlying segments, either degrading modeled data quality or filling storage space up excessively. To find a compromise between the two, since the ϵ value is highly correlated to the modeled data, one has to *know* the data; more specifically, we propose to study the signal's amplitude variation between consecutive values.

For example, in the context of location data, Fig. 6 characterizes—as a CDF—the evolution of longitude and latitude samples for all the traces stored in the Cabspotting and PrivaMov datasets. In particular, we plot the CDF of the drift d observed between 2 consecutive values (t_1, x_1) and (t_2, x_2), which we compute as $d = |x_2 - x_1|/|t_2 - t_1|$. One can observe that Cabspotting and PrivaMov datasets report on a drift lower than 1×10^{-4} and 2×10^{-5} for 90% of the values, respectively. Furthermore, due to the high density of locations captured by PrivaMov, half of the drifts are equal to 0, meaning several consecutive longitudes or latitudes are unchanged. This preliminary analysis highlights that FLI can indeed efficiently model mobility data, and demonstrates that $\epsilon = 10^{-3}$ is a conservative choice to model these datasets.

To automate the tuning of ϵ, one must input a sample of their data stream into our provided script[1], which proposes candidate ϵ parameter values to capture 90%, 95% and 99% of the sampled data. We discuss perspectives for parameter selection in Section 7.

[1]See file `lib/epsilon_choice.dart` in our code repository [Raes *et al.*, 2022b].

4.3 Storage Competitors

SQLite is the state-of-the-art solution to persist and query large volumes of data on Android devices. SQLite provides a lightweight relational database management system. SQLite is not a temporal database, but is a convenient and standard way to store samples persistently on a mobile device. Insertions are atomic, so one may batch them to avoid one memory access per insertion.

Sliding-Window And Bottom-up (SWAB) [Keogh *et al.*, 2001] is a linear interpolation model. As FLI, the samples are represented by a list of linear models. In particular, reading a sample is achieved by iteratively going through the list of models until the corresponding one is found and then used to estimate the requested value. The bottom-up approach of SWAB starts by connecting every pair of consecutive samples and then iterates by merging the less significant pair of contiguous interpolations. This process is repeated until no more pairs can be merged without introducing an error higher than ϵ. Contrarily to FLI, this bottom-up approach is an offline one, requiring all the samples to be known. SWAB extends the bottom-up approach by buffering samples in a sliding window. New samples are inserted in the sliding window and then modeled using a bottom-up approach: whenever the window is full, the oldest model is kept and the captured samples are removed from the buffer.

One could expect that the bottom-up approach delivers more accurate models than the greedy FLI, even resulting in a slight reduction in the number of models and faster readings. On the other hand, sample insertion is more expensive than FLI due to the execution of the bottom-up approach when storing samples. Like FLI, SWAB ensures that reading stored samples is at most ϵ away from the exact values.

Greycat [Moawad *et al.*, 2015] aims at compressing even further the data by not limiting itself to linear models. Greycat also models the samples as a list of models, but these models are polynomials. The samples are read the same way.

When inserting a sample, it first checks if it fits the model. If so, then nothing needs to be done. Otherwise, unlike FLI and SWAB which directly initiate a new model, Greycat tries to increase the degree of the polynomial to make it fit the new sample. To do so, Greycat first regenerates $d + 1$ samples in the interval covered by the current model, where d is the degree of the current model. Then, a polynomial regression of degree $d + 1$ is computed on those points along the new one. If the resulting regression reports an error lower than $\frac{\epsilon}{2^{d+1}}$, then the model is kept, otherwise, the process is repeated by incrementing the degree until either a fitting model is found or a maximum degree is reached. If the maximum degree is reached, the former model is stored and a new model is initiated. The resulting model is quite compact, and thus faster to read, but at the expense of an important insertion cost.

Unlike FLI and SWAB, there can be errors higher than ϵ for the inserted samples, as the errors are not computed on raw samples but on generated ones, which may not coincide. Furthermore, the use of higher-degree polynomials makes the implementation subject to overflow: to alleviate this effect, the inserted values are normalized.

4.4 Experimental Settings

For experiments with univariate data streams—*i.e.* memory and throughput benchmarks—we set $\epsilon = 10^{-2}$. The random samples used in those experiments follow a uniform distribution in $[-1{,}000; 1{,}000]$: it is very unlikely to have two successive samples with a difference lower than ϵ, hence reflecting the worst case conditions for FLI. For experiments on location data, and unless said otherwise, we set $\epsilon = 10^{-3}$ for FLI, SWAB and Greycat. For Greycat, the maximum degree for the polynomials is set to 14. The experiments evaluating the throughput were repeated 4 times each and the average is taken as the standard deviation was low. All the other experiments are deterministic and performed once.

4.5 Implementation Details

We implemented FLI using the Flutter *Software Development Kit* (SDK) [Google, 2018]. Flutter is Google's UI toolkit, based on the Dart programming language, that can be used to develop natively compiled apps for Android, iOS, web and desktop platforms (as long as the project's dependencies implement cross-compilation to all considered platforms). Our implementation includes FLI and its storage competitors.This implementation is publicly available [Raes *et al.*, 2022b].

For our experiments, we also implemented several mobile applications based on this library. To demonstrate its capability of operating across multiple environments (models, operating systems, processors, memory capacities, storage capacities), all our benchmark applications were successfully installed and executed in the devices listed in Table 1. Unless mentioned otherwise, the host device for the experiments is the Fairphone 3.

5 Experimental Results

In this section, we evaluate our implementation of FLI on Android and iOS to show how it enables efficient data stream storage on mobile devices. We first perform several benchmarks (memory, throughput & stability), before evaluating the performance of FLI beyond location streams. Finally, we perform a POI mining experiment directly on mobile devices, thus showcasing how INTACT enables *in-situ* big data processing.

