INTACT: Storing Unbounded Data Streams on Mobile Devices to Unlock User Privacy at the Edge

Anonymous authors
Paper under double-blind review

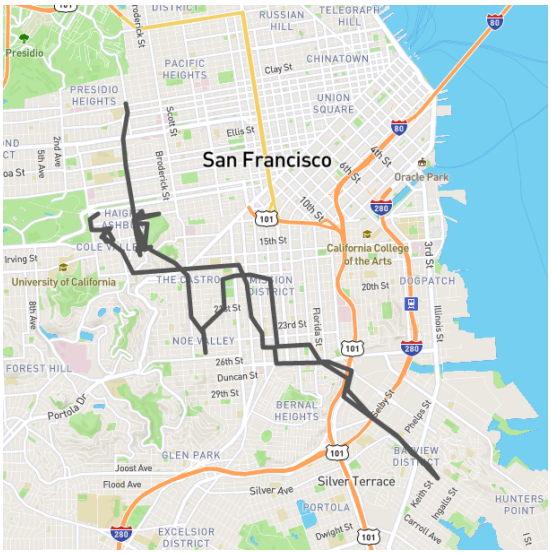
Abstract

Mobile devices are producing larger and larger data streams, such as location streams, which are consumed by machine learning pipelines to deliver location-based services to end users. 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 effectively been implemented, nor deployed in mobile devices to enforce user privacy at the edge of a network. We believe that the effective deployment of LPPMs on mobile devices faces a major challenge: the storage of unbounded data streams.

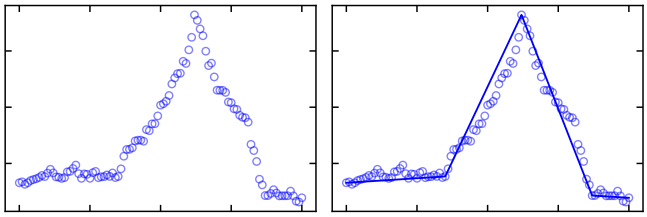
This article introduces INTACT, a cross-platform framework that leverages a piece-wise linear approximation technique, dubbed FLI, to increase the storage capacity of mobile devices. Then, we combine this storage capability with *Divide & Stay*, a new privacy preservation technique to execute *Points of Interest* (POIs) inference. By enabling *in situ* POIs inference, the sensitivity of location streams can be assessed to better enforce user privacy. Finally, we deploy all INTACT components on Android and iOS to demonstrate that a real deployment of LPPMs on mobile phones is now possible.

1 Introduction

With the advent of smartphones and more generally the *Internet of Things* (IoT), connected 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 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. Online services are heavily relying on this crowdsourced data to improve the user experience through machine learning, an example being Waze [1] live-monitoring traffic. 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 daily samples. These data streams tend to be uploaded

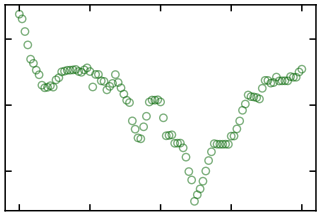


(a) Cabspotting subtrace for `user 0`.

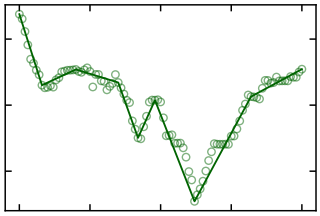


(b) Raw longitude trace for `user 0`.

(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, obtained from a piece-wise model.

from mobile devices to third-party service providers to extract the valuable information they contain. Notably, the *Points of Interest* (POIs) of a user can be extracted from their GPS traces. They represent locations where the user stopped, such as their home, workplace or social locations; POIs provide detailed information on people's behavior.

This continuous data stream inevitably includes *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 [13]. On the one hand, without crowd-sourced 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.

To address this ethical challenge, privacy-preserving machine learning [42] and decentralized machine learning [10, 43] 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 [3] generalizes differential privacy [15] to GPS traces, while PROMESSE [37] smooths the GPS traces—both temporally and geographically—to erase POIs from the input trace. LPPMs successfully preserve sensitive data, such as POIs, while maintaining the data utility for the targeted service.

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) [26] 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. Data processing algorithms are additionally not optimized for mobile devices.

This article demonstrates that modeling data streams enables us to address these device-level storage & processing challenges. To address the storage challenge, we first introduce *Fast Linear Interpolation* (FLI), a time series algorithm leveraging a *Piece-wise Linear Approximation* (PLA) technique to model and store data streams under memory constraints. Figure 1 displays FLI's behavior: it does not store

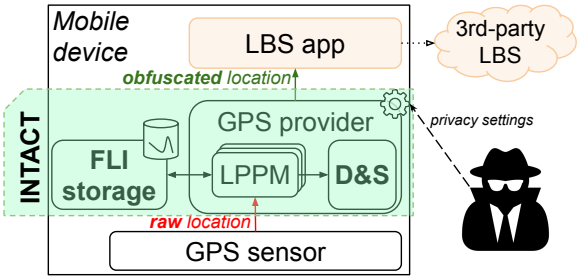


Figure 2: Overview of INTACT contributions.

raw data samples (Fig. 1b and 1d), but instead models their evolution as linear interpolations (Fig. 1c and 1e) —thus offering a much bigger storage capacity at the cost of a controlled approximation error. Then, to illustrate the processing challenge, we implement a LPPM working directly on mobile phones—which is made possible by the increased GPS storage capacity offered by FLI. However, the LPPM's privacy gains need to be evaluated *in situ* before being uploaded to service providers: are POIs effectively obfuscated? To this end, we also introduce a new POI attack algorithm, dubbed *Divide & Stay* (D&S), which can extract POI from large mobility traces stored in the device in tens of seconds. Altogether, these components form the *IN-siTu locAtion proteCTion* (INTACT) framework. Figure 2 displays its overall architecture, and how our FLI and D&S contributions work together with LPPMs to provide an auditable location privacy framework for mobile devices. All of INTACT's evaluations are performed directly on Android and iOS smartphones, demonstrating its readiness.

In the following, we first discuss the related work (Sec. 2), before diving into the details of FLI and D&S, and how they are combined to boost location privacy (Sec. 3). We then present our experimental setup (Sec. 4) and the results we obtained (Sec. 5) Finally, we discuss the limitations of our approach (Sec. 6), before concluding (Sec. 7).

2 Related Works

2.1 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 her 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 [18, 36]. The POIs can be extracted using spatiotemporal clustering algorithms [22, 45]. Alternatively, an attacker may also re-

identify a user directly from raw traces, without computing any POI [30].

2.2 Mobility Dataset Protection Mechanisms

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* [38] 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* [28] extends *k-anonymity* by ensuring that the l users are diverse enough not to infer SPI about the data owner. Finally, *differential privacy* [15] 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.

2.3 Location Privacy Protection Mechanisms

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 directly on the device before any data exchange.

***Geo-Indistinguishability* (GEOI) [3]** implements differential privacy [15] 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) [29] 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 [37] 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 [26, 31]. Because of potential remote leaks, the user should anonymize her trace locally before sharing it, which is how EDEN [26] 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 [29], for instance, requires locally loading a large set of GPS traces to operate.

2.4 Temporal Databases & Mobile Devices

To overcome the memory constraints of mobile devices, one needs efficient embedded temporal databases. To take the example of Android: only few databases are available, such as SQLITE and its derivative DRIFT [12], the cloud-supported Firebase [39], the NOSQL HIVE, and OBJECTBOX [14]. 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 from data, while not putting the emphasis on read nor write performances. Finally, in temporal workloads, it is unlikely to process writes & reads in the same single transaction [40].

