Acknowledgement

Funding from these generous sponsors made this thesis possible - DHS Award HSHQDC-16-3-00083, NSF CISE Expeditions Award CCF-1139158, DOE Award SN10040 DE-SC0012463, DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, IBM, SAP, The Thomas and Stacey Siebel Foundation, Apple Inc., Arimo, Blue Goji, Bosch, Cisco, Cray, Cloudera, Ericsson, Facebook, Fujitsu, HP, Huawei, Intel, Microsoft, Mitre, Pivotal, Samsung, Schlumberger, Splunk, State Farm and VMware, Alibaba, Ant Financial, CapitalOne, Futurewei, Nvidia and Scotiabank, the National Science Foundation under grant CPS-1239552, NSF CISE Expeditions Award CCF-1730628, the CONIX Research Center, one of six centers in JUMP, an SRC program sponsored by DARPA, CITRIS Seed Grant 2016-0072.

e-mission: an open source, extensible platform for human mobility systems

by

Kalyanaraman Shankari

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Sciences

and the Designated Emphasis

in

Global Metropolitan Studies

in the

Graduate Division

of the

University of California, Berkeley

Fall 2019

e-mission: an open source, extensible platform for human mobility systems

Abstract

e-mission: an open source, extensible platform for human mobility systems

by

Kalyanaraman Shankari
Doctor of Philosophy in Computer Sciences
and the Designated Emphasis in Global Metropolitan Studies

University of California, Berkeley
Professor David Culler and Professor Randy Katz, Co-chairs

Transportation is the single largest source of carbon emissions in the US. Decarbonizing it is challenging because it depends on individual behaviors, which in turn, depend on local land use planning. The interdisciplinary field of **Computational Mobility**, focusing on collecting, analysing and influencing human travel behavior, can frame solutions to this challenge.

Innovation flows in interdisciplinary fields are bi-directional. The flow to the domain is focused on building a strong foundation for methodological improvements. As the improvements are deployed, they result in use-inspired computational research. This temporal dependency results in our initial focus on the *modularity*, *accuracy* and *reproducibility* of **e-mission**, an extensible platform for instrumenting human mobility. This open source platform has a modular architecture that supports power efficient duty cycling using virtual sensors, a read-only data model and a pipeline with novel algorithm adaptations for smartphone sensing.

We also perform the first empirical evaluations of smartphone-based platforms in this domain. The *architectural* evaluation is based on three real world deployments: a classic travel diary, a crowdsourcing initiative, and a behavioral study. The *accuracy* evaluation is based on an novel procedure that uses artificial trips and multiple parallel phones to mitigate concerns over privacy, context sensitive power consumption and inherent sensing error. Data collected from three artifical timelines was used to evaluate the trajectory, segmentation and classification accuracies vs. power for various configurations.

On *computational* side, challenges derived from the deployments can contribute to ongoing CS research in privacy, trustworthiness, incentivization and decision making. On the *mobility* side, this enables methodological innovations such as Agile Urban Planning for prototyping infrastructure changes.

- To my parents, for the best start in life that any child could ask for, and for continuing to support me even when I have children of my own.
- To Tom, for supporting my crazy choice to return to school in middle age, for helping me frame my ideas, and for the ongoing commitment to our 50-50 agreement that made this PhD possible.
- To my kids, for the nights spent away from home, for the proof-reading, and for accepting a lifestyle that was different from your friends without complaint.
- And finally, to our glorious planet. I hope we can stabilize your climate, perhaps with some help from the work in this thesis, before it is too late.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to start by thanking my advisors, David Culler and Randy Katz. I came into the program with the unbaked, high level vision that access to fine grained travel data would help transportation sustainability. David and Randy helped me refine my vision, introduced me to collaborators and steered me away from potential pitfalls. They gave me the courage to go beyond my risk-averse immigrant choice of remunerative topics to work on and tackle an area that has the potential to make a broad impact in mitigating climate change. Without their ongoing support and advice, this thesis would have looked very different.

They were also full partners in navigating the post-PhD landscape for interdisciplinary research. Since interdisciplinary theses are still rare, there was no standard template to follow. They went above and beyond the normal recommendation letters to ensure that I found a home where I could continue to grow.

I would also like to thank the other members of my dissertation committee. Eric Paulos helped me understand the research techniques for UX work, and Paul Waddell introduced me to several mobility researchers, and his students were my co-authors for multiple papers.

I would also like to thank Harrison Liew and gennui raffill for their assistance in salvaging the data collection after the Lyft e-bike debacle. The data collection for the evaluation includes a one-way ride from the UC Berkeley campus to the Transbay bus stop on an e-bike. I used Lyft bikeshare e-bikes for the first round of data collection, but then a couple of e-bikes caught on fire, and Lyft pulled all e-bikes from service. I was now stuck since using a regular bike would preclude comparisons with the first round.

Fortunately, Harrison and gennui stepped in and helped out by lending an e-bike and riding the bike back to campus to complete the round trip. Neither rain nor heat nor lost phones or unexpected band practice stayed us from the slow collection of all four quadrants of the frequency/accuracy combinations. And Lyft had still not put e-bikes into service at the time of submission, so this thesis would not have been completed without their help.

I would like to thank my mentees. Since e-mission requires skills in so many areas, they filled in all the parts that were outside my expertise. Shanthi and Gautham built the first native UIs and helped me decide that we should use cordova instead. Josh and Naomi built the tour model, and Weijian reworked the UI and displayed it. Sunil and Juliana helped with the first real deployment — bic2cal. Yawen helped validate battery-based power drain as a technique for power evaluations. Jennifer and Felipe experimented with new features that will eventually make it into the core platform. The TripAware group — Jesse, Jack, John, Vas, Andy and Kyle — helped me figure out how to mentor student groups all the way through the research process. The GreenTrip group — Vanessa, Sam, Emma, Bill and Trevor — took on the challenging task of inferring trip destinations and built a base for a future study.

I would also like to thank my labmates at the SDB and RISE labs for being my guinea pigs, testing out e-mission and giving detailed feedback. Gabe, Micheal, Kaifei and Sam, in particular, were involved in multiple rounds of evaluation as e-mission evolved from a native-only app with an ugly UI to the current platform.

I would also like to thank the research staff in RISE and SDB for all their help with project logistics. In particular, I would like to thank Kattt for her help with getting the test phones, Shane for the automated testing, Jon for all the server redirects, and Boban for all the creative tinkering with printouts. And of course, Albert for printing posters at short notice, lending me servers for data collection, and for letting me in when I got locked out late at night.

I would also like to acknowledge the funding from our generous sponsors that made this thesis possible. The funding came from a variety of different sources over time and included:

AMP supported in part by DHS Award HSHQDC-16-3-00083, NSF CISE Expeditions Award CCF-1139158, DOE Award SN10040 DE-SC0012463, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, IBM, SAP, The Thomas and Stacey Siebel Foundation, Apple Inc., Arimo, Blue Goji, Bosch, Cisco, Cray, Cloudera, Ericsson, Facebook, Fujitsu, HP, Huawei, Intel, Microsoft, Mitre, Pivotal, Samsung, Schlumberger, Splunk, State Farm and VMware.

SDB supported by the National Science Foundation under grant CPS-1239552.

RISE In addition to NSF CISE Expeditions Award CCF-1730628, supported by gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

CONIX supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

CITRIS seed funding Grant 2016-0072: Putting the feedback cycle in high gear: community-sourced, data-driven approaches for sustainable transportation infrastructure

Chapter 1

Computational Mobility

Transportation has become the single largest source of greenhouse gas (GHG) emissions in the United States [UE]. Transportation decarbonization depends on individual choices, which are affected by local land use infrastructure and planning. Tackling this challenge requires an interdisciplinary approach. Truly interdisciplinary work can balance intellectual merit and broader impact, and lead to transformative research. However, it has a high startup cost, and needs researchers who are extremely strong in their chosen discipline, but have the flexibility to adapt to the needs of the other domain.

The interdisciplinary field of **Computational Mobility** (CM) can frame the interdisciplinary approach required for transportation decarbonization.

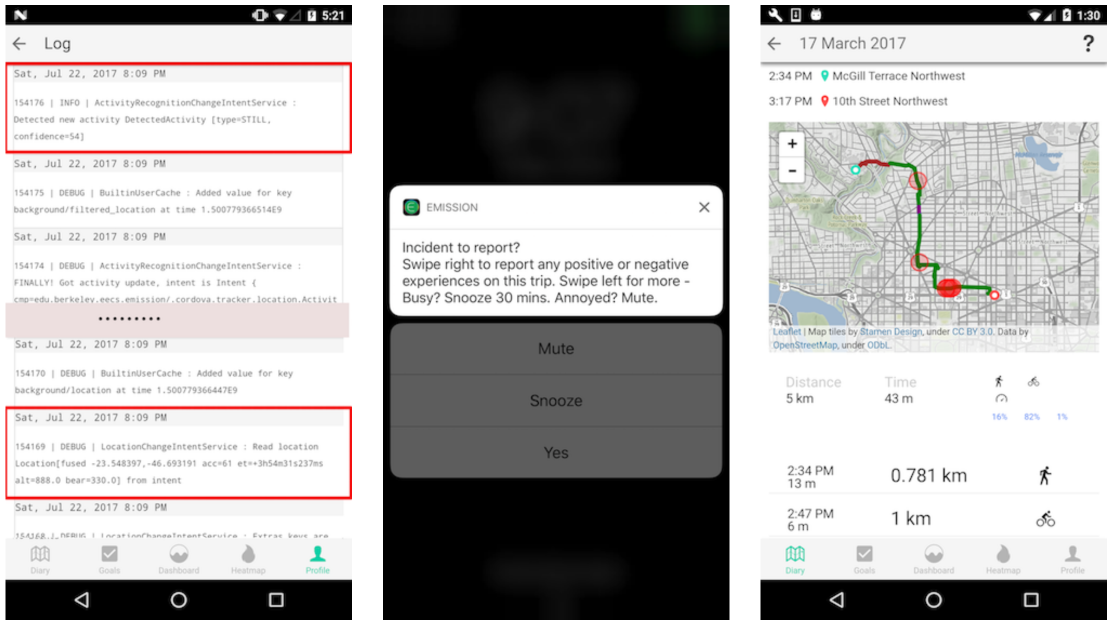
CM focuses on techniques to: (i) collect human mobility data, (ii) analyze it to generate models, and (iii) apply the models to study or influence behavior and plan infrastructure.

CM can provide insights that existing data sources cannot. For example, while considering the effect of ride hailing on congestion, separate ride hailing and traffic counts can show correlation but not congestion, cannot rule out alternate explanations or provide actionable information.

CM needs to include both ongoing, continuous sensed data, and intermittently triggered survey or qualitative data. This qualitative data can be user-generated, or provide stated preferences to planners. CM data is primarily used to generate individual *travel diaries*. However, in the aggregate, it can also provide infrastructure data and be used to generate geographically customized models.

This chapter explores the idea of CM in greater detail. It starts by outlining the data requirements, the models that might be generated, and the rigor required to fully understand them. It then introduces a set of research questions that are explored in the remainder of this thesis, and links their atypical nature to the demands of interdisciplinary work. Answers to the questions are embodied in **e-mission**, the open source, extensible platform which is the focus of this thesis.

1.1 Computational mobility definition



```
Sat, Jul 22, 2017 8:09 PM
154176 | INFO | ActivityRecognitionChangeIntentService :
Detected new activity DetectedActivity [type=STILL,
confidence=54]
Sat, Jul 22, 2017 8:09 PM
154175 | DEBUG | BuiltinUserCache : Added value for key
background/filtered_location at time 1.500779366514E9
Sat, Jul 22, 2017 8:09 PM
154174 | DEBUG | ActivityRecognitionChangeIntentService :
FINALLY! Got activity update, intent is Intent {
cmp=edu.berkeley.eecs.emission/.cordova.tracker.location.Activit
..........
Sat, Jul 22, 2017 8:09 PM
154170 | DEBUG | BuiltinUserCache : Added value for key
background/location at time 1.500779366447E9
Sat, Jul 22, 2017 8:09 PM
154169 | DEBUG | LocationChangeIntentService : Read location
Location[fused -23.548397,-46.693191 acc=61 et=+3h54m31s237ms
alt=888.0 bear=330.0] from intent
Sat, Jul 22, 2017 8:09 PM
154168 | DEBUG | LocationChangeIntentService : Extras keys are
```

Incident to report?
Swipe right to report any positive or negative experiences on this trip. Swipe left for more - Busy? Snooze 30 mins. Annoyed? Mute.

Figure 1.1. Overview of the basis for computational mobility. Left to right: sensed data, surveyed annotations, combined into a trip diary.

Computational Mobility is the sub-field of Computational Transportation Science (CTS) that focuses on human travel. In particular, the sub-field focuses on techniques to: (i) collect human mobility data, (ii) analyze it using data-driven, mathematical or computer simulation techniques, and (iii) apply the models to study or influence behavior and plan infrastructure.

Until autonomous vehicles are in widespread use, however, all transportation is related to human travel. Therefore, we start by defining the scope of Computational Mobility in terms of what it includes, what it excludes, and how it can integrate with other aspects of CTS.

Computational Mobility is typically based on (Figure 1.1): (i) end-to-end trip information linked to a single traveler, (ii) across all travel modes, (iii) annotated with qualitative details, and (iv) collated into a trip diary.

Most CTS projects tend to focus on sensed data, following a long computing tradition of sensor-based data gathering and analysis. However, because of its focus on human behavior, computational mobility needs to mix sensed data with surveyed data. Each of these methods have unique strengths: (i) sensed data can recall *precise* information, such as departure and arrival times, better than humans, but (ii) only humans can determine *behavioral* information such as trip purpose or *perceptual* information such as trip quality.

Figure 1.2. Sources that are not suitable for computational mobility. Left to right: traffic cameras (infrastructure)[4], bicycle counters (infrastructure)[5], OBD scanners (automobile-only)[6].

Computational Mobility does *not* typically include information from more traditional transportation data sources, for example (Figure 1.2):

(i) data that is linked to transportation *infrastructure* such as roads, generated using point sensors such as loop detectors or traffic cameras, or

(ii) data that is linked to *vehicles*, generated by private automobiles using sensors such as On Board Diagnostic (OBD) scanners or public transit vehicles using the on-board location trackers for real-time arrival times.

Although Computational Mobility does not directly use these traditional data sources, *integrating* mobility information with traditional data sources can enhance overall utility. Consider the classic problem of counts along a road. Infrastructure sensors have comprehensive coverage, but of one mode of transportation (automobiles). CM data covers all modes and include information on the origin (O) and destination (D) of the travel, but only for the subset of the population that participates in the tracking. Combining them could give a comprehensive picture of multi-modal, O-D travel across all modes.

1.2 The case for computational mobility

Recent years have seen a structural shift in the transportation landscape away from personal automobiles to active transportation (e.g., walking, biking), public transit (e.g., bus rapid transit, subway) and shared mobility (e.g., car sharing, ride hailing). This shift has already been shown [KAB$^+$12, SS13] to be: (i) *broad*, covering six industrialized countries including US, Japan and Germany; (ii) *deep*, including significantly lower rates of car licensure and ownership (e.g., from 75% to

[4]Image by PublicDomainPictures from Pixabay
[5]James Cridland[CC BY 2.0]
[6]Florian Schffer [CC BY-SA 3.0]

65% for the 21-29 age cohort in the UK); and (iii) *persistent*, spanning roughly two decades [KAB+12, p. 775].

This shift is expected to be transformative since it is supported by both push and pull factors. It is pushed by both economic considerations, and governmental policies that seek to achieve societal goals such as (i) improving air quality; (ii) combating climate change; and (iii) boosting health through active transportation. It is pulled by the advent of widespread mobile communication through smartphones that have enabled the introduction of new mobility services such as car sharing and ride hailing, and improved the experience of existing ones such as real-time transit alerts. In fact, recent plans for pilot deployments of autonomous vehicles in Boston [7] and Phoenix [8], and last-mile solutions such as podcars [9] and electrified personal mobility [10] suggest that the shift is likely to accelerate in the coming years, and result in a radically altered transportation landscape.

This shift implies that the newly popular transportation modes need to be given greater weight in transportation decisions by extending the associated models with newly relevant factors. Two short vignettes, from the SF Bay Area and New York, provide an intuition into how existing data collection methods and models need to be extended to account for this shift.

1.2.1 New data: Ride hailing and congestion in New York

In March 2017, the New York Times ran an article [FH17] asserting that ride-hailing services actually increased gridlock in New York City, with impacts that include lower societal efficiency, higher GHG emissions and lower safety. The article was based on a report [Sch17] that used data sources based on existing mobility monitoring methods to reach this conclusion.