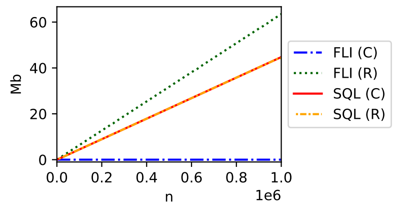


Figure 7. Inserting $1M$ samples, random (R) or constant (C), in SQLite and FLI.

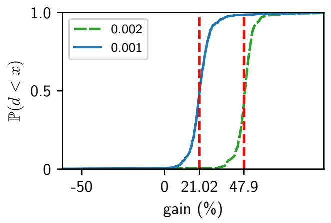


Figure 8. Memory gain distribution when storing Cabspotting with FLI.

5.1 Memory Benchmark

As there is no temporal database (e.g. InfluxDB), available on Android, we compare FLI's performances with SQLite, the only database natively available on Android.

Synthetic data. 2 identical operations are performed with SQLite and FLI: *(i)* the incremental insertion of random samples and *(ii)* the incremental insertion of constant samples. The memory footprint of both solutions on disk is compared when storing timestamped values. As FLI models the inserted samples, random values are the worst-case scenario it can face, while inserting constant values represents the ideal one. One million samples are stored and, for every 10,000 insertion, the size of the file associated with the storage solution is saved. The experiments are done with a publicly available application [Raes *et al.*, 2022d].

Fig. 7 depicts the memory footprint of both approaches. On the one hand, the size of the SQLite file grows linearly with the number of inserted samples, no matter the nature (random or constant) of the samples. On the other hand, the FLI size grows linearly with random values, while the size is constant for constant values. In particular, for the constant values, the required size is negligible. The difference between vanilla SQLite and FLI is explained by the way the model is stored: while SQLite optimizes the way the raw data is stored, FLI is an in-memory stream storage solution, which naively stores coefficients in a text file. Using more efficient storage would further shrink the difference between the two. As expected, the memory footprint of a data stream storage solution outperforms the one of a vanilla SQLite database in the case of stable values. While random and constant values are extreme cases, in practice data streams produced by ubiquitous devices exhibit a behavior between the two scenarios which allows FLI to lower the memory required to store those data streams.

GPS data. We use FLI to store latitudes and longitudes of the entire Cabspotting dataset (388MB) in memory, using both $\epsilon = 10^{-3}$ and $\epsilon = 2 \times 10^{-3}$ (representing an accuracy of approximately a hundred meters). For each user, we compute the gain of memory storage as a percentage, compared to storing the raw traces. Fig. 8 reports on the gain distribution as a CDF along with the average gain on the entire dataset. Most of the user traces largely benefit from using FLI, and FLI provides an overall gain of 21% (307MB) for

Table 1. Mobile devices used in the experiments.

Model	OS	CPU	Cores	RAM	Storage
Lenovo Moto Z	Android 8	Snapdragon 820	4	4 GB	32 GB
Fairphone 3	Android 11	Snapdragon 632	8	4 GB	64 GB
Pixel 7 Pro	Android 13	Google Tensor G2	8	12 GB	128 GB
iPhone 12	iOS 15.1.1	A14 Bionic	6	4 GB	64 GB
iPhone 14 Plus	iOS 16.0.1	A15 Bionic	6	6 GB	128 GB

$\epsilon = 10^{-3}$ on the entire dataset, and a gain of 47.9% (202MB) for $\epsilon = 2 \times 10^{-3}$.

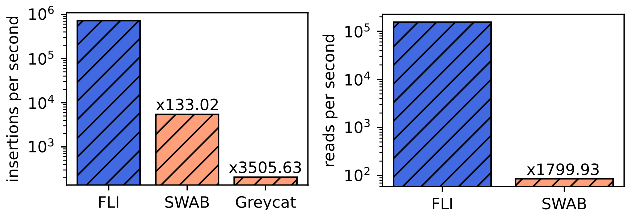
Additionally, we also compare SQLite and FLI to store the entire Privamov dataset (7.2GB). In this context, FLI only requires 25MB (gain of 99.65%) compared to more than 5GB (gain of 30.56%) for SQLite, despite the naive storage scheme used by FLI. Furthermore, with smartphones featuring limited RAM (cf. Table 1) and not allocating the whole of it to a single application, FLI enables loading complete datasets in memory to be processed: on mobile devices, loading the raw Privamov dataset in memory crashes the application (due to out-of-memory errors), while FLI succeeds in fitting the full dataset into RAM. This capability is particularly interesting to enable the deployment of data stream processing tasks on mobile devices that do not incur any processing overhead.

5.2 Throughput Benchmark

We compare FLI with its competitors among the temporal databases: SWAB and Greycat. We study the throughput of each approach in terms of IOPS. Insertion speed is computed by inserting $1M$ random samples (that is each of these solutions' worst-case scenario). For the reads, we also incrementally insert $1M$ samples before querying $10K$ random samples among the inserted ones. Greycat is an exception: due to its long insertion time (Sect. 4.3), we only insert $10K$ random values and those values are then queried. Our experiment is done using a publicly available application [Raes *et al.*, 2022e].

Fig. 9 depicts the throughput of the approaches for sequential insertions and random reads. On the one hand, FLI drastically outperforms its competitors for the insertions: it provides a speed-up from $\times$ 133 against SWAB up to $\times$ 3,505 against Greycat. The insertion scheme of FLI is fast as it relies on a few parameters. On the other hand, Greycat relies on a costly procedure when a sample is inserted: it tries to increase the degree of the current model until it fits with the new point or until a maximum degree is reached. Greycat aims at computing a model as compact as possible, which is not the best choice for fast online insertions.