Despite these profound differences, several relational databases offer support for temporal data with industry-ready performance. As an example, TIMESCALEDB [23] is a middleware that exposes temporal functionalities atop a relational POSTGRESQL foundation.

INFLUXDB [24] is one of the most widely used temporal databases. Implemented in Go, this high-performance time series engine is designed for really fast writes to collect metrics and events from IoT sensors. Unfortunately, its retention policy prevents the storage to scale in time: the oldest samples are dumped to make room for the new ones.

To the best of our knowledge, none of the existing solutions prioritize data compression to the extent that they would *prune* raw data samples in favor of *modeled* approximations.

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, *Piecewise 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 [25], however, is an online approach that uses a sliding window to buffer the latest samples on which a bottom-up approach is applied. EMSWAB [11] 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 [21]. For example, GREYCAT [32] adopts polynomial regressions with higher degrees to further compress the data. Unfortunately, none of those works have been implemented on mobile devices to date.

Closer to our work, FSW [27] and the SHRINKINGCONE algorithm [17] 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.

3 Enforcing User Privacy at the Edge

INTACT is a framework that protects the location privacy of its user while also being auditable *in situ*: directly on their mobile phone. As shown on Figure 2, INTACT stores the user's mobility traces using our *Fast Linear Interpolation* (FLI) module, after they have been obfuscated by a *Location Privacy Protection Mechanism* (LPPM). Geolocation attacks can be carried out to assert that no privacy leak remains. Our privacy analysis focuses on *Points of Interest* (POIs): small geographic areas where one remains for some time, thus unveiling their personal habits (home, workplace, social activities...). We deploy the PROMESSE [37] LPPM, that aims at erasing POIs from mobility traces, and the POI-Attack [36], which objective is to disclose POIs. We contribute an optimization of the POI-Attack, called *Divide & Stay* (D&S), to enable POI inference on mobile devices. The next subsection motivates our proposal. Then, FLI is presented in details in Sec. 3.2. Finally, we describe D&S in Sec. 3.3.

3.1 *In situ* Data Management

For privacy's sake, we advocate for *in situ* data management strategies—*i.e.*, *Sensitive Personal Information* (SPI) should be anonymized within the mobile device *before* any data exchange. This avoids anonymizing by relying on a trusted third party first gathering multiple users' raw data. Such a third party may accidentally or intentionally leak users' data, making the adoption of such protection mechanisms ineffective.

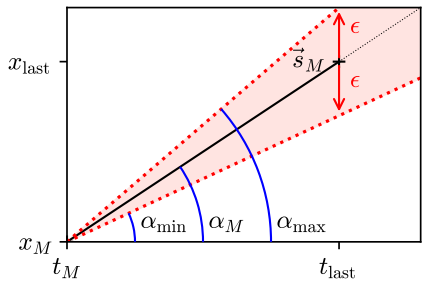


Figure 3: A new FLI interpolation $\vec{s}_M$ begins with two samples: the penultimate point $p_M = (t_M, x_M)$ as origin, and the latest sample $p_{\text{last}} = (t_{\text{last}}, x_{\text{last}})$ which defines the slope α_M. Two bounding gradients $\alpha_{\min}$ and $\alpha_{\max}$ are derived from the configuration parameter ε, and will be used to assert whether future inserts fit the current interpolation.

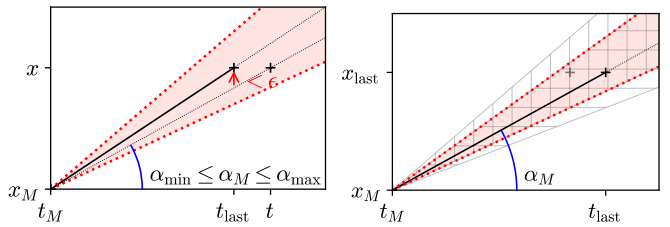
We believe that keeping the raw data where it is created—*i.e.*, in mobile devices—increases user privacy. However, sharing data is required to enable location-based services, such as traffic modeling. The user should share their mobility traces *after* they have been protected using an LPPM. The first challenge is to find which LPPM to use and which related parameters are optimal. To tackle this issue, a public dataset can be used to estimate the impact of an LPPM and to pick the best option. EDEN [26] proposes a more advanced solution: federated learning is used among the participants to learn a model which can predict the best configuration without sharing any mobility trace. Nonetheless, both approaches require storing an important volume of data to successfully protect user privacy.

The strong resource constraints of mobile devices prevent the previous solutions to work in practice. In particular, mobile ecosystems lack tooling for efficient local storage. Not only is there no advanced database readily available on mobile operating systems, but no native data modeling framework is provided either. For example, EDEN was implemented using the PYTORCH library [34], which is not available on smartphones[1]: the proposal was only simulated on a server. It is therefore crucial to deliver tools enabling the deployment of *state-of-the-art* techniques on mobile devices to support privacy-preserving strategies at the edge of a network.

3.2 Unleashing Your Device Storage with FLI

To overcome the memory constraint of mobile devices, efficient temporal storage solutions must be ported onto mobile environments. In particular, we advocate the use of data modelling, such as *Piece-wise Linear Approximation* (PLA) [21, 25] or GREYCAT [32], to increase the storage capacity of constrained devices. We propose FLI, a timeseries-modeling algorithm based on an iterative and continuous

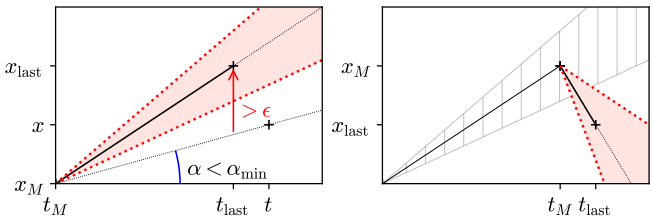
[1] PyTorch allows importing and using trained models on Android and iOS, but disallows training them locally.



(a) Gradient of sample p remains within $[\alpha_{\min}, \alpha_{\max}]$.

(b) α_M, $\alpha_{\min}$ and $\alpha_{\max}$ are updated to incorporate p into the current model.

Figure 4: When a new sample fits within $[\alpha_{\min}, \alpha_{\max}]$, it is added to the current model by updating $\alpha_{\min}$ and $\alpha_{\max}$ to ensure that all previous samples fit the updated model. α_M is also updated for read queries (see alg. 2).



(a) Error of $p = (t, x)$ exceeds ε.

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

Figure 5: When a new sample reports an error $> \varepsilon$, a new model is created using the penultimate sample p_{last} as p_M.

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.

FLI models one-dimensional points (or samples) p as piecewise linear segments (or interpolations) $\vec{s}$. It enforces the following invariant: *all samples modeled by an interpolation maintain an error below the configuration parameter* ε. Its data structure $\mathcal{D}$ is composed of a list of selected historical points $\mathcal{P}$, the latest segment's gradient α_M, and the two bounding gradients $\alpha_{\min}$ and $\alpha_{\max}$ used for insertion:

$$\mathcal{D} = (\mathcal{P}, \alpha_M, \alpha_{\min}, \alpha_{\max}), \text{ s.t.}$$
$$\mathcal{P} = [\ldots, p_i, p_{i+1}, \ldots, p_M] \subset \mathbb{R}^2 \ \&\ (\alpha_M, \alpha_{\min}, \alpha_{\max}) \in \mathbb{R}^3$$

Historical segments are represented by couples of consecutive samples: $\vec{s}_i = [p_i, p_{i+1}]$. The latest interpolation $\vec{s}_M$ takes the last inserted sample p_M as its origin and the gradient α_M as its slope, as displayed in Fig. 3. We first present how observed points are inserted, before explaining how reading a value is performed.