A careful examination of the data sources used in the study (Table 1.1) indicates that: (i) most of the sources are vehicle-oriented; and (ii) the only human-oriented data source is close to 10 years old.

While the author of the report [Sch17] has done an admirable job given the data sources available to him, the analysis could be even stronger if mobility data were available. In particular, the report:

1. shows *correlation* between lower subway ridership, increased TNC mileage and travel delays in congested areas, but cannot show *causation* (subway $\rightarrow$ TNC shifts) because the only data available on individual travel patterns is from the HTS, which predates the adoption of TNCs[11];

[7] http://www.businessinsider.com/lyft-self-driving-car-pilot-nutonomy-boston-2017-6

[8] https://arstechnica.com/cars/2017/04/waymo-trials-free-self-driving-taxi-service-in-phoenix/

[9] https://mv-voice.com/news/2015/11/13/podcar-proponents-tout-futuristic-transit-vision

[10] http://www.topsecretev.com/electric-rideables-last-mile-solution-vs-first-mile-solution/

[11] A prior, City-led study from 2015 on TNC impact found that increased congestion was caused by freight, construction, pedestrians and tourism

Source	Type	Date
NYC trip logs	End-to-end trip information, but associated with a vehicle instead of the traveler	2014-15
Daytime speeds in Manhattan	Infrastructure sensors	2016
Subway and bus ridership	Traveler linked trip information if Metrocard is used, but no trip end information since no swipe out is required, and no linkage to trips on other modes	2016
Bike ridership	Infrastructure sensors and commute to work data	2015
Ferry ridership	Cannot be linked to a traveler because private ferries are not linked to MetroCard, and the Staten Island ferry is free	2016
NYMTC HTS	Multi-modal, end-to-end trips linked to a particular traveler	2010-11

Table 1.1. Data sources used in the ride-hailing study [Sch17].

2. does not adequately rule out alternate explanations, since bicycle ridership also increased in the same time frame, and a subway $\rightarrow$ bicycle shift is also plausible[12]; and

3. does not provide actionable information since more comprehensive perceptual data on improvements (distance to stop, wait time, mechanical breakdowns...) needed to restore subway ridership is lacking [13].

1.2.2 New models: Parking in the Bay Area (SRI)

In order to determine the number of parking spaces required for SRI International's new campus[14], city planners in 2013 needed to estimate the number of cars that would visit it. This estimate is typically generated by a model from the Institute of Transportation Engineers, which has historically been based on data from isolated, suburban sites in the United States and Canada [15]. The traditional model thus assumes that the only relevant transportation mode is the personal automobile, so it

[12]http://hubway.virot.me/. This is vehicle-oriented because it represents counts at selected intersections and does not include end-to-end or semantic data. Unfortunately, in contrast to the comprehensive TNC data, bicycle ridership is projected from commute to work data reported by the census, 4 automatic counters and 2 manual count locations

[13]The NYT article had several anecdotes for the transit $\rightarrow$ TNC shift, but anecdotal evidence is insufficient for prioritizing and fixing problems

[14]`www.sri.com`

[15]`http://www.ite.org/parkinggeneration/datacollection.asp`

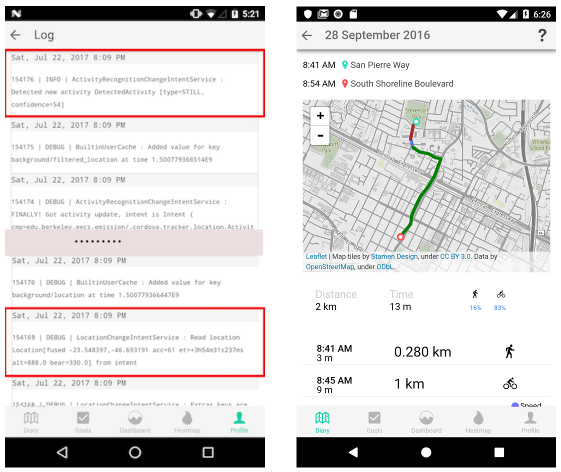
does not account for factors relevant to newly popular modes such as accessibility to frequent transit service, proximity to a bicycle network and new Transportation Demand Management programs that provide incentives for mode change. At SRI, these factors led to observed trip rates from a traffic study being *54%* lower than the rates predicted by the model[16].

1.3 What data matters

Since computational mobility is focused on understanding *human* travel, it must work on data linked to travelers, as opposed to data linked to *vehicles* or *infrastructure*. This data can be broadly characterized into *continuous* and *intermittently triggered* collection (Table 1.2).

1.3.1 Continuous sensed data collection

Continuously sensed data forms the basis of computational mobility. This large volume of continuously sensed data motivates the introduction of computational techniques into the understanding of mobility. The foundational continuously sensed data is spatiotemporal location traces. This data can be augmented with data from (i) other raw sensors, such as accelerometers or gyroscopes, or (ii) virtual sensors, such as the closed source motion activity sensors available on most mobile phone Operating Systems (OSes) used today. (Figure 1.3). The data is linked to a single traveler, and covers all travel, including active and multi-modal mobility.


```
Sat, Jul 22, 2017 8:09 PM
154176 | INFO | ActivityRecognitionChangeIntentService :
Detected new activity DetectedActivity [type=STILL,
confidence=54]
Sat, Jul 22, 2017 8:09 PM
154175 | DEBUG | BuiltinUserCache : Added value for key
background/filtered_location at time 1.500779366514E9
Sat, Jul 22, 2017 8:09 PM
154174 | DEBUG | ActivityRecognitionChangeIntentService :
FINALLY! Got activity update, intent is Intent {
cmp=edu.berkeley.eecs.emission/.cordova.tracker.location.Activit
..........
Sat, Jul 22, 2017 8:09 PM
154170 | DEBUG | BuiltinUserCache : Added value for key
background/location at time 1.500779366447E9
Sat, Jul 22, 2017 8:09 PM
154169 | DEBUG | LocationChangeIntentService : Read location
Location[fused -23.548397,-46.693191 acc=61 et=+3h54m31s237ms
alt=888.0 bear=330.0] from intent
Sat, Jul 22, 2017 8:09 PM
```

28 September 2016

8:41 AM San Pierre Way
8:54 AM South Shoreline Boulevard

Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

Distance	Time		
2 km	13 m	16%	83%

8:41 AM 3 m	0.280 km	
8:45 AM 9 m	1 km	

Figure 1.3. L: Spatio-temporal and activity data, R: across all modes.

1.3.2 Semantic labels for continuous data

[16]https://www.sri.com/sites/default/files/brochures/sri_master_plan_complete_part1.pdf, pages 39-40

Such semantic labels are typically provided by humans and represent the "behavioral explanatory variables" important for modeling [WBO+14]. For example, considering the complex tour in Figure 1.4, the sensor data alone does not indicate *why* the car mode was chosen. We could see the stop at Best Buy, and speculate that it was due to the bulkiness of return items, but it might be that the real reason was to transport a young child to the library. Without the appropriate semantic labels, we would not be able to appropriately model the user's behavior. While we cannot list every source of semantic data here, some popular ones [WBO+14] are trip modes, trip purpose and demographic information.

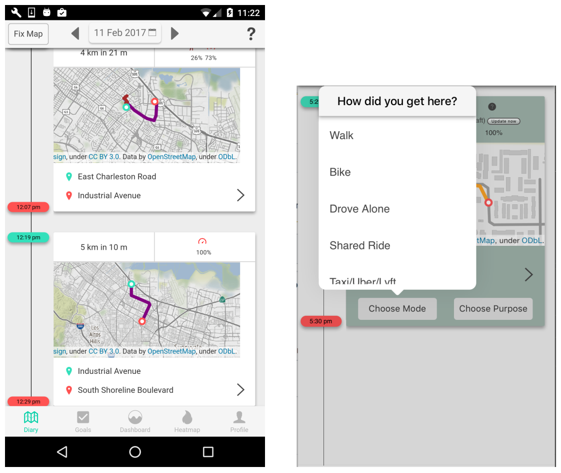


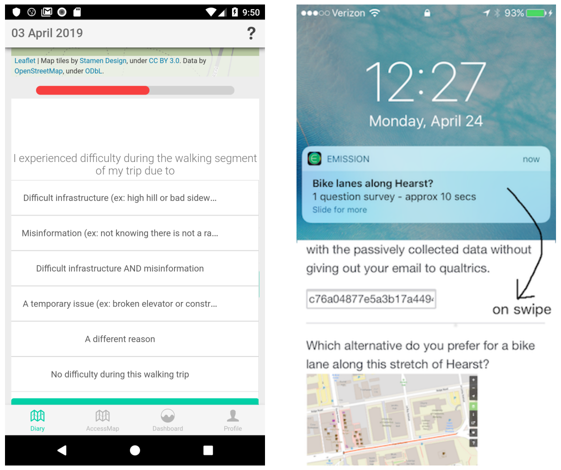
Figure 1.4. L: Complex tour involving a trip to the library and a stop at Best Buy; R: user confirmation of trip labels.

1.3.3 Intermittently triggered experience sensing

Intermittently triggered data collection is focused around capturing user information at specific points of interest.

Traveler-initiated, perceptual data allows us to capture incidents that affect the end-to-end *travel experience*. This data is of particular interest for understanding multi-modal travel, since it can capture the friction associated with bad sidewalks (Figure 1.5) or long waits for transit.

Planner-initiated, targeted surveys allow planners to obtain stated preferences for potential future scenarios, similar to the light rail extension and pricing policy options in Jerusalem studied in [OVW+11]. These could also be triggered for smaller-scale design decisions — e.g., "Would you start biking if we added a new bike lane to street X?".



I experienced difficulty during the walking segment of my trip due to

Difficult infrastructure (ex: high hill or bad sidew...

Misinformation (ex: not knowing there is not a ra...

Difficult infrastructure AND misinformation

A temporary issue (ex: broken elevator or constr...

A different reason

No difficulty during this walking trip

Bike lanes along Hearst?
1 question survey - approx 10 secs

with the passively collected data without giving out your email to qualtrics.

c76a04877e5a3b17a449

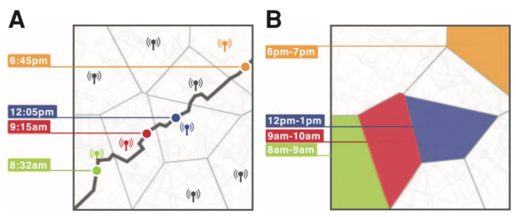
Which alternative do you prefer for a bike lane along this stretch of Hearst?

Figure 1.5. L: Traveler experience of sidewalk quality; R: planner initiated survey on proposed changes.

data collection types
- continuous sensed
  - spatio-temporal
  - motion activity
- continuous surveyed
  - trip modes
  - trip purpose
  - . . .
- intermittently triggered
  - traveler perceptions
  - planner-triggered surveys

data collection trade-offs
- sensor granularity
- sensor accuracy
- timeliness
- cost
  - operation
  - user attention
- interactivity

Table 1.2. Data characteristics represented as a tree. Roots are categories and leaves are the actual properties.



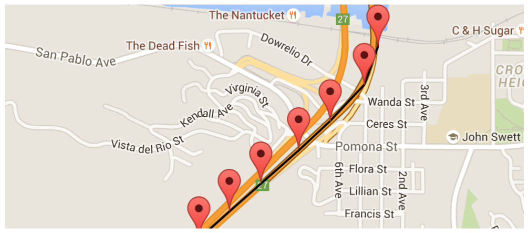


Figure 1.6. Granularity and accuracy L: CDR (from [dHVB13], A shows calls made by users that are received by cell towers, B shows how this cell tower data can be translated into coarse location polygons), R: smartphone app.

1.3.4 Data sources and quality

Since we want to capture all mobility traces associated with a single traveler, we need to use sensors that travelers carry around with them. Smartphones are the obvious choice for such data, but there are also other options such as dedicated wearables (e.g., smartwatches). And even if we focus only on smartphones, there are different possible trade-offs around cost, quality and completeness (Figure 1.2).

An example that illustrates this trade-off is the comparison between smartphone apps and cellphone call data records (CDRs) (Figure 1.6). Smartphone apps can collect very fine-grained sensor data with high accuracy, and augment it with user supplied information. But they also have a high burden on the user — not just the operational cost on the battery life but also the cost of seeking the users' attention to get the augmented data. Cellphone CDRs are generated from both smartphones and flip phones as part of regular operation, so they have no additional cost, but they are spatiotemporally coarse, have low accuracy and cannot capture semantic data.

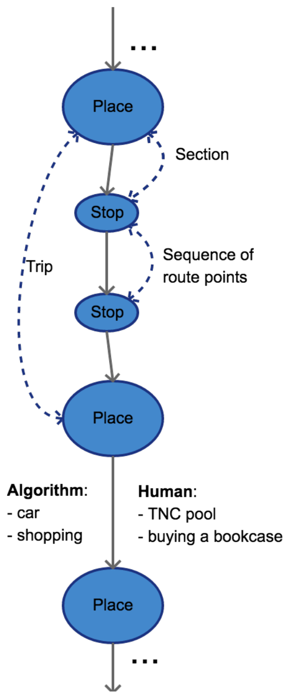
Similar trade-offs exist even if we only consider smartphone apps. Smartphone apps can control their sensing parameters, so they can target various points along the trade-off. For example, navigation apps have very high granularity and accuracy, but are typically plugged in to the automobile when running. Check-in apps such as Foursquare do not need to be plugged in, but that is because they have very low granularity, and obtain their high accuracy through significant user interaction.

1.4 How is the data modeled

This raw data can be analyzed and used to generate data-driven models. The models can be generated using mathematical or simulation techniques. They are key to converting the raw incoming data into a form that can be used to study or influence travel behavior. As with all models, these will not be perfectly accurate, but will have inherent error characteristics that depend on both the raw data and the analysis algorithms.

1.4.1 Travel diary

A travel diary is the canonical analysis result in Computational Mobility. It is also the building block for several of the other models, so should be considered a core component of computational mobility systems.

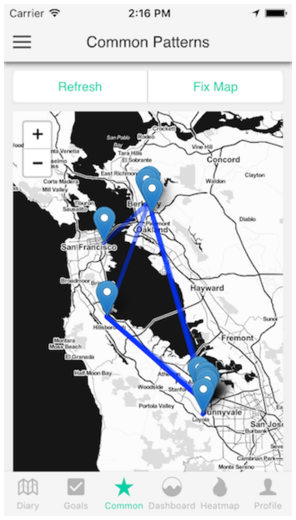


In its most complete form, a travel diary is a linked sequence of *trips* between *places*, each potentially split into *sections*. Each section is associated with a travel *mode* and each trip is associated with a travel *purpose* or *activity*. (We suggest using *purpose* to reduce ambiguity because *activity* can have other meanings in a travel context, as in [ZWHI15, BI04]). Travel *routes* are sequences of location points that can be associated with a particular section or trip, depending on the time range.

Since sensed data is better at detecting precise information such as trip start and end times, we expect that computational algorithms will perform trip and section segmentation. They can also perform automated mode and purpose detection. Depending on the accuracy of the algorithms and the granularity of the desired output classes (e.g., binary: walk/non-walk, or multiclass: car/carpool/ride-hail/motorcycle/...), the diary may be augmented with user inputs.

Note that the data quality (Section 1.3.4) has a direct impact on trip diary quality. A trip diary generated from CDR-granularity data is unlikely to detect short trips, or determine sections. But it can still detect long trips (e.g., commute) between places.

1.4.2 Personalized activity model



Once the travel diary has been constructed, it can also be used to generate personalized versions of classic behavioral models. For example, tours (chains of trips) are used as inputs to activity based models (ABMs) for forecasting travel behavior [Cas15]. A tour represents linked travel outside the home to perform *activities* (*purpose* in the trip diary above). The simplest tour model is home → work → home. More complex tours could include home → work → market → home. If we have longitudinal travel data for an individual traveler, we can build a data-driven, personalized activity based model that accounts for their complex travel patterns. The potentially complex tours in this model (e.g., home → sunday morning language classes → ethnic market → lunch with friends → home) can be used to forecast the traveler's individual travel behavior. This kind of model would help us better understand the ride-hailing example (Section 1.2.1) — we could see whether the trip mode for their tours had changed from subway to ride-hailing. These complex tours could also be used as input to regional-level ABMs for simulation-based travel forecasting.