For the reads (Fig. 9b), FLI also outperforms SWAB. Our investigation reports that FLI largely benefits from its dichotomy lookup inside the time index (see Alg. 1), compared to SWAB, which scans the list of models sequentially until the correct time index is found. SWAB reads have a complexity linear in the size of the list, while FLI has a logarithmic one. Greycat has the same approach as SWAB and this is why it is not represented in the results: with only $10K$ insertions instead of $1M$, its list of models is significantly smaller compared to the others, making the comparison unfair. Nevertheless, we expect Greycat to have a better throughput as its model list shall be shorter.



(a) Sequential Insertions **(b)** Random Reads

Figure 9. Throughput for insertions and reads using FLI, SWAB, and Greycat (log scale). FLI drastically outperforms its competitors for insertions and reads.

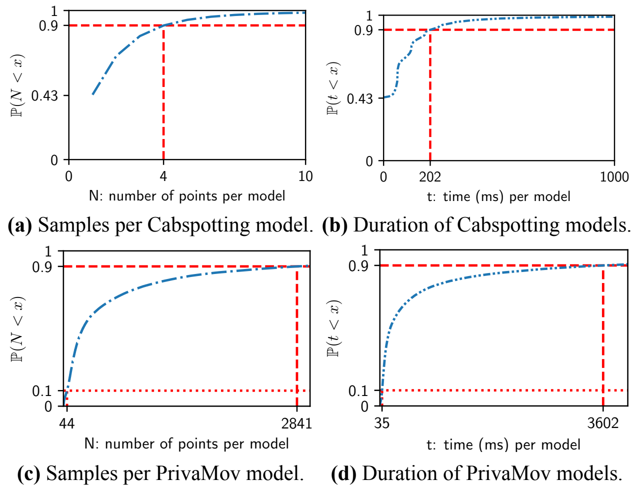
Note that those results have been obtained with the worst-case: random samples. Similarly, unfit for FLI are periodical signals, such as raw audio: our tests show a memory usage similar to random noise. Because FLI leverages linear interpolations, it performs best with signals that have a linear shape (e.g. GPS, accelerometer). We expect SWAB to store fewer models than FLI thanks to its sliding window, resulting in faster reads. However, the throughput obtained for FLI is minimal and FLI is an order of magnitude faster than SWAB for insertions, so it does not make a significant difference. We can conclude that FLI is the best solution for storing large streams of data samples on mobile devices.

5.3 Stability Benchmark

We further explore the capability of FLI to capture stable models that group as many data samples as possible for the longest possible durations. Fig. 10 reports on the time and the number of samples covered by the models of FLI for the Cabspotting and PrivaMov datasets. One can observe that the stability of FLI depends on the density of the considered datasets. While FLI only captures at most 4 samples for 90% of the models stored in Cabspotting (Fig. 10a), it reaches up to 2,841 samples in the context of PrivaMov (Fig. 10c), which samples GPS locations at a higher frequency than Cabspotting. This is confirmed by Fig. 10b and 10d, which report a time coverage of 202 ms and 3,602 ms for 90% of FLI models in Cabspotting and PrivaMov, respectively. Given that PrivaMov is a larger dataset than Cabspotting (7.2 GB vs. 388 MB), one can conclude that FLI succeeds in scaling with the volume of data to be stored.

5.4 Enabling *in-situ* big data processing

Location data is not only highly sensitive privacy-wise but also crucial for location-based services. While LPPMs have been developed to protect user locations, they are generally used on the server where the data is aggregated. The user



(a) Samples per Cabspotting model. **(b)** Duration of Cabspotting models.

(c) Samples per PrivaMov model. **(d)** Duration of PrivaMov models.

Figure 10. Stability of the FLI models on PrivaMov & Cabspotting with $\epsilon = 10^{-3}$.

location data is thus exposed to classical threats, such as malicious users, man in the middle, or database leaks. To avoid such threats, one privacy-preserving solution would be to keep the data in the device where it is produced until it is sufficiently obfuscated to be shared with a third party. With GPS data, this protection mechanism must be undertaken by a device-local LPPM. Evaluating the privacy of the resulting trace must also be performed locally, by executing attacks on the obfuscated data. Both processes require storing all the user mobility traces in the mobile device.

While existing approaches have simulated this approach [Khalfoun *et al.*, 2021], no real deployment has ever been reported. In this section, we show that using FLI enables overcoming one of the memory hurdles of constrained devices. We use FLI to store entire GPS traces in mobile devices, execute POI attacks, and protect the traces using the LPPM Promesse [Primault *et al.*, 2015].

Promesse [Primault *et al.*, 2015] is an LPPM that intends to hide POIs from a mobility trace by introducing a negligible spatial error. To do so, Promesse smooths the trajectories by replacing the mobility trace with a new one applying a constant speed while keeping the same starting and ending timestamps. The new trace T' is characterized by the distance δ between two points. First, additional locations are inserted by considering the existing locations one by one in chronological order. If the distance between the last generated location $T'[i]$ and the current one $T[c]$ is below δ, this location is discarded. Otherwise, $T'[i+1]$ is not defined as the current location $T[c]$, but the location between $T'[i]$ and $T[c]$, such that the distance between $T'[i]$ and $T'[i+1]$ is equal to δ. Once all the locations included in the new mobility trace are defined, the timestamps are updated to ensure that the period between the two locations is the same, keeping the timestamps of the first and last locations unchanged. The resulting mobility trace is protected against POI attacks while providing high spatial accuracy.

We, therefore, implemented a mobile version of POI attacks and Promesse [Primault *et al.*, 2015] using the Flutter library of FLI and evaluated its performances on the PrivaMov dataset. Fig. 11 depicts the resulting experimental deployments of POI attacks and the corresponding LPPM in mobile devices, alongside FLI. Our experiments are performed

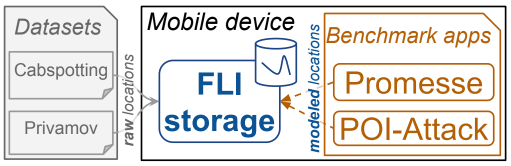


Figure 11. Architectural diagram of our *in-situ* big data processing experiment on a mobile device.