Insertion Data samples are inserted incrementally: the current interpolation is adjusted to fit new samples until it cannot satisfy the invariant. Upon insertion of a new sample p, the slope α of the segment $[p_M, p]$ is compared to the interval $[\alpha_{\min}, \alpha_{\max}]$: If it falls within (as shown in Fig. 4), this sample is added to the current interpolation: $\alpha_{\min}$ and $\alpha_{\max}$ are updated to encompass the new sample in the check, α_M is updated to α (for reading), and the previous sample is dropped. If p is outside the interval (see Fig. 5), a new interpolation begins from the two last observed points. This approach is lightweight memory and compute-wise, as all the needed information regarding modeled points is embedded in the above interval, and it only takes one check per insertion to update the model.

Algorithm 1 details a new sample's insertion $p = (t, x)$. As mentioned, on lines 2-3, FLI first computes the gradient α of the segment $[p_M, p]$. If α is inside $[\alpha_{\min}, \alpha_{\max}]$, p validates

Algorithm 1 FLI insertion using parameter $\varepsilon \in \mathbb{R}^{+*}$

Require: $\mathcal{D} = (\mathcal{P}, \alpha_M, \alpha_{\min}, \alpha_{\max}), (t_{\text{last}}, x_{\text{last}}) \in \mathbb{R}^2$

1: **function** INSERT($(t, x) \in \mathbb{R}^2$)
2: $(t_\Delta, x_\Delta) \leftarrow (t - t_M, x - x_M)$ ▷ Compute α
3: $\alpha \leftarrow x_\Delta / t_\Delta$
4: **if** $\alpha_{\min} < \alpha < \alpha_{\max}$ **then**
5: $\alpha_M \leftarrow \alpha$ ▷ Update model
6: $\alpha_{\min} \leftarrow \max\left(\alpha_{\min}, \frac{x_\Delta - \varepsilon}{t_\Delta}\right)$
7: $\alpha_{\max} \leftarrow \min\left(\alpha_{\max}, \frac{x_\Delta + \varepsilon}{t_\Delta}\right)$
8: **else**
9: $\mathcal{P} \leftarrow \mathcal{P} \cup [(t_{\text{last}}, x_{\text{last}})]$ ▷ Build new model
10: $(t_\Delta, x_\Delta) \leftarrow (t - t_{\text{last}}, x - x_{\text{last}})$
11: $\alpha_M \leftarrow x_\Delta / t_\Delta$
12: $\alpha_{\min} \leftarrow (x_\Delta - \varepsilon) / t_\Delta$
13: $\alpha_{\max} \leftarrow (x_\Delta + \varepsilon) / t_\Delta$
14: **end if**
15: $(t_{\text{last}}, x_{\text{last}}) \leftarrow (t, x)$ ▷ Update last sample
16: **end function**

the invariant and is thus added to the current model by updating α_M, $\alpha_{\min}$ and $\alpha_{\max}$ (lines 5-7). This case is illustrated in Figure 4. Graphically, we see that the resulting 'allowed cone' is the intersection of the model's previous one, and that of p's allowed error. By recurrence, the cone materialized by p_M and $[\alpha_{\min}, \alpha_{\max}]$ is the intersection of the error margin of every point embedded inside the current interpolation.If α falls outside the interval $[\alpha_{\min}, \alpha_{\max}]$, p breaks the invariant and a new interpolation needs to be computed, as depicted in Fig. 5. It begins from the last two observed samples: $\vec{s}_M$ becomes $[p_{\text{last}}, p]$. The penultimate sample p_{last} is thus persisted to the list of selected points $\mathcal{P}$ at l. 9, and new values for $\alpha_M, \alpha_{\min}$ and $\alpha_{\max}$ are derived based on the new $\vec{s}_M$ (lines 10-13). In any case, the penultimate sample p_{last} is updated on line 15.

Read In FLI, reading a value at time t is achieved by estimating its image using the appropriate interpolation, as is shown in algorithm 2. If t is ulterior or equal to t_M, the current interpolation $\vec{s}_M$ is used (line 3), defined by $p_M = (t_M, x_M)$ and α_M. When t is anterior to t_M, FLI reconstructs the in-

Algorithm 2 FLI approximate read

Require: $\mathcal{D} = (\mathcal{P}, \alpha_M, \alpha_{\min}, \alpha_{\max})$

1: **function** READ($t \in \mathbb{R}$)
2: **if** $t_M \leq t$ **then**
3: **return** $\alpha_M \times (t - t_M) + x_M$
4: **end if**
5: Select i s.t. $((t_i, x_i), (t_{i+1}, x_{i+1})) \in \mathcal{P} \wedge t_i \leq t < t_{i+1}$
6: $\alpha_i \leftarrow (x_{i+1} - x_i) / (t_{i+1} - t_i)$
7: **return** $\alpha_i \times (t - t_i) + x_i$
8: **end function**

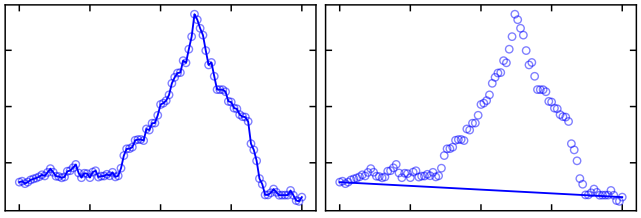
terpolation $\vec{s}_i$ in charge of approximating t by picking two consecutive points from $\mathcal{P}$ (lines 5-6). In practice, the search is made through a dichotomy search, as $\mathcal{P}$ stores points in insertion order. Using that model, the interpolation of t is computed on line 7.

The value of ε has an important impact on the performances of FLI. Figure 6 illustrates the longitude of Fig. 1 with two extreme values for ε. If ε is too small (Figure 6a), none of the inserted samples fits the current model at that time, 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 (Figure 6b), 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 point and underfits the data.

While FLI is designed for the modeling of one-dimensional data, it straight-forwardly generalizes to multiple-dimensional data by combining several instances of FLI. As long as the newly inserted data samples fit the existing model, the memory footprint of FLI remains unchanged. This potentially unlimited storage capacity makes FLI a key asset for mobile devices, making the storage of mobility traces possible. We claim that the use of FLI alleviates the memory constraint of mobile devices, making the real use of LPPM possible and paving the way for user control of SPI.

3.3 Evaluating Your Location Privacy with D&S

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 [36] iterates linearly over the mobility trace and



(a) ε is too small: overfitting. (b) ε is too large: underfitting.