1.4.3 Infrastructure models

The travel diaries can also be used to generate geographically customized, fine-grained infrastructure models. For example, traffic flow models are typically based on data collected from the Global North, and assume that the vehicles are mostly automobiles that obey lane discipline. They may not generalize well to conditions in the Global South, with large numbers of two-wheelers, bicycles and only a loose notion of lanes [MM16]. Fine-grained mobility data can be used to better understand traffic flow in a wide variety of conditions and build more sophisticated models. Similarly, in the SRI parking example (Section 1.2.2), local mobility data could have helped the city planners to understand how accessibility to transit modifies parking demand for large commercial sites in the SF Bay Area.

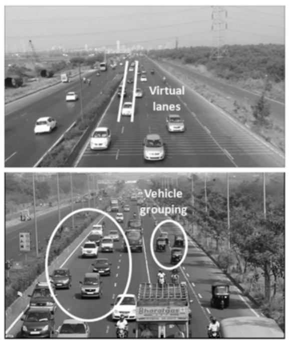


Figure 1.7. Driving patterns in the Global South [MM16].

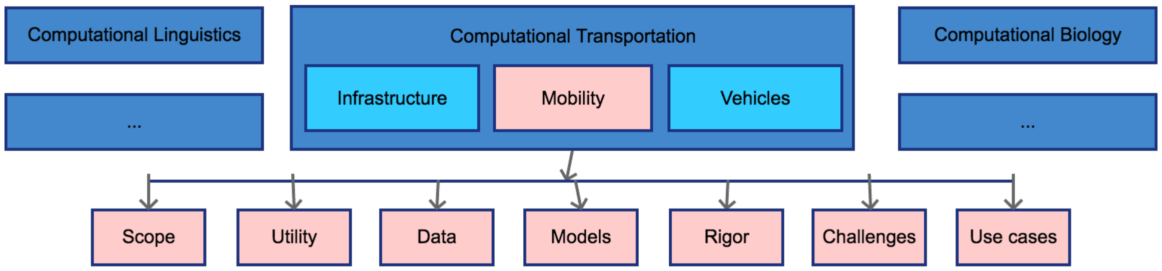


Figure 1.8. Computational Mobility in a broader context, and the associated topics.

1.5 What to expect in interdisciplinary work

Inter-disciplinary PhD theses are rare in Computer Science (CS), so this one asks and answers atypical questions (Section 1.7). This section lays the groundwork for the questions, and the rest of the thesis, through some perspectives on the nature of interdisciplinary CS. These ideas are consistent with prior framing [Com12] but observed rather than prescriptive.

Interdisciplinary CS is a difficult balancing act with a high startup cost. It must involve researchers who are extremely strong in their chosen discipline, so that they can advocate for the core of its ideas. At the same time, they must have the flexibility to understand the needs of the other domain and adapt to them. They must have the ability to see underlying patterns for generalization, but also translate the patterns to solve specific problems. Researchers from both sides need to build credibility and trust before they can collaborate.

If done right, it can also have great long-term impact. It can allow two distinct viewpoints to influence each other as equals and generate ideas that are fundamentally different from prior approaches. It can lead to transformative research that changes the way researchers think about problems in the field.

1.5.1 Interdisciplinary computational field examples

Computational Biology and Computational Linguistics are the most mature interdisciplinary fields, with dedicated departments at multiple universities and formal degree-granting programs. Newer fields include Computational Sustainability [GFF$^+$19] and Computational Transportation Science (CTS) [WSWG](Figure. 1.8).

Ten years after its inception, the progress on CTS has been directly proportional to its proximity to transportation engineering. Transportation involves *humans* and *vehicles* using *infrastructure* for *mobility*. There has been significant progress on CTS in the infrastructure (e.g., traffic cameras $\leftrightarrow$ computer vision) and vehicular (e.g., autonomous cars $\leftrightarrow$ real-time deep learning) sub-fields, but there has been little or no work on Computational Mobility.

1.5.2 Relation to purely applied work

There are three main distinctions between Computational Sciences and pure applied systems work. These distinctions can be seen in Intelligent Transportation Systems (ITS), which is the applied counterpart to CTS [WSWG, p. 2-3].

bi-directional The flow of information is not just from CS $\rightarrow$ the domain, but also from the domain $\rightarrow$ CS. Domain-specific use cases influence the state-of-the-art in Computer Science.

system rather than results The focus of the work is on the system rather than the results that the system enables. The system is evaluated rigorously, and is generalized so that it can be used to generate many different kinds of results.

developing new methods Instead of simply replicating existing techniques (e.g., "do X in software"), researchers develop new methods enabled by computation.

1.5.3 Broader Impact: CS $\rightarrow$ domain transfer

The research component of translation from CS to the domain typically involves creating a scientific framework for computation in the field. Instead of building custom, one-off solutions, researchers build a general foundation that can solve a family of problems. The innovations focus on generalization, extensibility, accuracy and reproducibility and include a strong engineering component. To encourage scientific rigor, researchers may need to evolve new metrics that can capture the extent of the innovations.

1.5.4 Intellectual Merit: CS $\leftarrow$ domain transfer

The research component of the translation from the domain to CS is typically dependent on sufficient adoption in the domain. As domain researchers explore the new techniques available to them, they may encounter limitations, or have new ideas on how to extend the foundation. If these gaps are sufficiently generalizable to similar domains, they can form the basis of CS research. This use-inspired fundamental research is likely to be highly relevant since it is derived from complex, real-world problems.

1.6 Computational Mobility is interdisciplinary

Interdisciplinary work has to meet a higher standard than the purely applied work involved in building a one-off system (Section 1.5.2). In this section, I outline how Computational Mobility (CM) meets that bar. It benefits from applying

Computer Science (CS) concepts around modularity and benchmarking to create generalizable systems with rigourous evaluation. It raises questions related to privacy, trustworthiness, incentivization and decision making that are relevant to Computer Science. And finally, it opens up new methods such as agile urban planning that can transform the way in which our cities are designed.

1.6.1 System rather than results

Computational Mobility introduces computational techniques to the mobility domain. These techniques are:

Modularity Highly modular systems can be generalized into a platform where the modules can be modified or configured independently. This relates to the core CS principles of abstraction and uses techniques from software engineering and software architecture.

Accuracy The overall accuracy for various points along the power/sensed accuracy/analysis tradeoff should be known. Since this involves assessing tradeoffs inherent in the design of a class of systems, it is related to techniques, such as synthetic benchmarks, for the empirical evaluation of computer systems.

Reproducibility In order to enable rapid iteration of analysis algorithms, the analysis pipeline should generate identical results when run multiple times on the same input data. This relates to core machine learning techniques for reproducible pipelines and read-only data.

1.6.2 Bi-directional

Computational Mobility introduces new CS challenges that span areas ranging from privacy and security to Persuasive Tech (Figure 1.3). Briefly, the new challenges are:

Privacy Travel diaries capture ongoing background travel data about an individual. This raises privacy concerns similar highly sensitive data (e.g., health information) and to background sensing, e.g., augmented reality or virtual reality systems.

Trustworthiness Collecting and analysing large quantities of *qualitative* data computationally raises questions about the truthfulness and consistency of user input. Similar issues arise in almost all online platforms with user-generated content such as social media posts or product reviews.

Challenge	CS domain	use case
Privacy for background sensing	IoT, privacy, security	Travel survey
Trustworthiness	Social networks, reputation, crowdsourcing	Crowdsourcing
Incentivization	Persuasive Tech (food/nudging, etc)	Behavior modification

Table 1.3. Mapping between challenges, CS domains, and use cases.

Incentivization Travel diaries can be used to change user behavior, but it is unclear which incentives work best. Similar issues rise in the field of persuasive technology, which a sub-field of Human Computer Interaction, and includes work on healthy eating habits, and the use of behavioral techniques (e.g., nudges) for social good.

1.6.3 Developing new methods

The traditional use of travel diaries is to enable Household Travel Surveys. However, high quality continuous travel data collection also enables *agile urban planning*, which prototypes changes in urban environments to quickly determine which are the most promising. Local governments that adopt agile urban planning practices can introduce a control feedback loop that helps them meet their sustainability goals. They can apply local travel patterns to existing models to develop some combination of infrastructure and incentive changes, and use the resulting shifts in travel patterns to propose new changes until their overall goals are reached. Once the system is in equilibrium, it can also sense disruptive external changes (e.g., advent of TNCs) and track ongoing compliance with goals (e.g., congestion levels).

1.7 The thesis problem

Innovations in interdisciplinary fields such as Computational Mobility (CM) can occur bidirectionally — from CS $\rightarrow$ domain and vice versa. Since the domain $\rightarrow$ CS changes depend on sufficient adoption, the CS $\rightarrow$ domain changes must be implemented first.

This thesis thus focuses on the CS $\rightarrow$ domain transfer. It builds a general foundation that can solve a family of problems. It includes a strong engineering component and focuses on establishing the scientific rigor needed to build general foundation for multiple CM projects.

Our questions evaluate the ability to apply computational concepts to the mobility domain.

Modularity How do we design a modular, extensible architecture for a **platform** that represents a family of systems for Computational Mobility? How do we know that this architecture meets the domain needs?

Accuracy How should we design and evaluate the continuous sensing given its context sensitivity? What are the power/accuracy/analysis **trade-offs**? What are the relevant metrics and how should they be computed?

Reproducibility What is the pipeline architecture that allows **reproducible** analysis? Given identical inputs, can it generate the same outputs on multiple runs?

The rest of this thesis explores these three questions in roughly the same order. The answers are also embodied in **e-mission**, an open source extensible platform for human travel data. e-mission has several real-world deployments and multiple features from external contributors.

The rest of this thesis is structured as follows.

Chapter 2 places this work in the context of the prior literature. It explores history of transportation data collection, starting from infrastructure data in the 1970s to vehicle data in the 1990s to CDR-based and smartphone app-based traveler data in the 2000s and outlines their strengths and weaknesses. It illustrates the builder–deployer gap in human mobility systems through a survey of one-off systems from practitioners, uni-platform pilots from builders, and prior attempts at building reusable platforms. It then outlines prior work in selected platform components including context sensitive sensing with virtual sensors and analysis algorithms for generating travel diaries. It also outlines the limitations of prior attempts at evaluating mobility data collection.

Chapter 3 answers the first research question by outlining a modular, extensible architecture for a human mobility system (HMS) platform. This is a traditional centralized, three-tier architecture comprised of client, server and analysis tiers. The standard architectural structure lowers the barriers to accessibility by deployers. The primary novelty lies in identifying the modules and data flows relevant for this domain. The platform extensions and customizations follow the principle of proportional effort. The client tier innovations include user interface channels and configurable cross-platform local event generation in the client tier. The analysis tier innovations include a pipeline and data model oriented towards reproducible analysis in the analysis tier. Small, medium and large customization and extension examples illustrate the principle of proportional effort.

Chapter 4 focuses on the design of the sensing infrastructure based on an initial round of data collection. The initial data collection from 2015 identified context sensitive sensing, virtual sensors, duty cycling and the challenges of restricted background processing as key concepts that the sensing design needed to tackle. The resulting sensing design uses virtual sensors for both location tracking and trip start/end detection. There are clearly defined Finite State Machines specific to both android and iOS, with both normal and error transitions between states. This also uses the semi-ad-hoc data collection to determine battery drain rates for various sensing regimes. These rates are extrapolated to a broad range of usage patterns using the American Time Use Survey (ATUS) in order to estimate the range of sensing battery drain at various settings.

Chapter 5 answers the third research question by focusing on the algorithms to convert the sensed data into the travel diary that is the basis of Computational Mobility. It starts with a quick summary of the reproducible pipeline architecture from Chapter 3, and the data generated by the virtual sensors in Chapter 4. It then outlines the various stages of the analysis pipeline, and the inputs and outputs for each stage. It then outlines novel aspects of adapting existing algorithms for use with smartphone-based virtual sensor data. This includes detecting untracked time, using motion activity virtual sensors for determining mode transitions, and handling zigzags generated by alternating underground and above ground segments. The outcome of these efforts is a travel diary that can be used for as the basis for further work in Computational Mobility.

Chapter 6 helps answer the second question by outlining a technique for reproducible geospatial evaluation. The technique adds rigor and reproducibility to the technique from Chapter 4. The modified technique adds predefined artificial trips for fine-grained ground truth and repeated experiments to account for transient variability. The chapter also lists the requirements for reproducible evaluation, and motivates them with examples from real data. It then shows how this procedure is both necessary and sufficient to meet the requirements.

Chapter 7 finishes answering the first and second questions. It first evaluates the ability of the platform to meet domain needs by comparing the usage of the architecture modules in the context of three use cases from different domains — (i) a classic travel diary, (ii) a crowdsourcing initiative for accessibility metrics, and (iii) a behavioral study on incentivizing sustainable transportation. The use cases contributed at least one extension, primarily client-related, back to the platform. Each of them used an average of 64% of the features of the platform, with $\approx$ 3-4 months of part-time CS undergraduate time for each new case. It then uses data collected from three different artificial timelines of different lengths to evaluate the power/accuracy/analysis tradeoff for a variety of configurations. The results, show that the determining factors on android and iOS are frequency and accuracy, respectively. These results can assist deployers in choosing the settings suitable for their application.

Chapter 8 concludes with a summary. It also expands on future work related to both CS $\rightarrow$ domain improvements such as benchmarking, plugin-based architecture and reinforcement learning, and CS $\leftarrow$ domain improvements to privacy, trustworthiness, incentivization and decision-making. It then introduces *Agile Urban Planning*, a methodological innovation in mobility enabled by these improvements. Finally, it takes a philosophical turn, discussing the potential for data misuse and some related technical solutions, before ending with speculation on extension to more general use.

Chapter 2

Background

Computational Mobility is the sub-field of Computational Transportation Science (CTS) that focuses on human travel. In particular, the sub-field focuses on techniques to: (i) collect human mobility data, (ii) analyse it using data-driven, mathematical or computer simulation techniques, and (iii) apply the models to study or influence behavior and plan infrastructure.

Human travel data has typically been collected using short-term *household travel surveys* and used to build travel behavior models. Continuous, long-term travel data has historically been generated from infrastructure and vehicle sensors. Replacing infrequent survey data with long-term continuous data collection allows transportation planners to get fresh behavior models. Less obviously, it also allows researchers to derive infrastructure and vehicle metrics that capture the complexity of the structural shifts in the transportation landscape (Section 1.2).

The rest of this chapter is structured as follows. First, we outline three kinds of transportation data — infrastructure-oriented, vehicle-oriented and human-oriented and show how human-oriented data can be used to derive the other two. We then compare three kinds of human-oriented data — navigation apps, cell tower data and smartphone apps, and discuss the trade-offs of each approach. This discussion indicates that, in spite of their limitations, smartphone apps are the most suitable source for the kinds of data that form the foundation of CM (Section 1.1).

Having settled on smartphone app-based data collection, we survey a cross-section of prior smartphone-based data collection systems and discover a *builder–deployer gap* that can be bridged through **modularity**. We examine prior **low power**, context sensitive sensing solutions, speculate that many of the proposed improvements have been incorporated into the phone OSes, and reiterate the importance of considering both sensing and CPU consumption. We survey well-known **analysis** algorithms and determine that they are mostly based on data collected from GPS devices and have not yet been adapted for use with smartphones. Finally, we elaborate on the lack of a suitable dataset for **evaluating** mode inference by identifying privacy, battery trade-offs and ground truth challenges in existing datasets.

2.1 Sources of transportation data

Transportation data collection can come from infrastructure, vehicles and humans. These roughly correspond to three waves of sensing technology and track advances in miniaturization, ubiquity, mobility and cost. Infrastructure sensors from the 70s could be mounted on vehicles in the 90s and be carried by travelers in the 2000s.

Infrastructure sensors focus on automatic sensing of traffic at fixed points. They were originally developed in the United States of America (USA) in the 1950s, so they have primarily focused on private automobile traffic. They can capture all vehicles passing through that point; but they cannot detect what happens before and after that point.

Vehicle sensors focus on the trajectories of vehicles while in motion. The most common sources are On-board Diagnostic (OBD) dongles that are powered by the vehicle's electrical system. As smartphone app-based mobility becomes popular, vehicle sensors provide additional sources of data and are expected to explode once V2V communication and autonomous vehicles become popular. Vehicular data can include end-to-end trajectories and support fine granularities, but they are not linked to each other to provide a complete picture of the travel.

Human sensors have historically not been possible due to the lack of portable battery powered sensors. The advent of smartphones has provided travelers the ability to capture fine-grained end-to-end travel across time. Smartphone travel data can be generated from cell towers, navigation apps and travel diaries. The data quality from travel diaries sensors is unparalleled, but it is only available from travelers who choose to capture it.

This thesis focuses on human-oriented data, captured using a configurable smartphone app-based app platform.

2.1.1 Infrastructure: Traffic Sensing at fixed points

Infrastructure-oriented mobility monitoring includes techniques to automatically instrument automobile traffic flow using continuously active freeway sensors, augmented by periodic roadway sensors (e.g., *rubber strips*) on city streets and arterials.