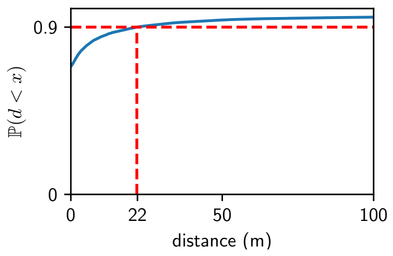


Figure 12. Distances distribution when using *Divide & Stay* on Cabspotting. The distances between the POIs are obtained using *Divide & Stay* and their closest counterparts, obtained with the traditional POI attack. Except for a few extreme values, the values are close: more than 68% are the same and 90% of the POIs are at a distance lower than 22 meters than a "real" one.

using a publicly available application [Raes *et al.*, 2022c].

Given that PrivaMov is the largest dataset, Table 2 reports on the worst-case processing times observed for `user 1` on different mobile platforms. This captures a user mobility trace of 4, 341, 716 locations reported by a mobile device. It shows that applying Promesse to the `user 1` mobility trace takes 1 second to run on all devices, on average. More interestingly, POI-Attack takes no longer than 30 seconds ($\pm\ 43ms$) to run on the latest generation hardware, both on Android and iOS. This is a significant improvement compared to the state-of-the-art implementation [Primault *et al.*, 2014], which requires one hour to run on a desktop computer. We believe that this is a critical step forward towards improving user privacy as all LPPM experiments until today were either simulated or centralized, contrary to what the literature suggests [Luxey *et al.*, 2018].

In addition to speed, the quality of the inferred POIs is the most salient concern about *Divide & Stay*. We assess the quality by computing the distances to the POIs obtained from the POI-attack on Cabspotting. We choose Cabspotting because computing it on PrivaMov is prohibitive in terms of computation time. Fig. 12 depicts the distribution of the distances below 100 meters: more than 68% are the same and 90% of the POIs are at a distance lower than 22 meters from actual ones. Fig. 13 shows some of the POI inferred from raw samples and the standard POI-attack (in blue) and POI obtained with FLI and D&S: they are different but very close, highlighting the negligible impact for applications that depend on location data.

Finally, Table 3 highlights that Promesse works well independently of the use of FLI or D&S: it successfully hides all the POI. Therefore, *Divide & Stay* provides an important speed-up without altering the quality of POIs. Note that FLI was not used in this case, as the performances of *Divide & Stay* are orthogonal to the use of a temporal database to model the samples.

Table 2. Execution time for PrivaMov `user 1` on different mobile platforms. Previous deployments of these algorithms were made on desktop computers. FLI now enables their instanciation on ubiquitous devices, with reasonable processing times.

Task	**Moto Z**	**Fair Phone 3**	**Google Pixel 7 Pro**	**iPhone 12**	**iPhone 14+**
Promesse	1.5s	1.3s	0.4s	0.2s	0.2s
POI-Attack	114.4s	109.2s	30s	18.8s	19.4s

Table 3. Impact of FLI and D&S on the number of inferred POIs from `user 0` trace in Cabspotting. Thanks to FLI and D&S, Promesse succeeds to protect user privacy at the edge.

	without Promesse		with Promesse	
Algorithm	**Raw POIs**	**FLI**	**Raw POIs**	**FLI**
POI-attack	30	31	0	0
D&S	30	30	0	0
POI-attack ∩ D&S	21	20	-	-

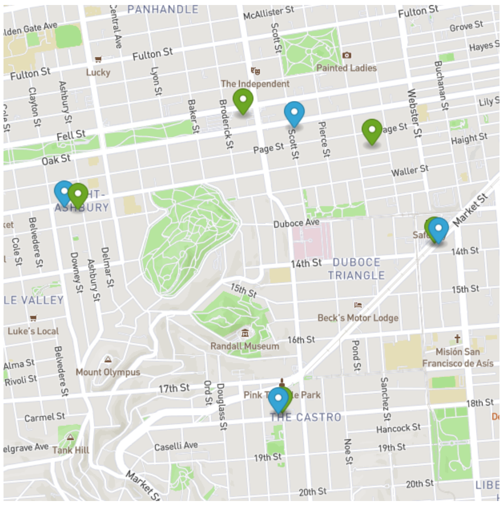
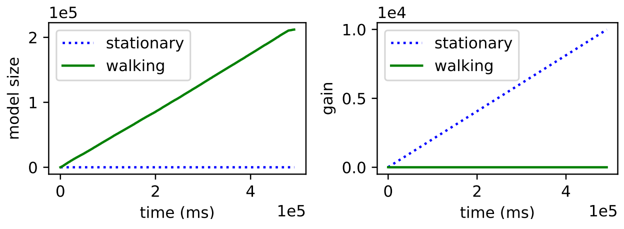


Figure 13. Example of POI of `user 0` from Cabspotting using the standard POI algorithm on raw data (in blue) and those obtained on the data modeled by FLI and computed with D&S (in green). The inferred POI are very close on average, but there are some outliers.

5.5 Beyond Location Streams

In this paper, FLI was mainly tested against location streams, but our proposal efficiently approximates any type of signal that varies mostly linearly: timestamps, accelerations, temperature, pressure, humidity, light, proximity, air quality, etc. This makes FLI a valuable candidate in ubiquitous contexts, as many physical quantities captured by e.g. IoT sensors have a piecewise linear behavior. To showcase different scenarios, we benchmark the storage of timestamps, device accelerations and heartbeat data using FLI. In any case, we use our ϵ-tuning script to capture the most appropriate value to store the data with a reduced error.