Figure 6: The performances of FLI are highly dependent on the value of ε: a too-small value will result in overfitting and a too-large one in underfitting.

compute *stays* as they appear. This approach is expensive for denser mobility traces—*i.e.*, with high frequency sampling. It is prohibitively long 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}. For example, 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 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 [36]. On l. 6, the indices space in 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

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** $i_{\text{last}} - i_{\text{first}} \leq s_{max}$ **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** $\neg(d_\Delta > d_{max} \wedge t_\Delta \leq t_{min})$ **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** $\neg(d_\Delta > d_{max} \wedge t_\Delta \leq t_{min})$ **then**
15: $S \leftarrow S \cup$ D&S $(i_{\text{split}}, i_{\text{last}})$
16: **end if**
17: **return** S
18: **end function**

$T[i_{\text{split}}..i_{\text{last}}]$, 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

This section presents observed indicators used to affirm the value of FLI's contribution to mobile machine learning on time series. We then introduce datasets that were used to assert FLI's storage capabilities. Next, we present competing solutions that were also implemented in benchmark applications to compare with FLI's performances. Finally, we discuss experimentation settings.

4.1 Key Performance Metrics

To evaluate how our approach performs, we use two classes of key performance metrics: system metrics and privacy-related metrics. Concerning privacy-related experiments, we only measure the computation time when evaluating *Divide & Stay*. Those metrics highly depend on the chosen algorithms, while the use of FLI has no impact. Since our objective is to demonstrate that FLI can help to port state-of-the-art LPPM techniques on constrained devices, we do not discuss privacy-related metrics for other experiments.

Memory footprint The key objective of FLI is to reduce the memory footprint required to store an unbounded stream of samples. More specifically, 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.

We will compare POI-inference algorithms, and POIs computed by the same algorithm using different data backends. 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.

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 [36], 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.

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 Mobility Datasets

In the following, we will focus on mobility traces. We believe that mobility traces are a good candidate for FLI as the

storage of user mobility traces may require a lot of storage space. A mobility trace is an ordered sequence T of pairs (t,g) where t is a timestamp and g is a geolocation sample, a latitude-longitude pair for example. The trace is ordered in chronological order and we assume that reported timestamps are unique.

Datasets CABSPOTTING [35] 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. PRIVAMOV [33] is a multi-sensors mobility dataset gathered during 15 months by 100 users around the city of Lyon, France. We use the full GPS dataset, which includes 156 million records, totaling 7.2 GB. Compared to CABSPOTTING, PRIVAMOV is a highly-dense mobility dataset.

Drifts Figure 7 characterizes the evolution of longitude and latitude samples for all the traces stored in the CABSPOTTING and PRIVAMOV datasets. In particular, we plot the GPS of the drift d observed along two consecutive values (x_1,y_1) and (x_2,y_2), which we compute as $d = |(y_2-y_1)/(x_2-x_1)|$. 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 that the consecutive longitudes and latitudes are unchanged. Our preliminary analysis of both datasets highlights that mobility traces are relevant candidates for FLI. The following sections will focus on the evaluation of FLI on those datasets to study the benefits of adopting FLI to capture real-world metrics in mobile devices.

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) [25]** 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.

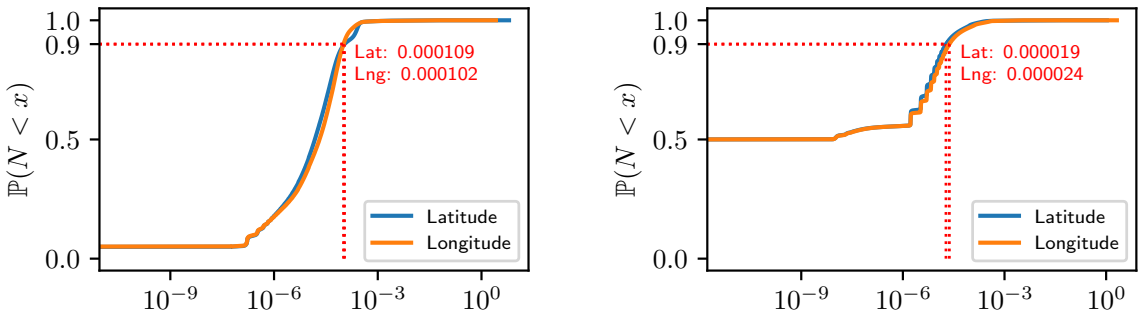
GREYCAT [32] aims at compressing even further the data by not limiting itself to linear models. GREYCAT also models the samples by a list of models, but these models are polynomials. The samples are read the same way.

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 higher than $\frac{\varepsilon}{2^{d+1}}$, 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 unidimensional data—*i.e.* memory and throughput benchmarks—we set $\varepsilon = 10^{-2}$. The random samples used in those experiments are following 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 $\varepsilon = 10^{-3}$ for FLI, SWAB and Greycat. For GREYCAT, the maximum degree for the polynomials is set to 14. For POI computations, we use $t_{min} = 5$ min and a diameter of $D_{max} = 500$ m for both



(a) CDF of CABSPOTTING locations' drifts.

(b) CDF of PRIVAMOV locations' drifts.

Figure 7: CDF of latitude and longitude drifts of successive location samples in CABSPOTTING and PRIVAMOV datasets. From one location sample to the next, latitude and longitude deviations are small.

Table 1: Mobile devices used in the experiments.

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

the standard approach and D&S. Similarly, we use $\delta = 500$ m for PROMESSE: it should remove all the POIs from the traces.

The experiments evaluating the throughput were repeated four 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 INTACT using the Flutter *Software Development Kit* (SDK) [20]. 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, its storage competitors, the POI-attack with and without our D&S extension, and PROMESSE. This implementation is publicly available [5].

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 installed and executed in the listed devices, as summarized in Table 1.

5 Experimental Results

In this section, we evaluate our implementation of FLI on Android and iOS to show how it can enable *in-situ* data management on mobile devices. We first show that using FLI

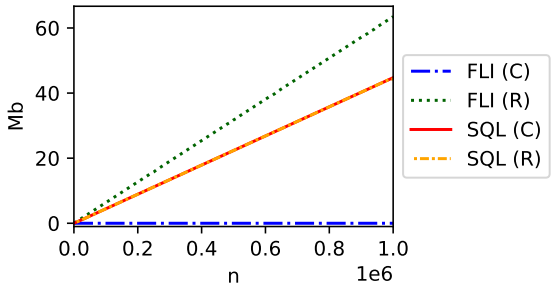


Figure 8: Insertion of 1,000,000 samples, random (R) or constant (C), in both SQLITE and FLI.

paves the way for storing a tremendous quantity of samples, by comparing it to SQLITE and reporting its performances when storing samples generated by the accelerometer. Then, we deploy the PROMESSE LPPM directly on mobile thanks to FLI. Still, on the mobile phones, we evaluate traces using our POI-attack *Divide & Stay* (D&S): to assess the precision of the GPS time series modeled by FLI, and the privacy gain of the LPPM.

5.1 Memory Benchmark

As there is no temporal database, such as INFLUXDB, available on Android, We first compare FLI's performances with SQLITE, as it is the only database natively available on Android.

To compare the memory consumption of the two approaches, two same operations are performed with both SQLITE and FLI: *(i)* the incremental insertion of random samples and *(ii)* the incremental insertion of constant samples. The memory footprint on the disk of both solutions 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 [7].

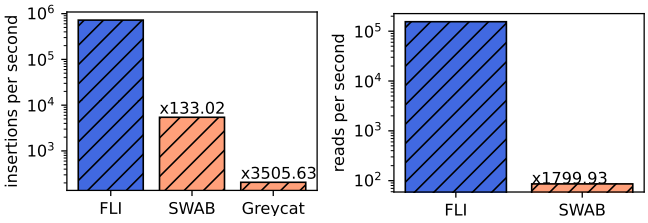
Figure 8 depicts the memory footprint of both approaches. On the one hand, the size of the SQLITE file grows linearly with the number of inserted samples, no matter the nature (random or constant) of the samples. On the other hand, the FLI size grows linearly with random values, while the size is constant 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 exhibit a behavior between the two scenarios which allows FLI to lower the memory required to store those data streams.