Since transportation data collection and modeling techniques [OW49] were pioneered in the United States in the 1950s, their focus was on understanding roadway congestion. Consequently, the first infrastructure sensors [Rob70] were developed for automobile detection. The collected data is traditionally used to optimize automobile traffic flow along roadway segments using (i) *metrics*, such as automobile Level of Service (LOS) to evaluate the performance of particular segments, combined with (ii) *models* of vehicle behavior, such as "weaving speeds on freeways" or "capacity vs. flow rate at roundabouts", to determine the impact of potential improvements.

There are three limitations of this traditional approach: (i) *collection is mostly* *periodic*, only with continuous monitoring restricted to freeways [1].; (ii) *what is measured is improved*, so improvements tend to be automobile-oriented [Vic17]; and there is (iii) *no end-to-end route*, so only segment-level optimizations are possible.

These limitations are only exacerbated by the shift away from personal automobiles towards a suite of mobility options, and the resulting increase in multi-modal, multi-segment transportation. The shift in mobility patterns makes it critical that we *broaden our measurement* to account for other modes and incorporate them fully in modeling and cost/benefit calculations. Further, the changing nature of these mobility patterns increases the importance of *end-to-end data* since the metrics are no longer linear (e.g., route quality is no longer simply the sum of the segment qualities because transfers can also play a large role)[2]

Recent work, fueled by the rapid adoption of information technology, has attempted to address some of the issues above. In particular:

1. Measurements such as *speed* and *travel time* that are feasible even with small numbers of probe vehicles can now be performed by GPS-enabled smartphones, including on freeways as part of the Mobile Century project [HWH+10], on arterials in Sweden [TMRR12], and using taxicabs in New York City [ZHUK13a].

2. Measurements such as *counts* and *travel classes* that need sufficient coverage to be effective, have tended to focus on *vision-based systems* using cameras, which provide 100% coverage within their field of vision [KB08], [Seg96] [SMP09], [ZSC13]).

3. Transportation guidelines have extending the automotive LOS metric to a *multi-modal LOS* by using route choice models [HSC11] to combine segment characteristics (e.g., number of lanes, width of sidewalk) with vehicle-oriented data (e.g., speeds, counts) [FDR+08].

However, even with these enhancements, the focus has remained on building segment-level models from fixed-point automobile data — the multimodal LOS explicitly touts that it can be computed using existing data[3], and there is no discussion of route-level or trip-level metrics. Although vision methods have significantly greater

[1] "Minimum 3-year count cycle", "Minimum 6-year count cycle","Where axle correction factors are needed to adjust raw counts, they should be derived from facility-specific vehicle classification data obtained on the same route or on a similar route with similar traffic in the same area." [FF13, p. 5-3]

[2] If the quality is assumed to be primarily influenced by speed/delay, as in automobile LOS, then the cumulative delay of a trip is the sum of the delays in individual segments. This means that "Segment performance can be aggregated to obtain an estimate of facility performance" [JMM08, p. 15-1]. But for multi-modal, multi-segment travel — e.g., walk $\rightarrow$ bus $\rightarrow$ bikeshare, the wait time for the bus will affect the overall delay, but will not be present in the aggregate of segment delays.

[3] "...combination of readily available data and data normally gathered by an agency to assess auto and transit level of service"

utility than pure fixed sensors, their coverage is also restricted because they can only instrument the segments that they are able to view. This implies that they cannot capture end-to-end trip-level metrics either. Therefore, the recent advances in the literature are not able to fully address the monitoring needs that were introduced by the shift in travel patterns.

2.1.2 Vehicle: Unlinked in-vehicle travel trajectories

Vehicle-oriented mobility monitoring includes techniques to automatically collect vehicular travel data such as speed, RPM, and temperature, combined with GPS-based location from personal automobiles or public transport vehicles.

As computer technology became smaller and more mobile, sensors could be incorporated directly into vehicles. This allowed manufactures to computerize many basic controls, and also allowed the sensor information used by the on-board computer to be exported for long-term analysis through mechanisms such as the On-Board Diagnostics (OBD) port [OnB]. While the evolutionary history of the OBD port is fascinating [HYT$^+$10], it is also beyond the scope of this thesis. The OBD port not only provides engine diagnostics [HYT$^+$10], but it can also power a GPS sensor included in the OBD port reader [LCSC09] or by interfacing with a smartphone over Bluetooth [ZCCM11].

Vehicular performance data has historically been used to optimize individual driver behavior — e.g., to detect accidents [ZCCM11] or reduce fuel consumption [MCCM15], either in consumer applications or as part of fleet management [MMN$^+$17].

There are three limitations of this approach: (i) it captures data only from personal automobiles, (ii) the trajectories are not linked, so it does not capture all travel over time, (iii) it covers sensed, not surveyed data.

Many of the modern mobility options have their own vehicular based data.

1. Public transit AVL data is used to estimate real-time arrivals [LZ99], and in the case of buses, potentially used for signal priority [HS]. Many modern AVL systems are GPS-based, although they can also use other sensors [HS]. Bus operators with limited resources such as small community bus operators [SMS13] or those operating in the Global South [LLYK16] use smartphones instead of a dedicated system.

2. Similar to low-resource transit operators, ride-hailing services such as TNCs and regular taxis collect vehicle trajectory information using smartphones. However, unlike the transit data, this data is not generally public [CK16, p. 177-178]. In some cities such as New York, taxicab origin-destination data is publicly available [Com] and forms the basis of multiple travel analyses [FPV$^+$13, ZHUK13b].

3. Vehicle to Vehicle (V2V) is likely to become standard in vehicles in the near future, similar to the prior introduction of the ODB-II standard [GR16, p. 3858].

> V2V systems, and by extension, future autonomous vehicles would exponentially increase the amount vehicular data available. However, V2V systems are explicitly prohibited from transmitting data that could link the vehicle to an individual [GR16, p. 3858] and it is reasonable to expect that autonomous vehicles will face similar restrictions.

This data source is remarkably complete if the traveler travels everywhere using only their personal automobile and does not wish to contribute non-sensed data. Some initial travel surveys in auto-dependent regions were actually vehicle-based [WGB01]. However, it is complicated by the shift away from personal automobiles towards a suite of mobility options. If the traveler switches between vehicles, the links between the trips in each vehicle are lost, and we no longer have a comprehensive record of their travel.

Thus, even with the rise of smartphone-based vehicular data, the focus of the vehicular monitoring technology remains the vehicle, and the limitation on linking trips to an individual traveler endures.

2.1.3 Human: Smartphone based, high quality, user consent

Smartphone-based mobility monitoring can generate the entire trip history for a particular user. It also has a granularity/coverage trade-off. Network providers collect large amounts of coarse data as part of providing their service. App-based data collection can be much more fine-grained, but users have to be convinced to install the app.

This thesis focuses on a platform for smartphone app-based travel diaries.

2.1.3.1 Cellular data

Cellular data from mobile phones is both broad and coarse. Cellular providers can automatically determine location information when the phone contacts a cell tower to place a call or send a text message. This implies that it has broad coverage, since every customer passively generates data while using their phone. However, it is also coarse, hard to validate and its lack of user consent raises ethical concerns.

Cellular data covers a significant percentage of the population. For example, the dataset used in [WHB$^+$12] covered $\approx 7 - 20\%$ of the population, and the dataset used in [HYL$^+$14] covered almost 50%. This broad reach and ubiquity have led to its use in determining driver sources for a road segment [WHB$^+$12], determining the life beat of a city [BCH$^+$11] and estimating public transit efficiency [HYL$^+$14].

However, the data is both spatially and temporally coarse. For example, for the dataset used in [dHVB13], the spatial accuracy ranged from $0.15km^2$ to $15km^2$ and the average inter-event time was 6 hours. For the dataset used in [HYL$^+$14], the spatial accuracy radius was $0.5km^2$ and the average inter-event time was 5 hours.

Since cellular providers collect the data, they control access to it. Users cannot opt-out of sharing their data. The data is also only available for limited use. For example, the durations for the datasets above ranged from 3 weeks to 2 months.

Finally, since there is no way to control the collection or to communicate with users, there is no direct validation. Prior work typically uses existing data sources — e.g., census [BCH$^+$11], household travel survey [HYL$^+$14] or probe vehicle GPS data [WHB$^+$12].

2.1.3.2 Navigation apps

Real-time navigation systems continuously track the users' trajectory with very high accuracy and granularity in order to provide a valuable service to travelers. If a traveler used navigation for every trip - e.g., drove everywhere, or turned on navigation for active transportation as well, then the data collected by the navigation system would be complete. However, if the traveler used unsupported modes such as public transportation, or skipped navigation for routine trips such as the grocery store, then the trip record would only contain partial data.

Further, most prior work on real-time navigation apps has been proprietary (i.e., Google Maps/Waze/INRIX), so the data is not easily available to researchers or planners [WBO$^+$14, p. 28].

2.1.3.3 Smartphone-based travel diaries

Smartphone-based travel diaries are focused around capturing a rich set of travel information about household travel. This includes end-to-end information such as start and end times, a rich set of transportation modes[4], and semantic information such as trip purpose. The traveler explicitly consents to the data collection by installing the app on their phone, and providing answers to survey questions.

This collected data is used to: (i) generate trip specific models (e.g., home $\rightarrow$ work, home $\rightarrow$ other, etc), and (ii) use those models to predict projected travel patterns at a regional level, both under current and future scenarios [Plu05, McN, MR].

This thesis focuses on this form of data collection, in line with prior work that selected GPS-assisted travel surveys as the most suitable source for travel demand modeling [WBO$^+$14, p. 45].

2.2 Prior HMSes from builders and deployers

Human mobility systems (HMSes) form the foundation for multiple different applications. The systems implementing these applications are all theoretically relevant

[4]The 2009 NHTS included support for 25 modes, including intercity buses, paratransit and neighborhood electric vehicles (NEVs)

to human mobility. A full listing of these systems is outside the scope of this thesis. Instead, we identify axes that define the HMS platform design space and select a sample of the related work that spans it. The related work includes examples of applications corresponding to the individual use cases for our platform (Table 2.1) and examples of ones that deal with survey data or sensed data or both (Table 2.2). In the rest of this section, we extract some patterns from these examples, delve deeper into differences from the most closely related platforms, and discuss the choice of projects and features for comparison.

2.2.1 Project and feature selection methodology

The methodology used to select projects and comparison features for the related work is designed to find a small, but representative spanning set.

Our use cases span popular application domains, so the related work is large. We picked a set of curated papers providing a flavor of the space, using the criteria of openness and novelty.

1. We chose systems from academia since they are more likely to be open, and discuss their architecture. This necessarily excluded proprietary projects such as rMove [FLHG17, GFHG16], Google Location History [5] or Strava [6].

2. We chose systems that were novel and varied from other systems in the same group in at least one feature. This avoided overwhelming the analysis with almost identical entries.

In order to quickly compare the projects in the related work to one other, we extracted very simple features that are relevant to the construction of systems and architectures for HMS. These features are:

sense: Indicates whether the project supports background sensing

survey: Indicates whether the project supports human-reported information using surveys

creator: Indicates whether the project was created by **B**uilders or **D**eployers,

architecture: Indicates the level of detail at which the architecture is described. At the highest level, it only shows the relationship between **T**iers, but it can also show the details for the **C**lient, **S**erver, **A**nalysis tiers,

OS: Indicates the phone OSes supported; **a**ndroid-only or **i**OS-only or **B**oth

open source: Indicates whether the project is open source and the code is actually accessible

[5]`https://support.google.com/accounts/answer/3118687`

[6]`https://strava.com`

Project	sense	survey	creator (B/D)	arch. (T/C/S/A)	OS (a/i/B)	open source	notes
Classic travel diaries							
FMS [CFF+13]	✓	✓	D	T	B	×	Data must be uploaded manually. Survey on website
SFTQS [CSW16]	✓	✓	D	×	a	×	Fairly complex app-based surveys for travel satisfaction. Requires surveys on 5 days of 6 week study. Based on ODK
DataMobile [PF16]	✓	×	D	T	B	✓	Only pre and post-study surveys are listed. System is open source, but only one application is described
Crowdsourcing applications							
Biketastic [RSD+10]	✓	*	B	T	a	×	sensed data used to derive traffic, and roughness. routes could be tagged with media. System was deployed but only for 12 users, so will categorize as created by builders
CycleTracks [HSC11]	✓	*	D	×	B	✓	open source single mode travel. Manual start/stop of trip. open source, extended by other MTAs, e.g. Atlanta to record infrastructure issues. Unclear if this is done through surveys [Poz13]
Tiramisu [ZTG+11]	✓	✓	B	T	i	×	single mode collection. manual start/stop of trip. provides a service (real-time bus and fullness information) to users.
Behavior change							
Matkahupi [JNS+13]	✓	×	B	×	a	×	allows users to set their own goals, and presents challenges based on travel patterns.
PEACOX [BPS+14, SPM+15]	✓	×	B	A	a	×	clear choice architecture with multiple theory-based approaches for persuasive change. provides service (trip planner).
QT [JAC+15]	✓	*	D	T	B	×	reports travel along cost, CO_2, time. Correction of automatically sensed mode by logging in to a website. No other ongoing survey information.

Table 2.1. Related applications, grouped along multiple axes. All the applications are published as standalone systems. Explanations: (i) * in a column implies that the answer is not clear, details are in the notes, (ii) column descriptions, including the abbreviations, are at Section 2.2.1

Project	sense	survey	creator (B/D)	arch. (T/C/S/A)	OS (a/i/B)	open source	notes
			Survey-only **or** sensing-only platforms				
Sensr [KMP13]	×	✓	B	T	i	×	allows authoring of web-based survey tools; each survey response can be tagged with locations/photo/text, but no background sensing
ODK [HLA+10, BSD+13]	×	✓	B + D	T*	a	✓	survey responses can include sensed data; both single locations and tracks, manually triggered; SFTQS app extended it for background tracking. ODK 2.0 architecture is much more complete
DIMMER [KJP15]	✓	×	B	S	*	*	platform proposes micro-services architecture for the server, unlike SOA of prior work; no details on "mobile applications"; funded by EU SMARTCITIES project, but no claim to open source
BOSS [DKT+]	✓	×	B	C + S	*	✓	services for smart building applications. only mobile component is web interface that supports personalized climate control. analysis (model training) is assumed to be part of the application
			Survey + Sensing platforms				
AndWellness [HRK+10]	✓	✓	B	C + S	a	*	similar to ODK but incorporates continuous location and activity sensing; shared server; architecture does not include analysis; extensibility is in future work; oriented toward ESM; visualization is standard, gamification would be hard; deployment options unclear
ohmage [TKK+15, HSE+13]	✓	✓	B	C + S + A	B	✓	from the same group at UCLA as AndWellness; follow-up project?; client architecture is scattered; most projects are survey-based; app was extended for PREEMPT, but unsure how much effort needed; analysis module was used for one project and hasn't been updated since 2014
ParticipACT [CCCF14, CCC+13]	✓	✓	B	C + S	a	*	Client is open source; server is not. client architecture is extremely detailed but android-specific; deployment only reported on pre-installed phones; extension by other groups unclear
Vita [HLD+13, HCCL13]	✓	✓	B	C + S	a	*	Extremely detailed SOA for client and server in mobile crowdsourcing; Smart City applications developed by research team; unsure if ever deployed; "open source mobile CPS", code location unknown

Table 2.2. Related platforms, grouped along multiple axes. Explanations are at Section 2.2.1

2.2.2 Characteristics of builder and deployer projects

The chosen one-off systems (Table 2.1) are from the travel diary, crowdsourcing and behavior change domains. The chosen platforms (Table 2.2) can be either sensing only, or survey only, or support both forms of data. They are all evaluated according to the metrics related to HMS system construction.

The prior work on one-off systems (Table 2.1) makes it clear that deployers have constraints that builders are often able to ignore. One obvious example is the set of mobile OSes supported — deployers almost always support both android and iOS because they care about coverage and representativeness. The one deployer project (SFTQS) that was android-only included an explicit argument that the bias in its data was small. But this constraint also restricts deployer effort to a fairly small set of use cases — most deployer effort is concentrated on the basic travel diary creation. Builders have the luxury of experimenting with new and innovative use cases, but typically stop with a proof of concept on either android or iOS. Further, most systems, even by builders, are one-off projects and are not open source. The CycleTracks app suggests that deployers do reuse open source projects if they meet a significant need.