Storing timestamps. In all the previous experiments, the timestamps were not modeled by FLI, as we expect the user to query the time at which she is interested in the samples. However, it is straightforward to store irregular timestamps using FLI: we store couples (i, t_i) with t_i being the i^{th} inserted timestamp. The nature of the timestamps makes them a good candidate for modeling, as insertion rates are generally fixed, or vary linearly. To assess the efficiency of FLI for storing timestamps, we stored all the timestamps of the `user 1` of the PrivaMov dataset with $\epsilon = 1$—*i.e.*, we tolerate an error of one second per estimate. The 4,341,716 timestamps were stored using 26,862 models for a total of 80,592 floats and an overall gain of 98%, with a *mean average error* (MAE) of 0.246 second. Hence, not only does the use of FLI result in drastic memory savings, but it also provides accurate estimations.



(a) Model size **(b)** Memory gain

Figure 14. Model size and gain of FLI storing accelerometer values ($\epsilon = 1$). FLI reports a constant memory whenever stationary, and a small gain ($> \times 1.39$) when walking.

Storing device accelerations. Accelerometer data is important for many context-aware applications, including transport mode detection [Yu *et al.*, 2014; Fang *et al.*, 2016; Wang *et al.*, 2019]. Coupled with GPS information, it is possible to infer whether the user is walking, biking, taking a car or a tram. However, storing the output of a mobile accelerometer is particularly challenging, as it generates hundreds of noisy 3D samples per second. Our implementation is publicly available [Raes *et al.*, 2022a].

We store 10,000 consecutive accelerometer samples with FLI and, for every 100 insertions, we report on the size of the file and the relative gain. We use FLI with $\epsilon = 1$ as the accelerometer has high variability, even when the mobile is stationary. Fig. 14 reports on a constant memory footprint of FLI during the experiment, while providing a high-level accuracy.

Storing heartbeat pulses. We downloaded pulse-to-pulse intervals, which oscillate between 500 and 1,100 ms and are reported as the time duration between cardiac pulses to the timestamp of the original pulse, from a *Polar Ignite 2* [Polar, 2021] smartwatch, gathering 28,294,762 samples covering the 12 months of 2023, as a file of 259.1 MB. Using FLI to model this dataset with an accuracy of $\epsilon = 100$ reports on a non-negligible storage space gain of 26.44%, with a *mean error* of 22.74 milliseconds. FLI is thus a suitable solution to store data streams produced by various sensors of wearable and mobile devices, which could find application in e.g. human context recognition [Vaizman *et al.*, 2017].

To conclude, our implementation of the data stream storage solution, FLI, enables the effective deployment of more advanced techniques, such as EDEN [Khalfoun *et al.*, 2021] or HMC [Maouche *et al.*, 2018]. This may require new algorithms, such as *Divide & Stay*, but it enables *in situ* data privacy protection before sharing any sensitive information. We believe that this is a critical step forward towards improving user privacy as all LPPM experiments until today were either centralized or simulated.

Thanks to its FLI storage layer and *Divide & Stay* attack layer, INTACT improves privacy, and the cost of this improved privacy is *in situ* data computation, which requires running potentially heavy data tasks on constrainted devices. We report on the execution time of POI-Attack in Table 2, and argue computation time is the price to pay for an increased privacy.

6 Threats to Validity

While the combination of FLI and D&S succeeds in embedding LPPMs within mobile devices and increasing user privacy, our results might be threatened by some variables we considered.

The hardware threats relate to the classes of constrained devices we considered. In particular, we focused on the specific case of smartphones, which is the most commonly deployed mobile device in the wild. To limit the bias introduced by a given hardware configuration, we deployed FLI on both recent Android and iOS smartphones for most of the reported experiments, while we also considered the impact of hardware configurations on the reported performances.

Another potential bias relates to the mobility datasets we considered in the context of this paper. To limit this threat, we evaluated our solutions on two established mobility datasets, Cabspotting and PrivaMov, which exhibit different characteristics. Yet, we could further explore the impact of these characteristics (sampling frequency, number of participants, duration and scales of the mobility traces). Beyond mobility datasets, we could consider the evaluation of other IoT data streams, such as air quality metrics, to assess the capability of FLI to handle a wide diversity of data streams. To mitigate this threat, we reported on the storage of timestamps, acceleration and heartbeat data in addition to 2-dimensional locations.

Although FLI increases storage capacity through data modeling, it might still reach the storage limit of its host device if using a constant ϵ parameter (which drives the compression rate). To address this issue, we could dynamically adapt data compression to fit a storage size constraint. Toward this end, An *et al.* [An *et al.*, 2022] propose an interesting time-aware adaptive compression rate, based on the claim that data importance varies with its age.

Beyond the current implementation reported in this article, one could envision a native integration of INTACT in the Android and iOS operating systems to enable LPPMs for any legacy application. This technical challenge mostly consists of packaging FLI and D&S as a new `LocationProvider` in Android [Google, 2011] and a new `CLLocationManager` in iOS [Apple, 2013]. Interestingly, such a native integration of INTACT can allow end users to configure the list of enabled LPPMs and their related settings through the operating system control panel. Although not implemented in practice, INTACT includes a GPS provider part, made of LPPM protections and POI attacks, which can supply obfuscated locations to LBS apps; this is however straightforward to implement on mobile devices, using system abstractions both available in Android [Google, 2011] and iOS [Apple, 2013].

The increased storage capacity offered by FLI not only allows for unlimited mobility data storage, but also allows applying *in-situ* LPPMs requiring lots of data to work, for instance, those offering k and l-anonymity guarantees by hiding user among others [Sweeney, 2002; Machanavajjhala *et al.*, 2007].