Beyond the above synthetic dataset, we compare SQLITE and FLI to store the entire PRIVAMOV dataset (7.2GB). FLI only requires 25MB compared to more than 5GB for SQLITE, despite the naive storage scheme used by FLI. Furthermore, on mobile devices, loading the raw dataset in memory crashes the application, while FLI fits the whole dataset into memory (cf. Table 1).

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 insertions and readings per second. For the insertions, we successively insert $1M$ random samples in the storage solution (random values are used as a worst-case situation for FLI, due to its way of modeling data). For the reads, we also incrementally insert $1M$ samples before querying 10,000 random samples among the inserted ones. GREYCAT is an exception: due to its long insertion time, we only insert 10,000 random values and those values are then queried. Our experiment is done using a publicly available application [8].

Figure 9 shows the throughput of the approaches for sequential insertions and random reads. Note the logarithmic scale. 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. While SWAB performs better, it cannot



(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.

compare to FLI because of the way SWAB inserts a sample: when its sliding window is full and a new sample does not fit the current model, a costly bottom-up approach is triggered over the entire window.

For the reads (Fig. 9b), FLI also outperforms SWAB. Our investigation reports that the gain reported by FLI largely benefits from the time index it exploits to fetch the models: SWAB browses the list of models sequentially until the good model is found while FLI relies on a dichotomy search, both lists having roughly the same size as random samples were added. SWAB has a complexity linear in the size of the models' 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 10,000 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.

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 an unbounded stream of samples on mobile devices.

5.3 Privacy Benchmark

5.3.1 Location privacy

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 data is thus exposed to classical threats, such as malicious users, man in the middle, or database leaks. To avoid such threats,

the best solution is 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 [26], 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 [37].

PROMESSE [37] 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.

Our experiments are performed using a publicly available application [6].

Enforcing privacy on CABSPOTTING Using FLI, we store the entire CABSPOTTING dataset's latitudes and longitudes in memory, using both $\varepsilon = 10^{-3}$ and $\varepsilon = 2 \times 10^{-3}$ (representing an accuracy of approximately a hundred meters). For each user, we compute the gain in terms of memory we save by modeling the dataset instead of storing the raw traces.

Figure 10 reports on the gain distribution as a CDF along with the average gain on the entire dataset. One can observe that most of the user traces benefit from using FLI, and FLI provides an overall gain of 21% for $\varepsilon = 10^{-3}$ on the entire dataset, and a gain of 47.9% for $\varepsilon = 2 \times 10^{-3}$. Nonetheless, the mobility of a few users imposes an important cost: for them, using FLI is counter-productive. Fortunately, this does not balance out the gain for the other users.

To better understand how the ε parameter introduced by FLI affects the utility of the resulting traces, we study the POIs inferred from the modeled traces. We compute the POIs of the trace both with and without using FLI. To estimate the relevance of the obtained POIs, we compute the distance

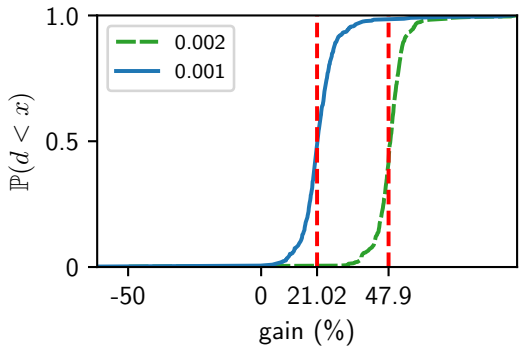
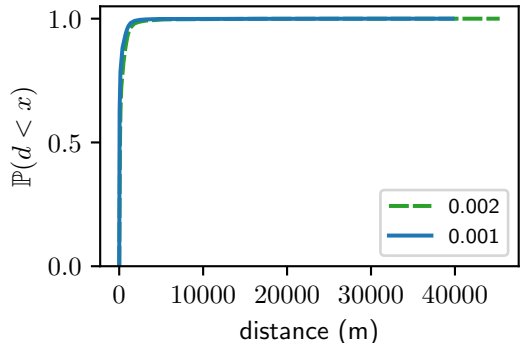


Figure 10: Memory gain distribution when storing CABSPOTTING with FLI. Using FLI with $\varepsilon = 10^{-3}$ reports on a gain of 21%, while $\varepsilon = 2 \times 10^{-3}$ reaches a gain of 48%.

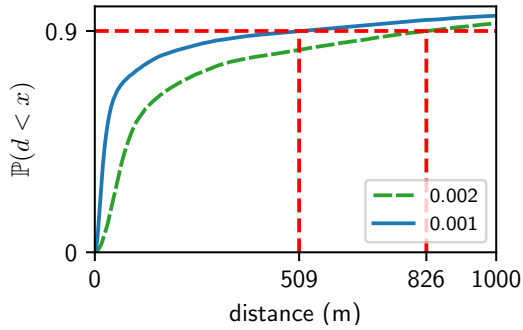
of each POI reported while using the trace modeled by FLI to the closest POI among the POIs in the raw trace. Figure 11 depicts the distribution, as a CDF, of this distance between "modeled" and "raw" POIs. Figure 11a shows that the distance is short: with $\varepsilon = 10^{-3}$, 99.5% of the distances are lower than 2,425 meters and 99% are lower than 1,700 meters. Figure 11b zooms on this distribution, focusing on distances lower than 1,000 meters. With $\varepsilon = 10^{-3}$, 90% of the obtained POIs using FLI are at a distance lower than 510 meters to a POI inferred from the raw trace. By construction, POIs are the center of spheres of a diameter of 500 meters where the user has stayed more than 5 minutes. The vast majority of the obtained POIs using FLI being within 500 meters of raw POIs, which means that FLI delivers relevant approximations. With $\varepsilon = 2 \times 10^{-3}$, 90% of the obtained POIs using FLI are at a distance lower than 826 meters to a POI inferred from the raw trace: the gain in memory has an impact on the utility of the resulting trace.

Figure 12 reports on the sensibility analysis of ε, both in terms of gains and distances. As expected, the higher ε, the better the gains, but the longest the distances. Regarding the gains (Fig. 12a), a low ε can induce a memory overhead. Indeed, if the model is used only for one data point, it generates a memory overhead similar to Figure 8, in this case, 50%. We, therefore, recommend using $\varepsilon = 10^{-3}$ as the minimal tolerated error to observe a gain. Regarding the distances, Figure 12b reports on the distribution of distances below 1,000 meters, as the higher values follow the same tendency as Figure 11a. Except for a few extreme values, most of the distances remain short, even for high ε values.