Most prior platforms (Table 2.2) were developed by builders, as expected. However, few appear to address large-scale deployment concerns. In particular, except for ohmage, they only support either android or iOS — mostly android — which severely limits their use in deployer applications. Platforms tend to be open source much more often than applications, which is expected, since writing extensible software without making it open source requires sophisticated interface design. However, the details are complicated — sometimes, part of the platform is not open source (ParticipAct), or the code is not linked anywhere (Vita).

2.2.3 Comparing open source, reusable systems or platforms

The only other open source, **deployer-created** platform is the most recent version of DataMobile [PFJM19]. Similar **builder-created** platforms are ohmage + lifestreams [TKK+15, HSE+13], and to a lesser extent,
ParticipAct + MSF/MoST [CCCF14, CCC+13, CCC+14b]. We outline some limitations of these platforms below. We have had discussions with the DataMobile team about merging platforms in the future.

2.2.3.1 Deployer created

The deployer-created DataMobile [PFJM19] platform focuses on ease of use, both for participants and developers. It supports both android and iOS and makes the apps available in the stores for ease of installation. It supports deployer-created surveys which are made available through a dashboard. However, it appears to be more of a reusable system than a platform. Its feature set is limited, its architecture is unclear, and it does not appear to have been used for any functionality other than a travel survey. To expand further:

Limited functionality The platform appears to support only unidirectional sensed data from the phone to the server. The data pushed from the server to the phone corresponds to survey prompts. There does not appear to be an analysis pipeline which performs near real-time analysis so the question of reproducibility does not arise.

Architecture The architecture outlined in the paper focuses on the server, and primarily outlines creation and retrieval operations. There is a description of the phone duty cycling *algorithm*, but no idea of the sensing architecture or its extensibility.

Other uses The platform has been used as part of the Canada Food Study, but as part of a travel subsample. The extensions and effort required are unclear. There is no evidence that anybody outside the DataMobile team has used the platform or contributed changes.

DataMobile appears to be at the edge between a configurable system and a platform. It has already improved re-use, adding support for deployer configurable servers between 2016 [PF16], and 2019. Merging the platforms can benefit both parties by improving DataMobile architecture and e-mission ease-of-use in parallel.

2.2.3.2 Builder created

Builder-created platforms were much better architected. ohmage includes a simplified sensing architecture, lifestreams includes a clear analysis architecture, and MSF includes a sensing architecture with clear pathways to extensibility. However, this clarity is limited to one tier each, MSF only supports android, and the app installation appears to be clunky.

No true multi-method Although it supports both sensing and surveys, ohmage is primarily survey focused. Two of their studies (Mobilize, PREEMPT) are purely survey-based. It also does not appear to support combined passive sensing and self-reporting — the third study (moms) involved applicants using two separate apps, for self-reporting (survey) and mobility (sensing). In contrast, MSF, ParticipAct's sensing architecture paper [CCC$^+$13] is focused on passive sensing, with a clear event architecture for combining various sensors, and for sensor based survey triggers. Unfortunately, it works only on android and does not address the limitations on background processing in iOS (**Location State Machine** in Section 4.1).

Android+iOS support? While self-reporting apps from ohmage are available for both android and iOS, it is not clear that the passive sensing ones are. Passive sensing frequencies are listed at 1 minute or 5 minutes ([TKK$^+$15], Section 3.2.2); iOS does not allow time-based configuration of sensor frequencies. MSF works only on android.

Unclear analysis architecture In ParticipAct, the server and analysis architecture is unclear. How is the data analyzed? Is there a pipeline? How can others reproduce the analysis? How can they extend it? This obscurity extends to the actual source code. Although ParticipAct is open source, their server code is only available "upon request,".

The ohmage analysis architecture is remarkably similar to E-MISSION's (Section 3.6) which provides additional validation for our design choices. It is more feature rich in terms of change detection and prediction, but it is unclear whether the design supports reproducibility. In particular, it does not appear that any of the ohmage studies collected data on their own server instances or ran the analysis on their own data.

App installation For ohmage, in two of their three studies, participants were provided with phones with the app preinstalled. In the third (Mobilize, 2013), all participants were also developers, so it is unclear how representative the deployment process was. For ParticipAct, all participants were provided with smartphones, presumably with the app preinstalled.

2.3 Context sensitive smartphone sensing

Context sensitive sensing is a well-known technique to reduce the power consumption of continuous background sensing by (i) automatically lowering the accuracy, (ii) automatically lowering the frequency, or (iii) augmenting information from low power sensors with additional analysis.

The academic literature outlines several context-sensitive sensing techniques for smartphones. However, they are typically evaluated in isolation and work at the kernel, rather than user level. Smartphone OSes have likely incorporated several of these enhancements but access to them is restricted at the app level. Instead, they are exposed to apps as *virtual sensors* with proprietary implementations and unclear evaluations. The e-mission sensing implementation makes heavy use of virtual sensors. In particular, it turns off all sensing when not in motion and uses trip start virtual sensors such as geofences to restart tracking. In this section, we discuss context sensitive sensing projects from the literature and relate them to virtual sensors and other built-in optimizations.

2.3.1 Academic Literature: Location

The most relevant theme deals with lowering the power of location tracking.

Paek [PKG10] and Entracked [KLGT09] assume that the tracking will be continuous, and provide strategies to turn off the GPS intermittently for short periods of time during a trip. Paek [PKG10] uses the requested accuracy and Entracked [KLGT09] uses the user's activity.

Phone OSes already implement a variation of this approach by adaptively varying the sensing frequency of the location virtual sensor. The app-visible impact is that any frequency configuration is a *hint* which may not be honored perfectly by the phone sensing module (Figure 5.2).

In order to cooperate with the restrictions on background operation in the OS, we explore the ability to stop tracking for large periods of time, perhaps for the majority of the day. In this context, we are closer to the manually launched tracking solutions such as CycleTracks [HSC11] or Biketastic [RSD+10], except that in our case, the launching is automated.

The functionality from Bareth [BK11], which determines location using sources other than GPS, appears to have been incorporated into mobile OSes, and provides the basis for the fused API. The data collected from Android using the `batterystat` API indicates that even while using high accuracy tracking, the GPS is rarely turned on. But as we can see from Figure 4.1, there is still a trade-off between accuracy and power drain, and the power drain of the high accuracy mode is significantly higher than the medium accuracy mode.

The TAMER project [MCF15] appears to be the academic precursor to doze mode — it automatically interposes itself between the applications and the OS in order to reduce the frequency of background tasks.

2.3.2 Academic Literature: Activity Detection

The activity detection literature has papers ([YSC+12] and [SP12]) on duty cycling for energy efficient sensing, but for ongoing activity recognition instead of location detection. They, particularly Srinivasan [SP12], also point out that a significant proportion of the power drain in continuous sensing is not the power drain of the sensor, but the power consumed by waking up the CPU to compute with the sensed data. These insights provided the motivation for us to completely turn off location sensing when not in motion, and use the built-in trip start virtual sensors to retrigger it.

Chu [CLL+11] and ACE [Nat12] implement on-phone classifiers on Windows Phones that use context-sensitivity to lower the power drain. Their applications include activity detection virtual sensors which can be used for triggering rules, such as physical activity reminders. [CCC+14b] implement a real-time activity recognition API on the phone using their android-based sensing architecture MoST and expose it for app usage. Both android and iOS now include proprietary virtual sensors for phone-based activity detection.

2.3.3 Industry

None of these provide any evaluation or implementation details, so they are listed here for completeness.

Google location history [7] is included by default on all Android phones. In 2015, data collection was opt-out, in 2019 it is opt-in. In 2015, it appeared to read the location every minute using medium accuracy. It focused on place, rather than trip detection, although in subsequent years, it was combined with activity recognition to display trips.

Moves [8] was a fitness tracker app for both Android and iOS that was acquired by Facebook in 2014 and shut down in 2018. In earlier work, we had integrated with Moves for data collection instead of writing our own [SYCK15]. Our result was that out of 44 users who installed moves, only 8 retained it for more than 3 months, and they were all Android users. All iOS users uninstalled as soon as the semester was complete. The top three reviews in the app store complain about battery life being impacted. The inability to understand their techniques and to modify them was part of the motivation around designing our own data collection system.

2.4 Analysis pipeline and algorithms

The raw sensor data needs to be processed into a meaningful trip diary through post-processing algorithms. There is a well-established literature on post-processing algorithms from the travel survey world. The work is mature enough to be included in a survey and comparative analysis (Chapter 3., [Wol14]).

However, all these algorithms were developed for use with dedicated GPS, or GPS + accelerometer devices. This implies that (i) power was not a very critical concern, (ii) they had access to under-the-hood information such as the number of satellites for a fix, and (iii) their data collection was always on at a fixed time frequency.

The peer reviewed literature has not generated similar algorithms for smartphone data. I suspect this is due to the unpredictability of *fused* location data [Zan09]. This unpredictability persists when GPS-level accuracy is specified [BCN13]. There has also been a trend towards proprietary system stovepipes which integrate the data collection and processing, so the resulting algorithms are proprietary and are not described in detail (e.g., [ZGP$^+$15] from the FMS project).

[7]https://support.google.com/accounts/answer/3118687

[8]https://newsroom.fb.com/news/2018/07/hello-tbh-moving-on/

Pipeline Step	Source	Limitation
Trajectory smoothing	[SJF05]	Needs satellite and HDOP data
	[CCH10]	Needs satellite and HDOP data
	[NK08]	Does not have under-the-hood data, uses domain knowledge of altitude ranges in Switzerland as a filter
Trip segmentation	[WGB01]	Data collected from vehicle-based GPS data logger. Does not fully invalidate the simple dwell-based algorithm but does not capture the same complexity
	[NK08]	Uses changes in density; will not work with duty cycling because there are gaps in data collection
	[NK08]	Does not have under-the-hood data, uses domain knowledge of altitude ranges in Switzerland as a filter
Section segmentation	[TS72, NK08]	Assumes that we can detect 60s walk segments at the beginning and end of a trip. This works for high frequency (1sec) data collection but is challenging if lower frequencies (e.g., 30 sec) are used to lower the power drain. It is also unclear if it supports detecting transfers between transit modes[9]
	[OVW+11]	Based on [TS72] but uses a 44sec moving window. Thus it has the same limitations wrt lower frequencies

Table 2.3. Algorithms selected for evaluation by prior summary report (Tables 3-3, 3-4 and 3-5 in [Wol14], along with their limitations)

Modern smartphone OSes provide access to location, not GPS data. They also augment raw location data with proprietary *virtual sensors*. Other variations include (i) frequency "hints" rather than specifications, or distance filters, (ii) duty cycling for lowering power drain, so no continuous collection, and (iii) opaque APIs that restrict information about satellites or even GSM towers

Other sections of the thesis discuss the data characteristics in detail (Section 7.2) and outline our modifications to address them. This section outlines the shortcomings of a selection of current algorithms given the known limitations on modern smartphone data. I focus on the algorithms selected for the existing survey [Wol14](Table 2.3). I also augment this survey with more recent work using smartphones.

2.4.1 Trajectory smoothing algorithms

Smartphones report location, not GPS data. Unlike standalone GPS devices, they use assisted GPS or (A-GPS) [ZB11] which uses cell towers to narrow the GPS search space. They can also be combined with other location sources such as Wi-Fi and cell towers [Zan09]. They can also be sparser than dedicated GPS devices, due to built-in or intentional frequency lowering [BCN13]. However, trajectory smoothing algorithms have not been adapted to these conditions.

2.4.2 Trip segmentation algorithms

The chosen algorithm in this thesis is a dwell-time algorithm, similar to the selected algorithms (Table 2.3). An alternative density based algorithm assumes always-on sensing with a fixed interval [HHE13]. A dwell-time based algorithm modified for smartphones [ZGP^{+}15] addresses challenges introduced by fusing different location sources (e.g., GPS, Wi-Fi, GSM), but location fusion is (i) already implemented in the smartphone OS, (ii) no longer implementable at the app level, at least in iOS[10]. [SAMF16a] combines trip and section segmentation and introduces a two-stage approach which merges similar consecutive trips until the number of trips and sections stabilizes.

2.4.3 Section segmentation algorithms

While GPS device based algorithms use only location information for section segmentation, the built-in accelerometers in smartphones can provide valuable additional information with low power. Recent papers have developed solutions for accelerometer based distinction of active modes only [YYW^{+}14], or a richer set of modes [HNT13][11], [FLF^{+}16][12]. The smartphone OSes have incorporated some of these techniques as virtual activity sensors, but their accuracy can stand to be improved by post-processing [ZWHI15]. All this work is for android and does not address iOS.

2.5 Evaluation approaches for smartphone sensing

Human Mobility Systems (HMSes) are complex to evaluate (Section 6.2). Other researchers have identified similar challenges as part of survey papers (e.g., phone context, privacy, learning, scaling [LML^{+}10], ground truth [SSLA15], and varying

[10]https://stackoverflow.com/questions/31335481/how-can-i-get-lac-cellid-and-all-of-these-device-information

[11]Stationary, Walk, Bus, Train, Metro, Tram

[12]Still, Walk, Run, Bike, HSR, Metro, Bus, Car, Train

metrics and time scales across research areas [PGS17]). However, to the best of our knowledge, no solution has been proposed.

Papers related to instrumenting travel behavior fall into six main categories; we list some work from each as an example. A comprehensive classification of papers into categories is beyond the scope of this thesis. We end with an extremely recent work in a transportation thesis from 2019 that closely parallels our approach.

2.5.1 Context sensitive algorithms: power without accuracy

This research area focuses on context sensitive, adaptive power management of sensors. Papers such as ACE [Nat12] and Jigsaw [LYL$^+$10] compare their power requirements to naive sensing techniques. However, their accuracy evaluations focus on the localization error (Jigsaw), or comparison to naive inferred results[13] (ACE).

2.5.2 Travel diary systems: compare to manual surveys

There is a vast variety of one-off travel diary systems that combine smartphone based sensing with cloud-based processing to generate travel diaries. Systems such as Data Mobile [PF16], Future Mobility Study (FMS) [CFF$^+$13, HLZ$^+$16] and rMove [GFHG16] aim to replace the paper and telephone based Household Travel Surveys with smartphone and cloud based systems. So they evaluate the accuracy of their systems against the traditional methods, not against ground truth. This can show that smartphone based methods are significantly better than traditional methods, but does not provide a quantitative estimate of the accuracy of their system. Similarly, they do not include quantitative power evaluations — preferring statements like "Among the three types of discrepancies, the second type, data gap due to battery drainage, was most frequently observed." [HLZ$^+$16] or "The battery consumption test was simply whether, under regular usage, the phone could make it through the day without having to be charged." [PF16]. So they do not rigorously evaluate either the power or the accuracy side of the trade-off.

2.5.3 Large scale testbeds: no ground truth, no power

Large scale testbeds collect naturalistic data from a large population of end-users for a significant time period. In order to get the maximum return on the recruitment effort, testbeds such as MDC [LGA$^+$12] ($\approx$ 200 users, $\approx$ one year) and RealityMining [ESP06] (100 users, 9 months) typically collect data from a wide assortment of sensors that can support an variety of future research. However, due to respondent burden, and because they cannot predict the potential future uses for the data, they typically do not collect ground truth. Due to the wide variety of sensors included, it

[13]e.g., based on speed

is also impossible to isolate the power demands of the sensor subset that is relevant to a particular analysis. So they cannot be used to evaluate either the power or the accuracy side of the trade-off.

2.5.4 Mode inference: accuracy without power

Mode inference of travel mode based on sensor data is an extremely popular subject in the literature[14]. Researchers have used decision trees [RMB+10, ZCL+10], Hidden Markov Models [ZWHI15, SWLN14], and neural networks [FZP15, GWB+10] to distinguish between various subsets of travel modes. However, although the inference algorithms are different, most such papers use similar methods for evaluating their accuracy. They typically recruit a small sample of their friends (e.g., 16 users over one day [RMB+10], 4 users over two weeks [ZWHI15]) to collect naturalistic data along with annotations of the ground truth. The data collection focuses on the sensors used for analysis and omits the battery. This kind of evaluation does not meet the any of the requirements outlined above, except privacy, which is addressed by not publishing the dataset.

2.5.5 Inference on ad hoc datasets: no privacy

There are some mode inference papers that overcome the limitations in reproducibility, representativeness and robustness by relying on large-scale, ground-truthed datasets. The most commonly used dataset is GeoLife [ZXM10] which provides ground-truthed GPS trajectories from 182 users collected from April 2007 to August 2012. However, as conversations about data collection and privacy enter the mainstream, it becomes increasingly less likely that such datasets will be collected in the future. Recent large-scale testbeds [JBR+16] do not publish their data, and even some previously published datasets (e.g., the MDC [LGA+12], HTC [YYW+14][15] datasets) seem to be unavailable. The GeoLife dataset also highlights the challenge of evaluating static datasets generated from rapidly changing data collection methods. It was collected using dedicated GPS devices recording data every two seconds [ZCL+10]. It cannot be extended to explore the effect of combining GPS and Wi-Fi/cell tower data for lower power geolocation or combining GPS and accelerometer data for better segmentation and mode inference.