Our implementations of FLI and D&S may suffer from software bugs that affect the reported performances. To limit this threat, we make the code of our libraries and applications freely available to encourage the reproducibility of our results and share the implementation decisions we took as part of the current implementation.

Finally, our results might strongly depend on the parameters we pick to evaluate our contributions. While FLI performances (gain, memory footprint) vary depending on the value of the ϵ parameter, we considered a sensitive analysis of this parameter and we propose a default value $\epsilon = 10^{-3}$ that delivers a minimum memory gain that limits the modeling error.

7 Perspectives

Beyond the deployment of LPPM at the edge of the network, we believe that our contributions open new avenues for improving user privacy and more generally enabling data processing at the edge. This includes the implementation of alternative LPPMs, the implementation of differential privacy algorithms, like *k-anonymity* and *l-diversity*, but also the inclusion of more privacy attacks to provide users with privacy reports for the data they intend to share.

FLI demonstrates that increasing the storage capacity of data streams on constrained devices offers new distributed computation models, inspired by the big data principles, that can deploy automated processing tasks on edge devices. Beyond LPPMs, one can think about the involvement of mobile devices in a federated or decentralized learning environment to train models without compromising sensitive personal information. Furthermore, we demonstrated with D&S that one can execute privacy attacks at the scale of a device, hence delivering support to assess the sensibility of a trained model before sharing it with some third party.

Regarding FLI storage capabilities, we think that the integration of compressed data could contribute to the further reduction of the memory footprint of the data streams stored on mobile devices, enabling computations over the older history of streams.

Regarding the ϵ parameter, we believe that its choice could not only be further automated by systematically applying the selection routine (outlined in Sec. 4.2) before the modeling starts, but could even be dynamically updated. This would allow for a dynamic data compression rate, which could be driven for instance by the data's age or its asserted importance. We plan on tackling this issue in future works.

While the literature in the field of mobile privacy is rich, we hope open source implementation of our Flutter libraries will encourage the research and developer communities to embed our contributions to strengthen the privacy of their users. We believe that FLI and D&S are key software assets that can stimulate the mobile database and privacy communities to deliver effective solutions on mobile devices, which are widely deployed nowadays.

8 Conclusion

Mobile devices are incredible producers of data streams, which are often forwarded to remote third-party services for storage and processing. This data processing pattern might be the source of privacy breaches, as the raw data may leak sensitive personal information. Furthermore, the volume of data to be processed may require huge storage capacity, from mobile devices to remote servers, and network capacity to deal with the increasing number of devices deployed in the wild.

On the other hand, while LPPMs are a promising solution to protect user locations, they are not widely deployed in practice, mostly because of their high computational cost, which is prohibitive on mobile devices.

We, therefore, proposed INTACT, a software framework that enables LPPMs on mobile devices by leveraging the increased storage capacity of mobile devices.

The contributions of INTACT are threefold: we leveraged *i)* a compact storage system based on a piece-wise linear model dubbed FLI (proposed in [Raes *et al.*, 2024]), *ii)* introduced a new way to compute POIs, called *Divide & Stay*, and finally *iii)* demonstrated how FLI could unlock device-local privacy protections on time series while using machine learning.

Additionally, we provided an operational Flutter package implementing them along with other existing temporal databases.

As a matter of perspectives, we therefore believe that FLI can be used as a building block for the development of privacy-friendly mobile applications, which can store and process data locally, without the need to rely on remote third-party services. In particular, one can explore the deployment of federated learning algorithms on mobile devices, which can be used to train machine learning models without compromising sensitive personal information. On the longer term, FLI paves the way to implement decentralized machine learning algorithms in mobile devices by leveraging device-to-device communication models, like Sprinkler [Luxey *et al.*, 2018]. For example, we believe *Sensitive Personal Information* (SPI) captured by ubiquitous devices should be anonymized locally *before* any data exchange.

While FLI can store tremendous data on mobile devices, *Divide & Stay* provides an important speed-up to reduce the total computation time of POI attacks by several orders of magnitude, making them suitable for mobile computing.

By sharing this INTACT framework with mobile developers, our contribution is an important step forward towards the real deployment of LPPMs and, more generally, privacy-friendly data-intensive workloads at the edge (*e.g.*, federated learning on mobile phones).

Acknowledgements

This research was supported in part by the Groupe La Poste, sponsor of the Inria Foundation, in the framework of the FedMalin Inria Challenge.

Authors' Contributions

Rémy Raes and Olivier Ruas contributed to the conception of the study and performed experiments. Both are the main contributors and writers of this manuscript. Adrien Luxey-Bitri and Romain Rouvoy supervised the work, wrote and revised the text. All authors read and approved the final manuscript. This work is an extension of Raes *et al.* [2024].

Competing interests

The authors declare no conflict of interest.

Availability of data and materials

Source code of the INTACT framework is publicly available in the Software Heritage repository at: `https://archive.softwareheritage.org/browse/origin/directory/?branch=refs/tags/jisa-2024-artefacts&origin_url=https://gitlab.inria.fr/Spirals/temporaldb_apps.git`

References

An, Y., Su, Y., Zhu, Y., and Wang, J. (2022). TV-Store: Automatically bounding time series storage via Time-Varying compression. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 83–100, Santa Clara, CA. USENIX Association. Available at: `https://www.usenix.org/conference/fast22/presentation/an`.

Andrés, M. E., Bordenabe, N. E., Chatzikokolakis, K., and Palamidessi, C. (2013). Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 901–914. DOI: 10.48550/arXiv.1212.1984.

Apple (2013). iOS `CLLocationManager` documentation. Available at: `https://developer.apple.com/documentation/corelocation/cllocationmanager` Last accessed on April 21st, 2024.