Processing Benchmark For dense datasets, *e.g.* with more than two GPS samples per second, the gain becomes even more significant. For example, storing the entire PRIVAMOV dataset using FLI with $\varepsilon = 10^{-3}$ results in **a memory gain of** 99.87**%**. Compared to sampling, FLI stores all the samples, instead of discarding a part of them. However, the large

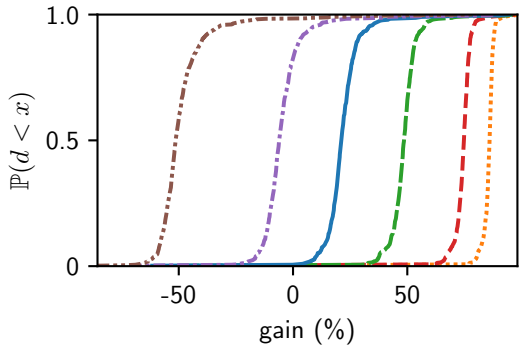


(a) Global distribution

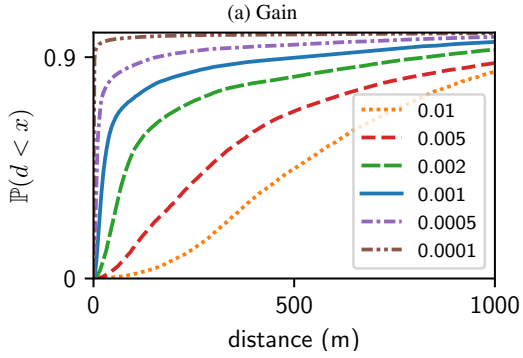


(b) Zoom for $d < 1{,}000$ m

Figure 11: Distances distribution when using FLI on CABSPOTTING. The distances are computed between the POIs obtained using the modeled traces and their closest counterparts, obtained with the raw traces. Except for a few extreme values, the values are close: 90% of the POIs are at a distance lower than 510 meters from the ground truth. The use of FLI does not alter the utility of the traces.



(a) Gain



(b) Distribution for $d < 1{,}000$ m

Figure 12: Distances distribution for different ε when using FLI on CABSPOTTING. Distances and memory gain are computed from the modeled traces with different values for ε. The higher ε, the higher the gain, but the longer the distances between the inferred and raw POI.

Table 2: Computation times of raw POIs for PRIVAMOV `user 1` on different platforms. *Divide & Stay* (D&S) is at least 100 times faster than state-of-the-art approaches.

Platform	POI-attack	D&S	Speed-up
Desktop	59 min 20 s	32 s	×111
iOS	1 h 00 min 01 s	22 s	×164
Android	1 h 58 min 04 s	59 s	×120

number of samples can be a hindrance to many approaches, including the extraction of POI. To be able to port LPPMs onto constrained devices, other bottlenecks of the systems should be resolved, in addition to storage.

For example, computing POIs with the traditional POI attacks may lead to unpractical computation time. Our preliminary investigation reported that computing the POIs of the `user 1` of PRIVAMOV takes 2 *hours*: computing the POIs for the entire dataset is far too costly. We cannot expect end-users to execute processes with such computation time on their mobile phone: while FLI has removed the memory constraint, computation time is still a hurdle. *Divide & Stay* is a way, in this case, to decrease the complexity of POI computation. Table 2 displays PRIVAMOV `user 1` POIs' computation time on different platforms. It shows that applying *Divide & Stay* to the `user 1` mobility trace decreases the computation from 2 *hours* to 59 *seconds* on Android, providing a ×120 speed-up; speed gain even reaches ×164 on iOS, computation time decreasing from 1 *hour* to 22 *seconds*. *Divide & Stay* makes the *in-situ* use of POI attacks and the corresponding LPPM possible.

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. Figure 13 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. 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.

Bringing back privacy to the user. By using both FLI and D&S we can perform POI-attacks and use LPPMs directly on the user's device. We consider the POIs of `user 0` of CABSPOTTING with and without FLI, D&S, and PROMESSE, see Table 3.

The use of FLI and D&S alters the number of POIs, which explains the extreme values obtained in the distribution of the distances (Fig. 11 and 12b): it corresponds to POIs that have no counterpart and may be far away from other POIs. The use of D&S corroborates the results of Figure 13: an important

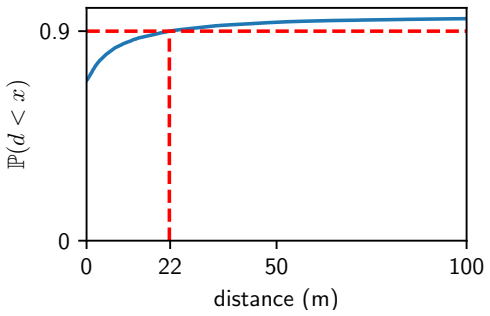


Figure 13: Distances distribution when using *Divide & Stay* on CABSPOTTING. The distances between the POIs are obtained using *Divide & Stay* and their closest counterparts, obtained with the traditional POI attack. Except for a few extreme values, the values are close: more than 68% are the same and 90% of the POIs are at a distance lower than 22 meters than a "real" one.

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.

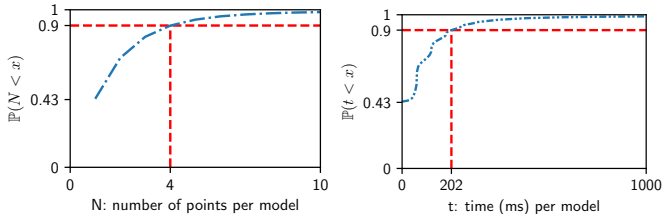
	without PROMESSE		with PROMESSE	
Algorithm	**Raw POIs**	**FLI**	**Raw POIs**	**FLI**
POI-attack	30	31	0	0
D&S	30	30	0	0
POI-attack ∩ D&S	21	20	-	-

part of the inferred POIs look similar to the raw ones. On the other hand, even though the number of POIs is similar, none of the POIs obtained using FLI is equal to the original one, with or without D&S, despite being very close.

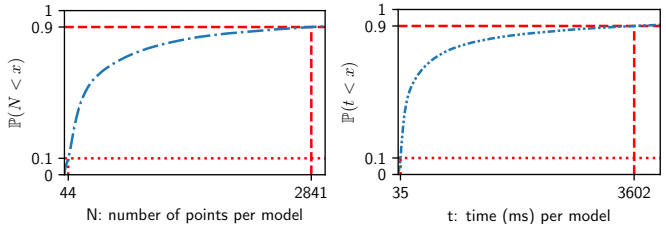
To conclude, our implementation of the data stream storage solution, FLI, enables the effective deployment of more advanced techniques, such as EDEN [26] or HMC [29]. This may require new algorithms, such as *Divide & Stay*, but it enables *in situ* data privacy protection before sharing any sensitive information. We believe that this is a critical step forward towards improving user privacy as all LPPMs experiments until today were either centralized or simulated.

5.4 Stability Benchmark

We further explore the capability of FLI to capture stable models that group as many samples as possible for the longest possible durations. Figure 14 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. 14a), it reaches up to 2,841 samples in the context of PRIVAMOV (Fig. 14c), which samples GPS locations at a higher frequency



(a) Samples per CABSPOTTING model. (b) Duration of CABSPOTTING models.



(c) Samples per PRIVAMOV model. (d) Duration of PRIVAMOV models.

Figure 14: Stability of the inferred models when using FLI on PRIVAMOV and CABSPOTTING with $\varepsilon = 10^{-3}$.

than CABSPOTTING. This is confirmed by our observations of Figures 14b and 14d, 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 to scale with the volume of data to be stored.

5.5 Beyond Location Streams

While INTACT explores the use of FLI for location streams, we believe that FLI can also be used for other types of data streams, which can threaten user privacy. We, therefore, believe that LPPMs can benefit from FLI to store other types of data streams, such as timestamps, accelerations, or any other signals captured by IoT devices.