[14]Probably because it is hard, and nobody has really solved it well yet for modes other than walking

[15]used by [FLF+16], but now https://code.google.com/archive/p/transportation-mode-detection/, which contained "transportation log and parser" is archived and all links are inaccessible

2.5.6 Recent high quality datasets: no trade-offs, no privacy

There have been a couple of recent attempts to collect and release high-quality ground-truthed datasets for activity inference, notably the SHL dataset [GCM+17] and the TMD datset [CLB+18]. Both of them work only on Android and collect data from a variety of possible sensors without a power control, so they cannot evaluate the power/accuracy trade-off. The TMD dataset contains 31 hours of data and does not include location information. Although the SHL dataset contains 7 months of data from 3 users [CMG+17], not all the data is actually public — only subsets of 229 hours and 391 hours have been released so far[16] in conjunction with machine learning challenges.

2.5.7 Power/accuracy trade-off with artificial trips: no reproducibility

The closest related work is [HSHM, Har] which used artificial trips to evaluate the power/accuracy trade-off of multiple smartphone apps in Toronto. Their approach is intended to meet 2 of our 3 requirements - they do not address privacy since they do not publish their data. Their rationale for using artifical trips was to ensure a balanced sampling of modes and trip lengths. Other limitations of their work were: (i) they did not use a standardized platform for data collection so their trade-off points were represented by various app implementations, (ii) their spatial ground truth was from a single android "logger" phone with manual corrections instead of predefined trajectories, which required considerable manual post-processing, and (iii) they do not appear to repeat timelines exactly and account for context-sensitive variations in sensing between multiple runs of the same timeline. Nevertheless, there is considerable overlap between their requirements and ours, which is an encouraging sign that our method meets the needs of the mobility community.

2.6 Conclusion

Sensor-based infrastructure and vehicle data has long been available, but traveler focused sensed and surveyed data had to wait for the widespread popularity of smartphones. Smartphone apps are the most suitable source for travel demand modeling. Smartphone app-based systems exhibit a builder–deployer gap, are typically closed source and tailored to one particular application. The few open source platforms appear to have only been deployed by the platform creators.

Context sensitive sensing can reduce power consumption, and implementations incorporated into phone OSes have been exposed as virtual sensors. Using virtual sensors necessitates modifying analysis algorithms designed for GPS devices with

[16] `http://www.shl-dataset.org/download/`

inflexible assumptions about data quality. There is no existing dataset for HMS evaluation that uses virtual sensors, captures trade-offs, preserves privacy and ignores transient effects.

We now examine the overall architecture of the e-mission platform before diving more deeply into specific modules.

Chapter 3

Computational mobility architecture

3.1 Introduction

Computational Mobility (CM) focuses on techniques to collect and analyze human mobility data (Chapter 1). Sensed data can automatically infer *precise* information, such as the origins and destinations of travelers and the time it takes to travel. However, only humans can determine *behavioral information*, such as cost trade-offs and trip purpose, or *perceptual* information such as trip quality. This combination of sensed and surveyed data is critical to gain a holistic understanding of travel. Smartphone app-based systems are the practice standard for mobility data (Section 2.1). Other sources of transportation data, such as infrastructure and vehicle sensors, and other sources of mobility data, such as cell tower data, are unable to adequately combine sensed and surveyed data.

Historically, human mobility data, captured in phone-based household travel surveys, has been used for travel behavior modeling. Expanded data collection by smartphone apps also enables city planners, transport engineers, healthcare advocates, and gaming gurus develop persuasive applications based on longitudinal travel diaries. Building such applications currently requires undertaking a significant software engineering effort. There exists neither a comprehensive platform to collect data nor transparent access to the data once collected.

We believe that this oversight is attributable to the *builder–deployer gap* in this domain. *Deployers* (e.g., mobility researchers) use these systems as tools in their work, focusing on the application, while *builders* (e.g., computing experts) focus on building the systems themselves. We propose an interdisciplinary approach that combines system-building rigor with the concerns of deployers to generate a family of *human mobility systems* (HMSes).

This chapter includes two main contributions.

- It describes a **platform** generalized from three canonical, real-world use cases. The platform includes novel design features to encourage extensibility and reuse. To our knowledge, this is the first such HMS platform in which the applications were developed by groups other than the primary platform builder, and installed by end users on their personal devices. It is also the first such platform evaluated using quantitative metrics.

- It outlines an **architecture** for this class of platforms. The platform architecture is complete, detailed, and end-to-end. It identifies the tiers that typically constitute such platforms, breaks them up into individual modules, determines the design trade-offs for each module, and classifies the modules as core and extensible.

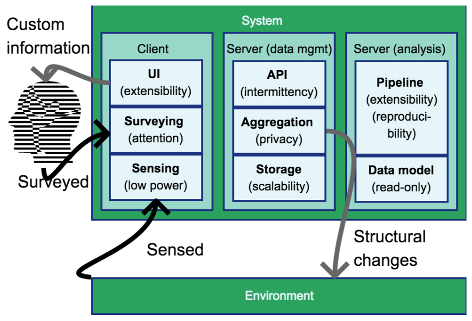


Figure 3.1. High-level components of HMSes and their primary challenges. Such systems receive *inputs*(black arrows) from sensors (e.g., travel trajectories) and surveys (e.g., trip quality). They can also provides *outputs*(gray arrows) of personalized information to travelers and of aggregate metrics to planners. The aggregate metrics can be used for short-term (e.g., congestion pricing) or long-term (e.g., new transit line) changes.

The rest of this chapter is structured as follows. First, it draws analogies between buildings and software to intuit the relationships between architectures, platforms and systems (Section 3.2). It then outlines the platform tiers and their interaction, and highlights three novel features that differ from the prior work (Section 3.3). Next, it shows the modular decomposition of each tier, extracts the design challenges for each module, and document the choices that the platform supports (Sections 3.4, 3.5, 3.6). It ends with examples on how to extend these modules to instantiate systems. The examples show that the complexity of the changes required is proportional to the effort (Section 3.7).

3.2 Architectures, platforms and systems

We can explore the concepts of software architectures, platforms and systems by drawing analogies to their building counterparts. These analogies are not perfect, but can provide an intuitive sense of the relationships to the deployer community.

- Software architectures and building architectural styles both outline the **common design elements or modules**. Architectural styles for buildings can

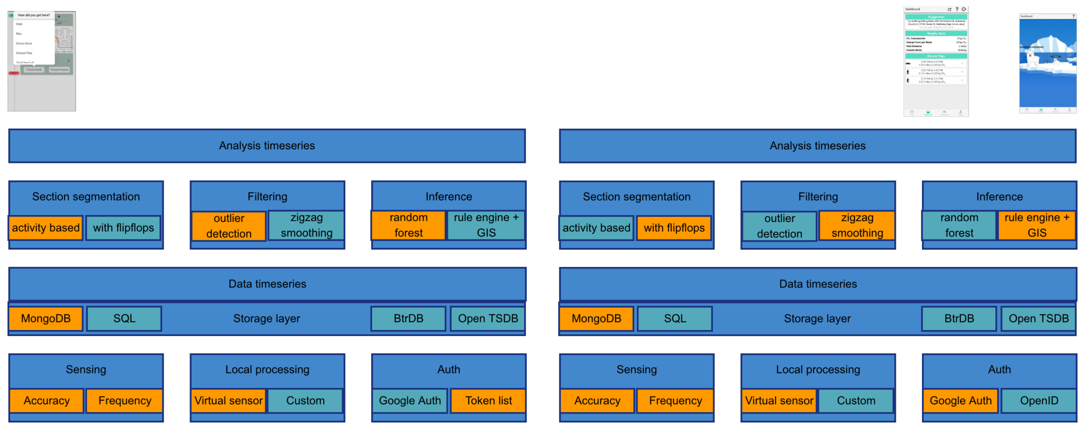


Figure 3.2. Architecure modules with alternate implementations, composed to create specific systems. The dark blue boxes are the modules, the light blue boxes are the alternate implementations, and the orange boxes are the ones chosen to create a user-visible system

include entry atria for Eichlers, exterior arcades for Mission Revival and high-pitched roofs for Gothic Revival. Standard modules for a three tier software architecture include the front-end, the webserver and the appserver.

- Artifacts represent **engineered instantiations of these abstract elements**. Building artifacts can be engineered with construction materials such as stone, wood, or earth. Software modules can be implemented in different languages (e.g., python, Java) or frameworks (e.g., Angular, React) and with different design trade-offs.

- Individual instances, whether buildings or systems, represent a **particular choice of artifacts** for the elements in an architecture. Individual buildings incorporate a subset of the design elements, engineered as artifacts, and realized in specific materials. Individual systems are composed of a subset of the modules with specific software implementations.

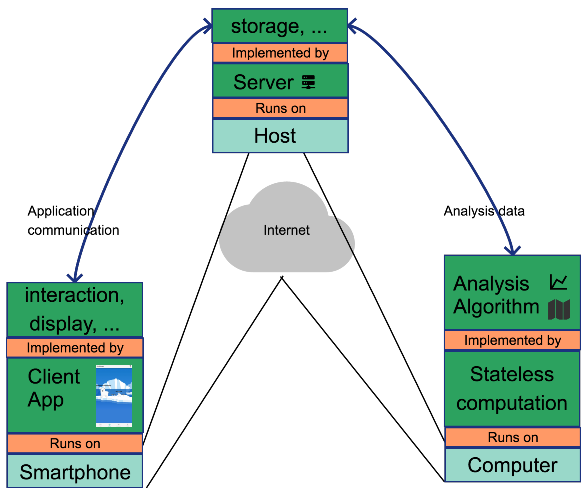
An architecture shifts focus from software programs to the general concepts that underlie a class of systems. A platform shifts the focus from the superiority of specific implementations to range of implementation choices and the design trade-offs associated with each (Figure 3.1). The artifact visible to the end-user is a system that is assembled from the design elements of the architecture (Figure 3.2), with implementations suited to the particular use case.

This generalization can be useful to both *deployers* and *builders*. *Deployers* can now have a shared vocabulary to compare different systems in this class, and determine the one that is most appropriate for their needs. Similarly, *builders* can now have a set of small, well-defined modules that they can focus on developing or improving, and a skeleton to put new modules that they develop in context.

The novelty of this architecture lies in its completeness, and provenance. Since the architecture needs to engage both deployers and builders, the particular tiers and modules that comprise the architecture are intentionally **not** novel. The goal is to use concepts that are so conventional that deployers can use Internet resources aimed at a lay audience to build familiarity. Future researchers can leverage this platform architecture for implementing their own platforms.

We recognize that the architecture for HMS presented here may not be the final word, as it is generalized from a small, but diverse, set of use cases. Our main goal is to use our interdisciplinary background to start a discussion around generalizing and evaluating human mobility systems.

3.3 Architecture overview and highlights



The platform architecture consists of the client, server and analysis tiers, each of which consists of multiple modules, which in turn are defined by some key challenges. Information is collected by the client app (Figure 3.3, Table 3.1) and transmitted to the service on a server (Figure 3.4) for storage. Reproducible analysis algorithms (Figure 3.5) are run on the raw data to generate inferences. Here, we focus on the novel concepts that help us achieve extensibility, reproducibility, and privacy.

3.3.1 User Interface (UI) channels

Deployers who use the platform should be able to configure it according to the requirements of their use case. Modifications may include removing or adding interaction components or simply changing the colors to match the deployment institution.

E-MISSION separates the UI into a separate layer (or skin) that can be easily modified using standard web technologies. It also supports separate UI *channels* or *themes* that the end user can switch to. The UI channels also contain the configuration options for the native layers. Separating the UI channels and publishing them independently can also help with standardization and reproducibility.

3.3.2 cross-platform event generation

Generating app-local event notifications as the sensing moves through the location state machine (Section 3.4.1) allows context-sensitive modular operation. This loosely coupled event architecture separates the notification code from the core FSM while still allowing it to reuse its functionality. For example, in E-MISSION, the event notifier module can listen to the FSM notifications and display a deployer configured notification message to the traveler.

Trip end notifications are particularly challenging on iOS since it supports a limited set of *background modes* for sensing **and** disallows time-based periodic sampling (Section 3.4.1). This limitation makes detecting start and end trip times on iOS tricky since, without time-based sampling, dwell time detection is not guaranteed to work. In E-MISSION, a silent push notification functions as a coarse timer (Section 3.4.3) that permits periodic housekeeping, including acting as a backup for the built-in trip end detection.

3.3.3 Pipeline and data model

The raw data from either sensors or surveys needs to be converted into *inferred* data (e.g., trips, places, sections, modes) to make it meaningful to both users and the community. This conversion is done through inference algorithms with stages for cleaning and post-processing. These inference algorithms need to be transparent and reproducible so that they can be understood and improved by the research community. We meet these goals by defining a data model and algorithm structure for reproducible analysis.

In E-MISSION, the pipeline consists of a sequence of linked transformation stages. The input data is read-only and immutable and is transformed through a sequence of intermediate outputs into the final output. The inputs are independent of any outputs, so are saved using a time association rather than a particular output. The pipeline uses pipeline states for efficiency so that it processes freshly arrived data incrementally.

3.3.4 Data ownership and aggregation

The individual travel diaries can be aggregated for structural changes. In addition to basic timestamp-based queries, E-MISSION also supports geo-queries and local time based filters (e.g., to support "commute time" queries similar to [FPV+13]).

One of the advantages of an end-to-end platform is that deployers have the opportunity to run their own server, control all data collected, and ensure that it matches local privacy standards. E-MISSION also primarily integrates with external services that are *open source*, ensuring that deployers can run local copies to avoid leaking data. (Section 3.3.4).

3.4 Client architecture

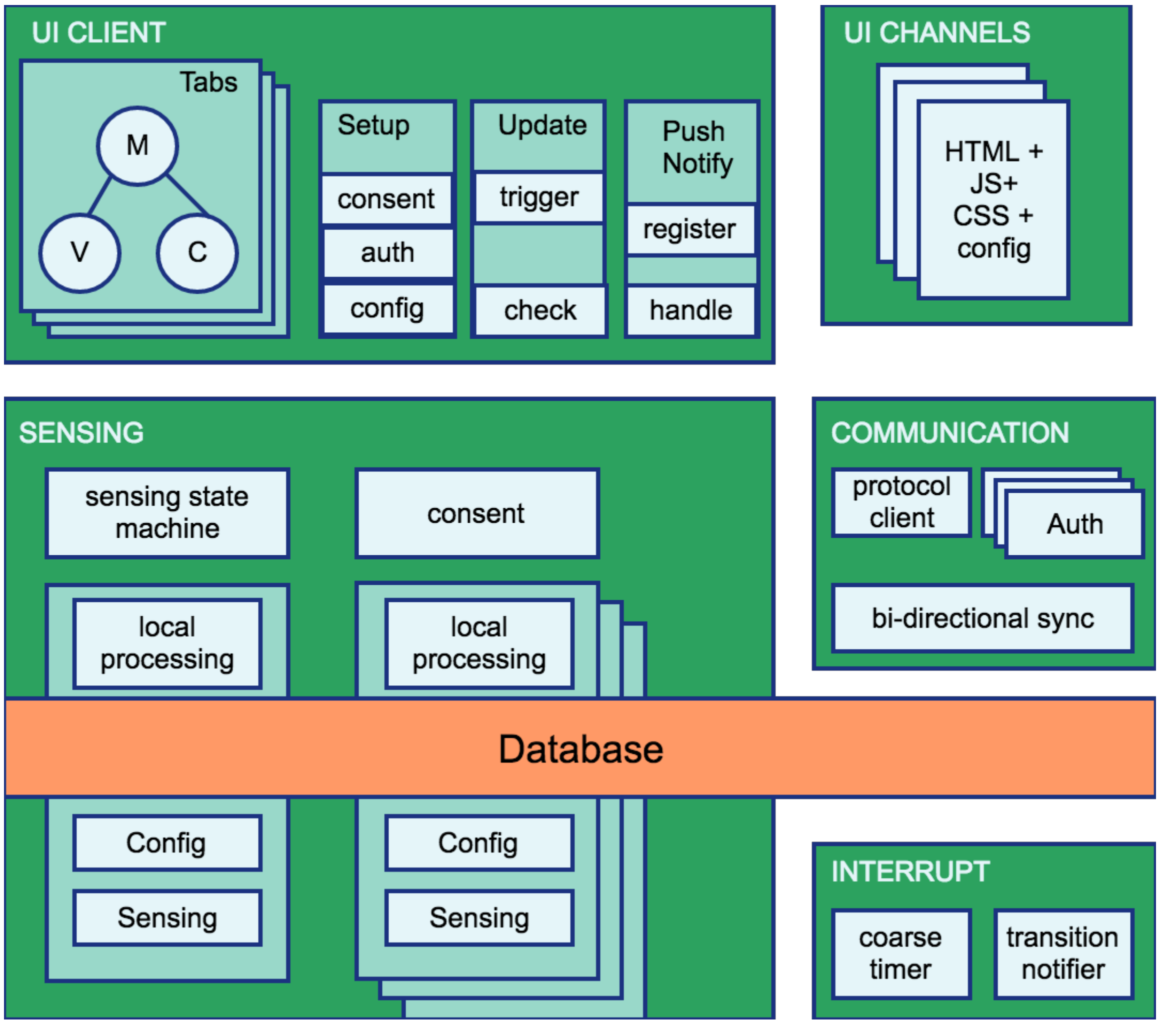


Figure 3.3. Detail of the client architecture, including modules for configurable sensing, robust communication and customizable UI

Most prior HMS projects have focused on the smartphone app and its ability to sense location and accelerometer data (Section 2.2). However, they focus purely on the sensing and ignore the human interaction component. This chapter develops a more complex architecture outlined in Figure 3.3 and Table 3.1) addressing this gap.