Bellet, A., Guerraoui, R., Taziki, M., and Tommasi, M. (2017). *Fast and differentially private algorithms for decentralized collaborative machine learning*. PhD thesis, INRIA Lille. Available at:

https://inria.hal.science/hal-01665410/file/Decentralized_Personalized_CD_Privacy.pdf.

Berlin, E. and Van Laerhoven, K. (2010). An on-line piecewise linear approximation technique for wireless sensor networks. In *IEEE Local Computer Network Conference*, pages 905–912. IEEE. DOI: 10.1109/lcn.2010.5735832.

Binder, S. (2019). Drift library. Availabe at: https://pub.dev/packages/drift Last accessed on April 21st, 2024.

Blalock, D., Madden, S., and Guttag, J. (2018). Sprintz: Time Series Compression for the Internet of Things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3). DOI: 10.1145/3264903.

Cerf, S., Primault, V., Boutet, A., Mokhtar, S. B., Birke, R., Bouchenak, S., Chen, L. Y., Marchand, N., and Robu, B. (2017). PULP: achieving privacy and utility trade-off in user mobility data. In *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, September 26-29, 2017*, pages 164–173. IEEE Computer Society. DOI: 10.1109/SRDS.2017.25.

Dollinger, V. and Junginger, M. (2014). Objectbox database. Available at: https://objectbox.io Last accessed on April 21st, 2024.

Dwork, C. (2008). Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer. DOI: 10.1007/978-3-540-79228-4_1.

Fang, S.-H., Liao, H.-H., Fei, Y.-X., Chen, K.-H., Huang, J.-W., Lu, Y.-D., and Tsao, Y. (2016). Transportation modes classification using sensors on smartphones. *Sensors*, 16(8):1324. DOI: 10.3390/s16081324.

Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R., and Kraska, T. (2019). FITing-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1189–1206. DOI: 10.1145/3299869.3319860.

Gambs, S., Killijian, M.-O., and del Prado Cortez, M. N. (2014). De-anonymization attack on geolocated data. *Journal of Computer and System Sciences*, 80(8):1597–1614. DOI: 10.1109/trustcom.2013.96.

Google (2011). Android LocationManager documentation. Available at: https://developer.android.com/reference/android/location/LocationManager#addTestProvider(java.lang.String,%20android.location.provider.ProviderProperties,%20java.util.Set%3Cjava.lang.String%3E) Last accessed on April 21st, 2024.

Google (2018). Flutter framework. Available at: https://flutter.dev/ Last accessed on April 21st, 2024.

Grützmacher, F., Beichler, B., Hein, A., Kirste, T., and Haubelt, C. (2018). Time and memory efficient online piecewise linear approximation of sensor signals. *Sensors*, 18(6):1672. DOI: 10.3390/s18061672.

Hariharan, R. and Toyama, K. (2004). Project lachesis: parsing and modeling location histories. In *International Conference on Geographic Information Science*, pages 106–124. Springer. DOI: 10.1007/978-3-540-30231-5_8.

InfluxData (2013). InfluxDB. Available at: https://www.influxdata.com/products/influxdb-overview/ Last accessed April 21st, 2024.

Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2001). An online algorithm for segmenting time series. In *Proceedings 2001 IEEE international conference on data mining*, pages 289–296. IEEE. DOI: 10.1109/icdm.2001.989531.

Khalfoun, B., Ben Mokhtar, S., Bouchenak, S., and Nitu, V. (2021). EDEN: Enforcing location privacy through re-identification risk assessment: A federated learning approach. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(2). DOI: 10.1145/3463502.

Liu, X., Lin, Z., and Wang, H. (2008). Novel online methods for time series segmentation. *IEEE Transactions on Knowledge and Data Engineering*, 20(12):1616–1626. DOI: 10.1109/tkde.2008.29.

Luxey, A., Bromberg, Y.-D., Costa, F. M., Lima, V., da Rocha, R. C. A., and Taïani, F. (2018). Sprinkler: A probabilistic dissemination protocol to provide fluid user interaction in multi-device ecosystems. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. DOI: 10.1109/percom.2018.8444577.

Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. (2007). l-Diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es. DOI: 10.1109/ICDE.2006.1.

Maouche, M., Ben Mokhtar, S., and Bouchenak, S. (2018). HMC: Robust privacy protection of mobility data against multiple re-identification attacks. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):1–25. DOI: 10.1145/3264934.

Maouche, M., Mokhtar, S. B., and Bouchenak, S. (2017). Ap-attack: a novel user re-identification attack on mobility datasets. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 48–57. DOI: 10.4108/eai.7-11-2017.2273573.

Meftah, L., Rouvoy, R., and Chrisment, I. (2019). Fougere: user-centric location privacy in mobile crowdsourcing apps. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 116–132. Springer. DOI: 10.1007/978-3-030-22496-7_8.

Moawad, A., Hartmann, T., Fouquet, F., Nain, G., Klein, J., and Le Traon, Y. (2015). Beyond discrete modeling: A continuous and efficient model for IoT. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 90–99. IEEE. DOI: 10.5555/3351736.3351751.

Mokhtar, S. B., Boutet, A., Bouzouina, L., Bonnel, P., Brette, O., Brunie, L., Cunche, M., D'Alu, S., Primault, V., Raveneau, P., *et al.* (2017). PRIVA'MOV: Analysing human mobility through multi-sensor datasets. In *NetMob 2017*. Available at: [https://inria.hal.science/hal-01578557].

Piorkowski, M., Sarafijanovic-Djukic, N., and Grossglauser, M. (2009). CRAWDAD data set epfl/mobility (v. 2009-02-24).

Polar (2021). Ignite 2. Available at: https://www.polar.com/en/ignite2 Last accessed on April 21st, 2024.