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 timestamps using FLI: we store couples (i, t_i) with t_i being the i^{th} inserted timestamp. Unlike other sensor samples, the nature of the timestamps makes them a good candidate for modeling: their value keeps increasing in a relatively periodic fashion. To assess the efficiency of FLI for storing timestamps, we stored all the timestamps of the `user 1` of the PRIVAMOV dataset with $\varepsilon = 1$—*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 the use of FLI results in a dramatic gain of memory, but it provides very good estimations.

Storing accelerations To assess that FLI is suitable for storing unbounded data streams, we use FLI to store accelerometer samples. While storing random samples is of little benefit, accelerometer samples are used in practice to model user mobility. Coupled with other sensors' data, such as GPS values, we can infer if the user is walking, biking or taking a car for example [16, 41, 44]. However, the accelerometer produces more than 15 samples per second, hence challenging the storage of such a data stream. Our implementation is publicly available [4].

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 $\varepsilon = 1$ as the accelerometer has high variability, even when the mobile is stationary. FLI reports a constant memory whenever stationary, and a small gain ($> \times 1.39$) when walking. FLI is thus a suitable solution to store data streams produced by the sensors of mobile devices.

We also observed that the performances of FLI may differ, depending on device configurations (cf. Table 1). As older hardware's accelerometers are noisier and produce fewer samples than newer sensors, FLI's gain appears as higher on latter-generation hardware. For instance, inserting $10k$ samples with a *Pixel 7 Pro (Android 13)* smartphone is completed in 21 seconds, while doing the same on a *Moto Z (Android 8)* lasts for 49 seconds. Regarding iOS, latest *iPhone 14 Plus (iOS 16.0.1)* takes up 1 minute 39 seconds to store same samples count.

6 Perspectives

While the combination of FLI and D&S succeeds to embed 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 both FLI and D&S 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 and accelerations 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.* [2] 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 [19] and a new `CLLocationManager` in iOS [9]. 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.

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 [28, 38].

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 $\varepsilon = 10^{-3}$ that delivers a minimum memory gain that limits the modeling error.

7 Conclusion

While LBS are mainstream applications, they are also a major threat to user privacy if user locations are not properly protected. 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 this INTACT are threefold: we introduced *i)* a compact storage system based on a piece-wise linear model dubbed FLI, *ii)* 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. Our extensive evaluations, based on real mobile applications available for Android and iOS, highlight that FLI drastically outperforms its competitors in terms of insertion throughput—FLI is more than 130 times faster than the traditional SWAB—and read throughput—FLI reads 1,800 times faster than SWAB. 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).

References

[1] Waze. https://www.waze.com/. Last accessed on May 12th, 2023.

[2] Yanzhe An, Yue Su, Yuqing Zhu, and Jianmin Wang. TVStore: 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, February 2022. USENIX Association. URL: https://www.usenix.org/conference/fast22/presentation/an.

[3] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 901–914, 2013.

[4] Anonymous. *Fast Linear Interpolation* accelerometer example application. https://anonymous.4open.science/r/temporalbddflutter/example/README.md, 2022. Last accessed on July 27th, 2023.

[5] Anonymous. *Fast Linear Interpolation* implementation. https://anonymous.4open.science/r/temporalbddflutter, 2022. Last accessed on July 27th, 2023.

[6] Anonymous. In-situ lppm. https://anonymous.4open.science/r/in-situ-lppm, 2022. Last accessed on July 27th, 2023.

[7] Anonymous. Memory space benchmarking application. https://anonymous.4open.science/r/benchmarking_memory_space, 2022. Last accessed on July 27th, 2023.

[8] Anonymous. Throughput benchmarking application. https://anonymous.4open.science/r/benchmarking_throughput, 2022. Last accessed on July 27th, 2023.

[9] Apple. ios cllocationmanager documentation. https://developer.apple.com/documentation/corelocation/cllocationmanager, 2013. Last accessed on July 26th, 2023.

[10] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. *Fast and differentially private algorithms for decentralized collaborative machine learning*. PhD thesis, INRIA Lille, 2017.

[11] Eugen Berlin and Kristof Van Laerhoven. An on-line piecewise linear approximation technique for wireless sensor networks. In *IEEE Local Computer Network Conference*, pages 905–912. IEEE, 2010.

[12] Simon Binder. Drift library. https://pub.dev/packages/drift, 2019. Last accessed on May 12th, 2023.

[13] Sophie Cerf, Vincent Primault, Antoine Boutet, Sonia Ben Mokhtar, Robert Birke, Sara Bouchenak, Lydia Y. Chen, Nicolas Marchand, and Bogdan Robu. 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, 2017. doi:10.1109/SRDS.2017.25.

[14] Vivien Dollinger and Markus Junginger. Objectbox database. https://objectbox.io, 2014. Last accessed on May 12th, 2023.

[15] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[16] Shih-Hau Fang, Hao-Hsiang Liao, Yu-Xiang Fei, Kai-Hsiang Chen, Jen-Wei Huang, Yu-Ding Lu, and Yu Tsao. Transportation modes classification using sensors on smartphones. *Sensors*, 16(8):1324, 2016.

[17] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. Fiting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1189–1206, 2019.

[18] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. De-anonymization attack on geolocated data. *Journal of Computer and System Sciences*, 80(8):1597–1614, 2014.

[19] Google. Android locationmanager documentation. https://developer.android.com/reference/android/location/LocationManager#addTestProvider(java.lang.String,%20android.location.provider.ProviderProperties,%20java.util.Set%3Cjava.lang.String%3E), 2011. Last accessed on July 26th, 2023.

[20] Google. Flutter framework. https://flutter.dev/, 2018. Last accessed on May 12th, 2023.

[21] Florian Grützmacher, Benjamin Beichler, Albert Hein, Thomas Kirste, and Christian Haubelt. Time and memory efficient online piecewise linear approximation of sensor signals. *Sensors*, 18(6):1672, 2018.

[22] Ramaswamy Hariharan and Kentaro Toyama. Project lachesis: parsing and modeling location histories. In *International Conference on Geographic Information Science*, pages 106–124. Springer, 2004.

[23] Timescale Inc. Timescale database. https://www.timescale.com, 2018. Last accessed on Sep May 12th, 2023.

[24] InfluxData. Influxdb. https://www.influxdata.com/products/influxdb-overview/, 2013. last accessed May 12th, 2023.

[25] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Proceedings 2001 IEEE international conference on data mining*, pages 289–296. IEEE, 2001.

[26] Besma Khalfoun, Sonia Ben Mokhtar, Sara Bouchenak, and Vlad Nitu. 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):1–25, 2021.

[27] Xiaoyan Liu, Zhenjiang Lin, and Huaiqing Wang. Novel online methods for time series segmentation. *IEEE Transactions on Knowledge and Data Engineering*, 20(12):1616–1626, 2008.

[28] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[29] Mohamed Maouche, Sonia Ben Mokhtar, and Sara Bouchenak. 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, 2018.

[30] Mohamed Maouche, Sonia Ben Mokhtar, and Sara Bouchenak. 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, 2017.

[31] Lakhdar Meftah, Romain Rouvoy, and Isabelle Chrisment. Fougere: user-centric location privacy in mobile crowdsourcing apps. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 116–132. Springer, 2019.

[32] Assaad Moawad, Thomas Hartmann, François Fouquet, Grégory Nain, Jacques Klein, and Yves Le Traon. 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, 2015.