The core modules include configurable sensing, robust communication, and context-sensitive prompts. The novel component primarily involves the user interface and customization. The three canonical use cases that we consider modified the UI, configured the local end of trip detection module notifier to display different prompts, and configured the communication to send data to their own server instance.

3.4.1 Sensing

The sensing module is conceptually simple — it reads and stores sensor values, automatically, in the background. However, power and latency considerations are important while choosing a particular point in the design space.

Local buffering The primary storage trade-off relates to the frequency at which the data is uploaded to the server. While it may seem intuitive to use the server directly as storage by uploading the data as it is read, the radio draws significant power when turned on, so data should be buffered locally as much as possible. In the case of primarily passive data collection, such as for HMS, it is sufficient to upload data after a trip is complete.

Buffering also reduces data loss due to poor connectivity, and decreases the latency of computations on locally sensed data. However, it increases the latency of aggregate operations computed on the server, such as traffic speeds or counts for particular segments.

E-MISSION buffers data until the end of a trip. It also periodically pushes pending data to the server when triggered by the coarse timer.

Local processing The primary processing trade-off involves latency versus flexibility and complexity. Local processing on buffered data has the lowest latency but the least flexibility, since it has to be implemented in native code for each mobile OS (e.g., android, iOS, ...) that the platform supports. Sensors that generate large amounts of local data, such as the accelerometer, threaten to overwhelm the local buffer. Local processing allows the creation of custom virtual sensors whose compact results can be stored instead of the verbose raw values.

E-MISSION uses local processing for *low latency,* basic filtering of location points for use in the location state machine.

Location state machine iOS supports a limited set of *background modes* [1], restricting the sensors (i.e., sound, location and bluetooth) that can be accessed in the background. The sensor must be relevant to the published app functionality (e.g., the VoIP background mode can only be used by VoIP apps, not mapping apps), the user must permit the app to access these sensors, and then explicitly permit them to be accessed in the background. This means that all other sensors (e.g., accelerometer) have to piggyback on one of the supported sensors for their operation. Further, the sensor APIs disallow periodic sampling, probably to prevent them from being used as periodic timers - e.g., the location sensor has a distance filter instead of a time filter. Detecting the trip end is challenging

[1] `https://developer.apple.com/library/archive/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html`

with a distance filter since once the user has stopped moving, the app will not receive any data until she starts moving again.

E-MISSION uses OS-specific finite state machines (Chapter 4) to turn tracking on and off based on local processing. On iOS, it uses the visit detection virtual sensor, backed up with a coarse timer based on hourly silent push notifications from the server. When the timer is triggered, the iOS version of the app checks the buffered data and generates the trip end event if appropriate. It piggybacks on the location API to read both location and the motion activity sensor. On iOS, the motion activity for the duration of the trip is read at the end, before the data is synced to the server.

Consent Most mobile OSes already require explicit consent for access to privacy-sensitive sensors such as location. However, additional regulations such as the European General Data Protection Regulations (GDPR) or academic Institutional Review Boards (IRB)s may require deployers to obtain more explicit consent that covers not just which data is collected, but also how it is processed and stored. All sensing should be stopped until explicit consent is received, and the consent should be documented for future reference.

3.4.2 Communication

The communication module deals with automatic upload of collected data and download of recently computed data for improved performance. This module needs to handle all aspects of communication, including establishing connections, authentication, and dealing with errors.

Auth All API calls to the server that transmit or receive personal data should be authenticated. The most basic form of authentication is to send a stored password, entered by the user, from the app to the server. While this is intuitive and easy to use, it should be combined with verification to avoid email hijacking.

For short studies with significant researcher interaction, an alternative is to pre-generate a list of random tokens and hand them out to participants. The researcher then does not need to know the users' email and can just use the unique token for all indexing.

For longer studies, the OAuth standard specifies the generation of encrypted tokens (JWT) with configurable expiry times. OAuth JWT tokens can be generated using open source auth servers such as Keystone, or by integrating with third party sign-in provides such as Google or Facebook.

bi-directional sync The main consideration for the bi-directional to/from data transfer is the **D**urability component of ACID transactions. Since any data transfer can be unreliable, the transfer should handle both poor connectivity and potential server errors without losing data.

One technique to accomplish this is to delete buffered data only after a `push` call fully succeeds. This may result in duplicate data from partial retransmissions but will not lose data. iOS allows apps to run in the background for no more than 30 seconds, so this code path should use parallel, async calls and rate limiting to speed up execution.

protocol client The HTTP REST protocol is a popular choice for client-server communication in prior HMS. However, pub/sub protocols such as MQTT, are popular for iOT systems. The resulting trade-offs are closely related to those for local processing (Section 3.4.1). REST is better for batched intermittent connections, where connection setup and teardown do not cause significant overhead. MQTT works better for data that is continuously streamed to the server since the persistent connection reduces overhead. Again, for primarily passive data collection, REST is sufficient.

3.4.3 Interrupt handler

The interrupt handler deals with external triggers. Two current examples are:

Coarse timer We need to have a timer interrupt fire periodically to perform regular maintenance and recover gracefully from unexpected situations. For example, we may want to: (i) push any pending data that was retained in the buffer from previous partial retransmissions, or (ii) reset the location state machine if it is an inconsistent state.

This may appear to be trivial, since most standard OSes include a timer interrupt. However, in order to reduce power drain, many mobile OSes have limits on background operation for non-system services. If the limits are too strict (e.g., on iOS), we may need server intervention (e.g., *silent push notifications*) for reliable operation.

Event notifier The module also needs to deal with context-specific user notifications. While various events can be detected by the location state machine, the notification message and actions should be configurable. Since the event is detected in the background, the message should be configurable by the UI but not rely on the UI for display.

3.4.4 User Interface (UI)

The primary trade-off for the UI is performance versus effort. Native UIs have better performance, but more effort. This increased effort is intrinsic — native UIs

Module	core/ ext	key challenge	design choice
Local buffering	core	data upload frequency	at trip end
Local processing	core	flexibility v/s latency	both, mostly server
Location state machine	core	continuous v/s duty cycled	configurable, duty cycling recommended
Consent	core	none	configurable
Auth	ext	none	configurable
bi-directional sync	core	durability with intermittency	delete only after success
protocol client	core	batching/streaming	HTTP REST
Coarse timer	core	OS limits on background operations	silent push notifications
Event notifier	ext	background detection	user visible messages and handling are configurable
Setup	ext	user attention	UI configurable
UI update	core	user expectations	can configure, shouldn't need to
notifications	ext	user attention	UI configurable
UI channel	core	UI friction	users can click on a link to switch channels

Table 3.1. Brief description of the modules for the client tier, their primary challenge and the design chosen by the E-MISSION platform

will require a different implementation for each mobile OS that needs to be supported. Using a hybrid app approach, (e.g., PhoneGap, Apache Cordova), allows the core modules to be in native code while the UIs use standard web technologies (HTML+CSS+Javascript). This approach allows a single, consistent UI to be reused across multiple mobile OSes, while channel specific UIs can be dynamically downloaded on demand.

While the UI can be completely customized to meet the needs of the application, all the three canonical use cases have used these three core components.

Setup An onboarding process introduces the app, acquires consent, and authenticates the user. This process can also include other initial steps, such as choosing a username or collecting demographic information.

UI update There needs to be a mechanism (triggered on app launch, or by the *coarse timer interrupt*, Section 3.4.3) that periodically checks for updates to the UI channel and applies them, potentially asking the user for confirmation.

Notifications The app needs to register for event notifications — both for context-specific user notifications and, due to OS restrictions (Section 3.4.3) for coarse timer interrupts.

3.4.5 User Interface (UI) channels

Each deployer who uses the platform should be able to configure it accordingly. Since the user interacts primarily with the UI, we expect that deployers might want to change the information displayed, the qualitative input solicited and the controls visible to the end user.

In order to provide maximum flexibility, platforms might want to support separate UI *channels* that the end-user can switch to. Each UI channel can have a completely different look and feel and can specify completely different configurations for the various modules.

Supporting dynamic UI *channels* also includes several other benefits:

Randomized trials It is easy to conduct randomized behavior trials by randomly directing end-users to different channels as they install the app.

Custom server support Since modules can be configured by the UI, installs using different channels can send their data to different servers. This allows deployers to have complete control over the collected data.

Standardization Particular deployer communities (e.g., travel survey groups) can develop canonical user interfaces for their particular use cases. This makes it easier to launch new examples of that use case, and also shortens the methods section of the resulting papers.

Reproducibility Such standardization would be difficult for behavioral studies, in which the goal is to innovate new methods of interaction. However, once the new interaction method has been embodied in a published channel, the study can be generalized or reproduced by recruiting new users and asking them to use the channel.

3.5 Server architecture

Although sensing (Section 3.4) is typically the focus of the related work, most deployers will also want to upload the data to a server for long-term storage, shared access, and complex analysis. The architecture of this server software is typically elided from platform descriptions. For example, a review of Experience Sampling Software ([PLMM15], Table 1) indicates that only ohmage includes a server component. The key modules for this tier are storage, data communication, and analysis. This section describes these modules in greater detail.

Module	core/ ext	key challenge	design choice
storage	core	data representation, scalability	time-series for sensed and analysed data, K-V store for modifiable objects (e.g. `config`)
Incoming buffer	core	complexity v/s flexibility	flexibility
Webapp	both	core endpoints fixed; easy to add others	separate routes from related functions
Push notify	both	push provider idiosyncracies	silent push is core, targeted notifications are configurable
Integrations	both	external API issues, hosting	host open-source instances for integration

Table 3.2. Brief description of the modules for the server tier, their primary challenge and the design chosen by the E-MISSION platform

3.5.1 Storage

Storage is the key component of the server architecture. However, the actual storage instance or database product chosen depends on multiple factors like the number of users, the response time expected, and the resources available for the data collection. So we instead focus on the types of data collected and the broad storage category for each data type. These broad types are listed below.

Input timeseries The input data received from the smartphone app represents a spatiotemporal datastream which maps logically to a timeseries database. Our data model and analysis pipeline (Section 3.6) treat this data as *read-only*.

The data can be conceptually viewed as separate user databases, each with multiple streams of data (e.g., `location`, `transition`). Most processing will work on one user at a time, multi-user queries will be aggregated across user databases. This formulation is compatible with future privacy preserving implementations.

Note that we actually have intermittent timeseries data because the sensors on the phone are not typically guaranteed to be periodic. And even if they were, if we use the state machine for lower power drain, there will be no data for long stretches of time. However, we still consider it to be a timeseries, since the primary querying method will be for a time range, and the primary index should be the timestamp associated with each data point.

Analysis timeseries Analysis results generated after processing the input data are stored in a separate timeseries database. While the volume of this data is not likely to be as high as the raw data, it is also time-indexed, and using the familiar timeseries interface allows us to consistently stack analysis results (Section 3.6).

K-V store Modifiable objects (e.g., `profile`, `config`) are conceptually modifiable objects associated with a particular user by a key. If the deployers would like versioning, and don't want to install two separate database packages, this data can also be stored in the timeseries database — the entry with the most recent timestamp is valid. However, these data will be looked up by key and not by time range, unless somebody requests an audit. So it does not need to be indexed on the timestamp, although such an index does not hurt.

Incoming buffer Since the background operation on iOS is time-bound (Section 3.4.2) we want the data received from the phone to be stored as quickly as possible. As server and database loads grow, directly storing incoming data into a potentially distributed timeseries database could introduce high latency. Instead, we can dump the incoming data into a separate, potentially local buffer, and move it into the timeseries before processing. This additional step also allows us to run preprocessing steps before insertion.

3.5.2 Aggregation

Aggregate queries (e.g., mode shares, or pedestrian counts) can provide useful information without directly exposing individual participant trajectories. The basic format of e-mission aggregate queries is simple — the analyst provides a query type and a spatiotemporal range and then receives mode-specific aggregates. Some non-obvious details to note are:

Geo queries Analysts may want to restrict data retrieval to a particular region. This implies that all aggregate queries should support an (optional) georegion, and the underlying timeseries storage should support geo-queries as well.

Time selections Most timeseries will support range queries where the range is specified in UTC. However, deployers may want to query by time slices instead — e.g., studying commute time travel patterns might involve accessing data from 2pm – 5pm for the month of April [FPV+13]. Storing expanded times in the local time can support such disjoint time ranges.

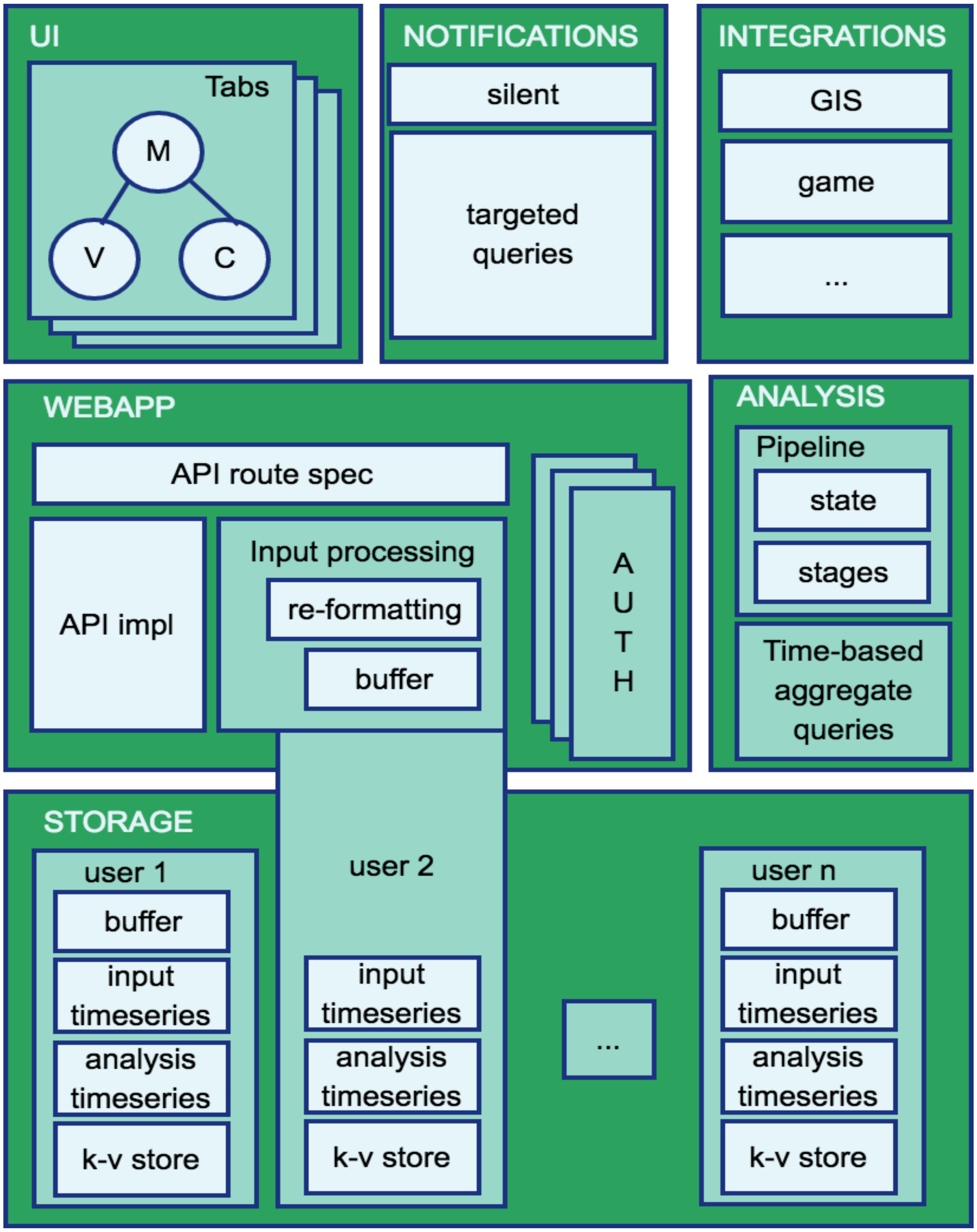


Figure 3.4. Server architecture, including modules for storage, communication and integration.