Primault, V., Mokhtar, S. B., Lauradoux, C., and Brunie, L. (2014). Differentially private location privacy in practice. *arXiv preprint arXiv:1410.7744*. DOI: https://doi.org/10.48550/arxiv.1410.7744.

Primault, V., Mokhtar, S. B., Lauradoux, C., and Brunie, L. (2015). Time distortion anonymization for the publication of mobility data with high utility. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 539–546. IEEE. DOI: 10.1109/trustcom.2015.417.

Raes, R., Ruas, O., Luxey-Bitri, A., and Rouvoy, R. (2022a). *Fast Linear Interpolation* accelerometer example application. Available at: `https://archive.softwareheritage.org/browse/revision/570c744aaa82ce8c8f75fce53234013b001872fb/?origin_url=https://gitlab.inria.fr/Spirals/temporaldb_apps.git&path=temporaldb/example&revision=570c744aaa82ce8c8f75fce53234013b001872fb` Hosted on Software Heritage Last accessed on November 4th, 2024.

Raes, R., Ruas, O., Luxey-Bitri, A., and Rouvoy, R. (2022b). *Fast Linear Interpolation* implementation. Available at: `https://archive.softwareheritage.org/browse/revision/570c744aaa82ce8c8f75fce53234013b001872fb/?origin_url=https://gitlab.inria.fr/Spirals/temporaldb_apps.git&path=temporaldb&revision=570c744aaa82ce8c8f75fce53234013b001872fb` Hosted on Software Heritage Last accessed on November 4th, 2024.

Raes, R., Ruas, O., Luxey-Bitri, A., and Rouvoy, R. (2022c). In-situ LPPM. Available at: `https://archive.softwareheritage.org/browse/revision/570c744aaa82ce8c8f75fce53234013b001872fb/?origin_url=https://gitlab.inria.fr/Spirals/temporaldb_apps.git&path=in_situ_lppm&revision=570c744aaa82ce8c8f75fce53234013b001872fb` Hosted on Software Heritage Last accessed on November 4th, 2024.

Raes, R., Ruas, O., Luxey-Bitri, A., and Rouvoy, R. (2022d). Memory space benchmarking application. Available at: `https://archive.softwareheritage.org/browse/revision/570c744aaa82ce8c8f75fce53234013b001872fb/?origin_url=https://gitlab.inria.fr/Spirals/temporaldb_apps.git&path=benchmarking_memory_space&revision=570c744aaa82ce8c8f75fce53234013b001872fb` Hosted on Software Heritage Last accessed on November 4th, 2024.

Raes, R., Ruas, O., Luxey-Bitri, A., and Rouvoy, R. (2022e). Throughput benchmarking application. Available at: `https://archive.softwareheritage.org/browse/revision/570c744aaa82ce8c8f75fce53234013b001872fb/?origin_url=https://gitlab.inria.fr/Spirals/temporaldb_apps.git&path=benchmarking_throughput&revision=570c744aaa82ce8c8f75fce53234013b001872fb` Hosted on Software Heritage Last accessed on November 4th, 2024.

Raes, R., Ruas, O., Luxey-Bitri, A., and Rouvoy, R. (2024). Compact Storage of Data Streams in Mobile Devices. In *DAIS'24 - 24th International Conference on Distributed Applications and Interoperable Systems*, Proceedings of the 24th International Conference on Distributed Applications and Interoperable Systems (DAIS'24), Groningen, Netherlands. LNCS. DOI: 10.1007/978-3-031-62638-8$_4$.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570. DOI: 10.1142/s0218488502001648.

Tamplin, J. and Lee, A. (2012). Firebase services. Available at: `https://firebase.google.com` Last accessed on April 21st, 2024.

Timescale (2019). Building a distributed time-series database on PostgreSQL. Available at: `https://www.timescale.com/blog/building-a-distributed-time-series-database-on-postgresql/` Last accessed on May 12th 2023.

Timescale Inc (2018). Timescale database. Available at: `https://www.timescale.com` Last accessed on April 21st, 2024.

Vaizman, Y., Ellis, K., and Lanckriet, G. (2017). Recognizing Detailed Human Context in the Wild from Smartphones and Smartwatches. *IEEE Pervasive Computing*, 16(4). DOI: 10.1109/MPRV.2017.3971131.

Wang, L., Gjoreski, H., Ciliberto, M., Mekki, S., Valentin, S., and Roggen, D. (2019). Enabling reproducible research in sensor-based transportation mode recognition with the sussex-huawei dataset. *IEEE Access*, 7:10870–10891. DOI: 10.1109/access.2019.2890793.

Wolfson, O., Chamberlain, S., Dao, S., Jiang, L., and Mendez, G. (1998). Cost and imprecision in modeling the position of moving objects. In *Proceedings 14th International Conference on Data Engineering*, pages 588–596. DOI: 10.1109/ICDE.1998.655822.

Xu, K., Yue, H., Guo, L., Guo, Y., and Fang, Y. (2015). Privacy-preserving machine learning algorithms for big data systems. In *2015 IEEE 35th international conference on distributed computing systems*, pages 318–327. IEEE. DOI: 10.1109/icdcs.2015.40.

y Arcas, B. A. (2018). Decentralized machine learning. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1–1. IEEE. DOI: 10.1109/bigdata.2018.8622078.

Yu, M.-C., Yu, T., Wang, S.-C., Lin, C.-J., and Chang, E. Y. (2014). Big data small footprint: The design of a low-power classifier for detecting transportation modes. *Proceedings of the VLDB Endowment*, 7(13):1429–1440. DOI: 10.14778/2733004.2733015.

Zhou, C., Frankowski, D., Ludford, P., Shekhar, S., and Terveen, L. (2004). Discovering personal gazetteers: an interactive clustering approach. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 266–273. DOI: 10.1145/1032222.1032261.