[33] Sonia Ben Mokhtar, Antoine Boutet, Louafi Bouzouina, Patrick Bonnel, Olivier Brette, Lionel Brunie, Mathieu Cunche, Stephane D'Alu, Vincent Primault, Patrice Raveneau, et al. Priva'mov: Analysing human mobility through multi-sensor datasets. In *NetMob 2017*, 2017.

[34] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch library. https://pytorch.org/, 2016. Last accessed on May 12th, 2023.

[35] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. Crawdad data set epfl/mobility (v. 2009-02-24), 2009.

[36] Vincent Primault, Sonia Ben Mokhtar, Cédric Lauradoux, and Lionel Brunie. Differentially private location privacy in practice. *arXiv preprint arXiv:1410.7744*, 2014.

[37] Vincent Primault, Sonia Ben Mokhtar, Cédric Lauradoux, and Lionel Brunie. Time distortion anonymization for the publication of mobility data with high utility. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 539–546. IEEE, 2015.

[38] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[39] James Tamplin and Andrew Lee. Firebase services. https://firebase.google.com, 2012. Last accessed on May 12th, 2023.

[40] Timescale. Building a distributed time-series database on PostgreSQL, August 2019. Last accessed on May 12th 2023. URL: https://www.timescale.com/blog/building-a-distributed-time-series-database-on-postgresql/.

[41] Lin Wang, Hristijan Gjoreski, Mathias Ciliberto, Sami Mekki, Stefan Valentin, and Daniel Roggen. Enabling reproducible research in sensor-based transportation mode recognition with the sussex-huawei dataset. *IEEE Access*, 7:10870–10891, 2019.

[42] Kaihe Xu, Hao Yue, Linke Guo, Yuanxiong Guo, and Yuguang Fang. Privacy-preserving machine learning algorithms for big data systems. In *2015 IEEE 35th international conference on distributed computing systems*, pages 318–327. IEEE, 2015.

[43] Blaise Agüera y Arcas. Decentralized machine learning. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1–1. IEEE, 2018.

[44] Meng-Chieh Yu, Tong Yu, Shao-Chen Wang, Chih-Jen Lin, and Edward Y Chang. Big data small footprint: The design of a low-power classifier for detecting transportation modes. *Proceedings of the VLDB Endowment*, 7(13):1429–1440, 2014.

[45] Changqing Zhou, Dan Frankowski, Pamela Ludford, Shashi Shekhar, and Loren Terveen. Discovering personal gazetteers: an interactive clustering approach. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 266–273, 2004.

A Methodological Transparency & Reproducibility Appendix

A.1 Tools setup

Our benchmarking applications are developed with Flutter, an open-source framework by Google for building multi-platform applications (installation instructions: https://docs.flutter.dev/get-started/install); we used Flutter version 3.3.4: to ensure you are using the correct version, check the `flutter --version` command output:

```
user@computer:~$ flutter --version
Flutter 3.3.4 • channel stable • https://github.com/flutter/flutter.git
Framework • revision eb6d86ee27 • 2022-10-04 22:31:45 -0700
Engine • revision c08d7d5efc
Tools • Dart 2.18.2 • DevTools 2.15.0
```

To run the applications, you can use smartphones (*i.e.* real Android or iPhone devices) or emulators, though performances will be poorer with the latter. In both cases, you will need to install the Android Studio IDE: https://developer.android.com/studio.

If you own an Android smartphone, it can be used to test our applications: you need to plug it into your computer using USB and set up development mode: https://developer.android.com/studio/run/device. Otherwise, Android Studio contains the Android emulator component: if you need to set up one emulator, instructions can be found here: https://developer.android.com/studio/run/emulator.

In both situations, your device should appear in the `adb devices` command output:

```
user@computer:~$ adb devices
List of devices attached
13241JEC208547  device
```

A.2 Applications

Results and figures presented in this article were obtained using four experimental applications:

- Accelerometer [4]
- Benchmarking memory space [7]
- Benchmarking throughput [8]
- In-situ LPPM [6]

A.2.1 TemporalBDDFlutter

(installation time: 2 minutes, run experiment duration: 2 minutes)

The core library of our contribution, `temporalbddflutter` includes all classes used to model data; this package also includes a toy application modeling accelerometer data with FLI in real-time.

To run the experiment, click the "Launch XP: no movement" button (and do not move your phone if it is a physical device). This will start listening to your device's accelerometer and store its values in FLI models. The "no movement" part of this experiment shows that FLI saves memory space by modeling data instead of storing discrete records, providing an important space gain. You can also try the "move" experiment while moving your phone around: you will see that size gain is lower than the previous experiment, due to data randomness. This experiment has values count and time bounds to stop it automatically after 10k inserted values or 2 elapsed minutes, depending on which limit is reached first.

A.2.2 Benchmarking memory space

(installation time: 2 minutes; run experiment duration: 5 minutes)

This application allows comparing data sizes of random or constant values, using an SQLite database or a FLI model.

This application allows you to store 1 million values in the phone's memory, using either random or constant values, and either using SQLite or FLI; once an experiment is finished, you can read the size of the file in which values are stored. It demonstrates

that FLI modeling performances are approximately the same as SQLITE regarding random values, but are superior while storing constant values.

Results obtained by this benchmark are reported in Figure 8.

A.2.3 Benchmarking throughput

(installation time: 2 minutes; run experiment duration: 10 minutes)

This application allows comparing speeds of inserting or reading values, using FLI, SWAB or Greycat modeling.

Results obtained by this benchmark are reported in Figure 9.

A.2.4 In-situ LPPM

(installation time: 10 minutes; run experiment duration: 3 hours)

This application allows you to model location datasets when varying the tolerated errors and to execute the in-situ POI search on stored data.

As it uses existing datasets (namely CABSPOTTING and PRIVAMOV), you need to load those to your test device before running experiments:

1. Download datasets from https://drive.google.com/file/d/1Z1ccze31-uCfmiSouxp6SwhFzz5ihtfS
2. Extract them
3. Upload dataset files on your phone in the app directory using the file device explorer (documentation: https://developer.android.com/studio/debug/device-file-explorer) (app path is something alike `/data/data/package_name.in_situ_lppm/files`, you'll have to create a `datasets` folder there)
4. Run experiments

Since those experiments compute POIs on entire datasets, they take some time to run; runtime metrics we obtained from a Pixel 4a device (Android 13) are summarized in Table 4.

Experiments output results on the phone directly, in the directory `results/` (at the same place where you stored the datasets). Results are then used to plot Figures 11, 12 and 13.

Table 4: Computation times of LPPM experiments on a Pixel 4a using Android 13.

Experiment	Description	Elapsed time
CABSPOTTING: storage + POI + Promesse	Applies Promesse on each CABSPOTTING user, compares Promesse-computed POIs with raw POIs	3 min 35 s
CABSPOTTING: changing the error	Applies Promesse on each CABSPOTTING user with varying tolerated error (0.0001, 0.0005, 0.001, 0.002, 0.005, 0.01), compares Promesse-computed POIs with raw POIs	21 min 07 s
CABSPOTTING: using our new POI attack	Computes POIs with D&S attack for each CABSPOTTING user, compares results with raw POIs	3 min 55 s
Compute POI on PRIVAMOV user #1	Computes POIs with D&S attack for first PRIVAMOV's user, compares results with raw POIs	1 h 47 min
Load PRIVAMOV full dataset	Model all PRIVAMOV's users with FLI modelling	6 min 19 s