3.5.3 Data ownership

Location tracks are extremely privacy sensitive. Even if the location data are not explicitly linked to personally identifiable information, such as name, e-mail, or phone number, long-term travel patterns can leak home and work information. Platforms should enable privacy-enhancing features by default, and include documentation on proper configuration for security.

E-MISSION includes the following privacy-enhancing features by default —

Consent The data collection is only enabled after the application has called `markConsented` on the data collection module.

Third party leakage Since e-mission is an end-to-end platform and the server is also open-source, deployers own the data that they collect without any third party intermediaries. Further, the core analysis algorithms integrate primarily with open-source projects such as Nominatim[2] or Overpass[3]. This allows deployers to run their own copies of the servers and avoid sending location traces to services such as Google Roads[4] or the Mapbox Map Matching API[5]

However, no open-source project can guarantee that it is deployed according to the builder's intent. Ultimately, deployers are also responsible for ensuring that their data collection and data handling procedures are compatible with local standards such as the European Union's General Data Protection Regulation (GDPR) or India's Constitutional Right to Privacy. For example, although the platform clearly documents the use of an encrypted filesystem to protect both data and logs on production systems[6], the `cci` project, in accordance with their Institutional Review Board (IRB) protocol, collected data on a windows desktop hosted on the UC Berkeley campus and did not use encrypted filesystems.

3.5.4 Other components

The other components of the server architecture are fairly straightforward.

Webapp The webapp layer defines the API routes used by all clients, including the smartphone app, and any browser-based UIs. The webapp layer also authenticates all user-specific API calls, and needs to support the same set of authentication methods as the smartphone app (Section 3.4.2).

[2]`https://nominatim.openstreetmap.org/`

[3]`https://wiki.openstreetmap.org/wiki/Overpass_API`

[4]`https://developers.google.com/maps/documentation/roads/intro`

[5]`https://docs.mapbox.com/help/glossary/mapbox-map-matching-api/`

[6]`https://github.com/e-mission/e-mission-docs/blob/master/docs/e-mission-server/deploying_your_own_server_to_production.md#cryptfs-suggested`, `https://github.com/e-mission/e-mission-docs/blob/master/docs/e-mission-server/deploying_your_own_server_to_production.md#configuring-logs`

Push Notifications This module integrates with push notification services to send both *targeted surveys*, which send a link to a survey based on user mobility patterns, and *silent notifications*, which are used as coarse timer interrupts on the smartphone (Section 3.4.3).

Integrations This module handles external integrations. Current examples include `OpenStreetMap`, for GIS lookups, and Habitica, for gamification.

3.6 Analysis architecture

HMSes convert raw data to various analyzed outputs, including the travel diaries that form the basis of computational mobility 1.4.1. They perform this conversion by cleaning and post-processing the raw sensor data using inference algorithms. These inference algorithms need to be transparent and reproducible so that they can be understood and improved by the research community. We meet these goals by defining a data model and algorithm structure for reproducible analysis. This section focuses on the high level structure and the data model for the algorithms, more details are in Chapter 5.

Module	core/ ext	key challenge	design choice
Data model	both	reproducibility	core principles: incoming data is read-only; output data is write-once; can define new objects that fit the model
Pipeline	both	extensibility + reproducibility	re-running the pipeline multiple times generates identical inferred results

Table 3.3. Brief description of the modules for the analysis tier, their primary challenge and the design chosen by the E-MISSION platform

3.6.1 Pipeline

In E-MISSION, the only permanent state of the analysis component is the input data received from the smartphone app. The algorithm is structured as a pipeline with a set of deterministic *stages*, and the input to each stage is the output from the previous stage. Since each stage only modifies its output, the stages are idempotent. This implies that, given the same inputs and the same algorithm stages, the results will be the same.

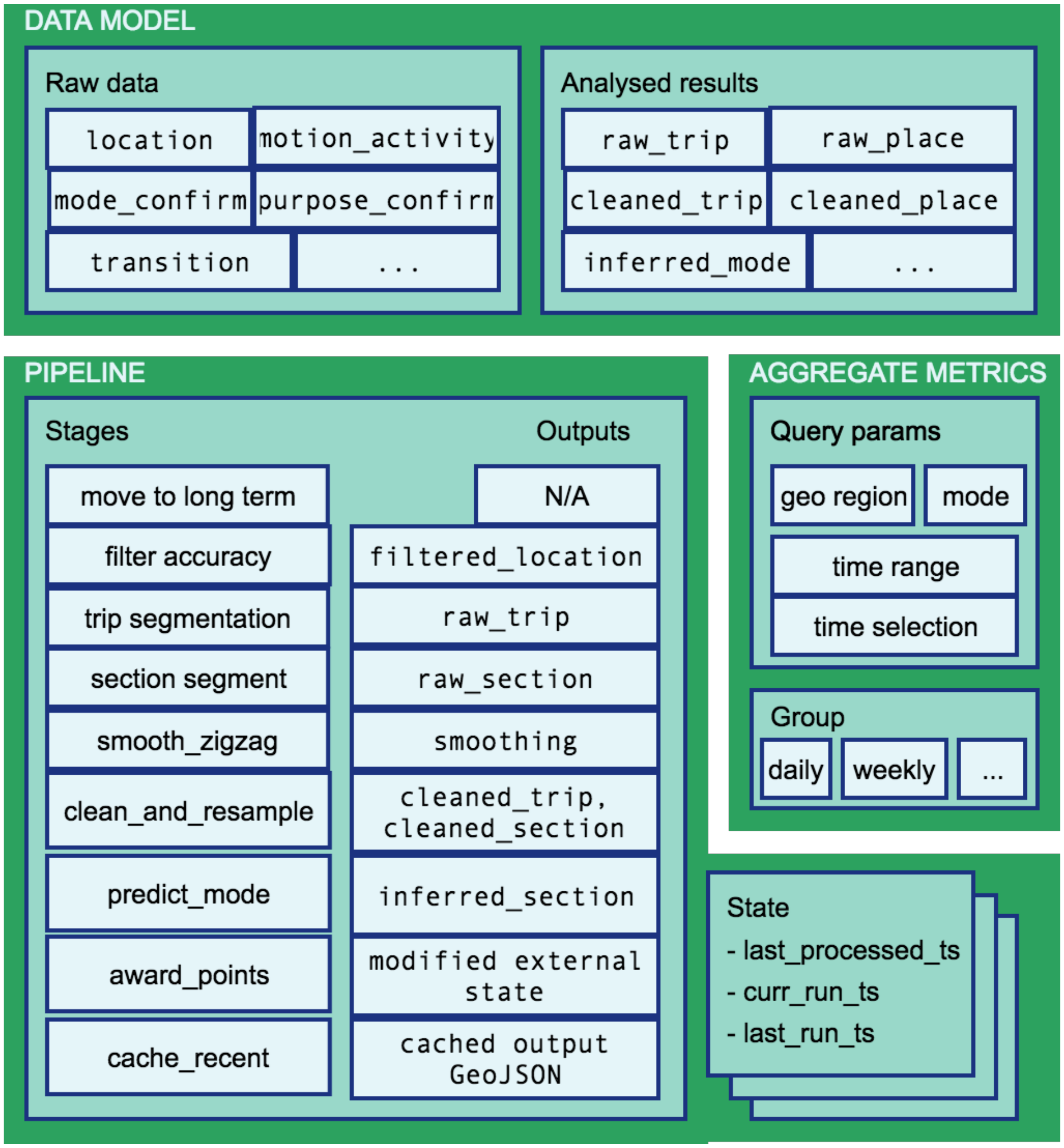


Figure 3.5. Analysis architecture, including modules for processing the data in idempotent stages, a data model that supports such an algorithm, and aggregate queries.

The only exception to this rule occurs when some inputs are retrieved from external integrations. For example, integrating with a GIS that includes transit routes can help distinguish transit from bicycling. But if the transit routes are modified, re-running the pipeline at a later date can change the results. If steps that integrate with external services need to be reproducible, the services also need to support versioned queries — e.g., the ability to answer the query based on the data version that was current at the time.

For steps that do not have external integrations, or whose integrations support versioning, this structure enables various important steps.

Reproducibility Analysts can reproduce results from any version of the algorithm simply by running the code on a fresh set of inputs. If the code is versioned properly in a source control system (e.g., `github`), then reproducing results at a previous time t is as simple as: (i) downloading the raw data, (ii) checking out the version of the source code at time, (iii) running that version on the raw data. This allows analysts to reproduce prior results even as the codebase has evolved beyond the time that the data were collected.

Extensibility If a researcher develops a new algorithm for a particular stage, she can run both the current state of the art and the new algorithm against the same input data and compare the results. If she chooses to publish the algorithm implementation, other researchers can reproduce her results by running the published algorithm against the raw data.

3.6.2 Data model

These design principles, and the notion of interchangeable steps in the pipeline imply an unconventional model structure with the following characteristics:

- The pipeline can choose from various candidate algorithms for each of the steps. For example, segmentation algorithms from one source can be combined with inference algorithms from another source to improve the overall efficiency.

- There can be multiple outputs in existence in parallel at one time. Instead of choosing from the multiple candidate algorithms, we can run them all in parallel, and generate multiple outputs at the same time. Each output can be tagged with the generating algorithm to eliminate confusion. This allows us to experiment with ensemble methods instead of a single solution.

- Inputs can never depend upon outputs. If analysts can delete and regenerate outputs at will, or use ensemble methods with multiple outputs at a time, then inputs cannot refer to particular outputs. This is not a challenge for sensed data since it is typically collected before the analysis is run. However, it can be a challenge for surveyed data. Users confirming the mode for a particular trip may lose their confirmations if that trip object is deleted. And if there are many trip objects for that time range, it is unclear which one to modify.

These requirements can be accomplished in at least two ways.

Carry forward The data from the previous step is carried forward to the next step. For example, every trip can store the location points associated with it. Unfortunately, as the number of inferred objects increases, this can get increasingly unwieldy. Some example questions are:

- Since each multi-modal trip can be split into multiple *sections*, should every section also store the location points associated with it? This will cause duplication of location points between trips and sections.
- How can we store ground truth for a trip as it ends, potentially before the pipeline has run and generated a trip object?

Time association The newly created objects for each step are associated with start and end time information. We can then associate raw or processed inputs with any object by querying for entries within that time range. This approach addresses many concerns with the *Carry forward* method. For example:

- Each section and each trip will have start and end timestamps. The set of points associated with a particular section or trip is then just the set of points between the start and end timestamps. The analyst can then use the same kind of query on a trip to retrieve: (i) *reconstructed locations*, which are resampled at a known frequency for consistency (ii) *filtered locations*, which are raw locations locally filtered for accuracy (Section 3.4.1) (iii) *locations*, which are raw locations with no filtering. This general query structure makes it easier for researchers to experiment with alternate algorithm implementations — each algorithm segments the raw data differently but does not duplicate it.
- Since ground truth is a user input, it should not contain a reference to inferred data. Instead, the ground truth object also contains the time range that the user has *confirmed* (e.g., with `mode` or `purpose`). The confirmed value for an inferred trip is represented by the `confirm` object that overlaps with the inferred time range. This approach can be used for both trips and sections, depending on how much editing power the deployer wishes to provide to the end-user.

The e-mission pipeline uses the *Time association* method since it meets our requirements and is consistent with our design goals. This additional flexibility does introduce additional complexity in the data access since there is no one stored object that contains all the information about a trip — instead the complete trip object has to be reconstructed by reading the location stream (raw, filtered, or reconstructed) associated with the trip and finding manual annotations (if any) that overlap with the trip. The e-mission server provides utility libraries (primarily under `emission.storage.decorations`) to such lookups easier.

3.7 Principle of proportional effort

In addition to being full-featured, a successful software platform for smartphone data collection must be easy to extend so that it can meet the needs of a variety of projects. Small configuration changes should be easy, and more significant additions to functionality should be achievable using well-defined extension points. Ideally, these changes should be made publicly available for reuse and reproducibility ([IHG12]).

3.7.1 Usage without customization

If the standard e-mission interface and functionality meet the needs for a study, the practitioner can simply file a research protocol with her institution's review board (IRB) and specify that she will use the e-mission platform for background location data collection. (This is similar to specifying the use of a platform like Qualtrics to collect survey responses.)

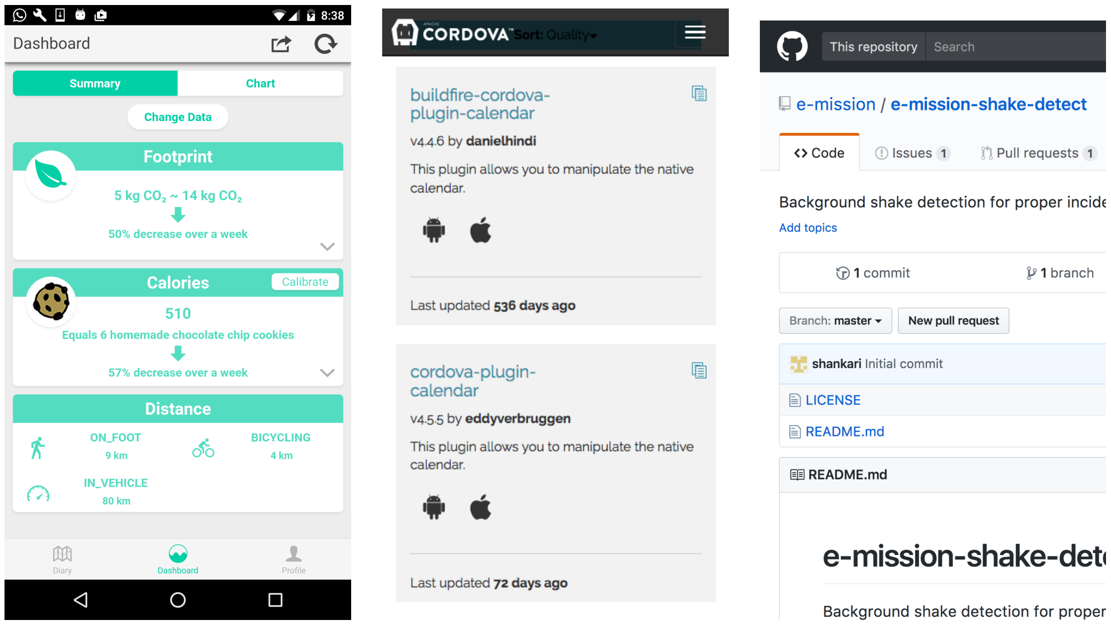
The practitioner would then instruct participants to download the e-mission app from the Android or iOS app stores, and obtain separate consent from the participants according to the method specified in the protocol. This consent would need to include the email address that the participant uses to register in e-mission, in order to confirm which users are associated with the study. At the end of the study, the practitioner would show the consent documents to the e-mission lead researcher and receive a copy of the data from those users.[7] Full in-app consent can be done with simple UI customization; see below.

Thus, practitioners can collect automatically sensed location and motion activity data without writing any code, simply by directing survey participants to use the app.

3.7.2 Extending the smartphone app

This section outlines a range of customizations, from changing the UI to incorporating existing implementations, to authoring new implementations, for the client tier of the platform.

[7]The standard e-mission consent document is available here: https://e-mission.eecs.berkeley.edu/consent



This plugin allows you to manipulate the native calendar.

This plugin allows you to manipulate the native calendar.

Background shake detection for proper incide

Background shake detection for proper

Figure 3.6. Options for client extensibility proportional to effort. (l-r): customizing the UI, adding existing plugins, writing a new plugin.

3.7.2.1 Easy: Customizing the user interface (UI)

Many practitioners will want to customize the user interface of the app: to add a study logo, to add custom consent, or remove unneeded features. This can generally be done with HTML and CSS changes alone, although functionality related to message prompts involves Javascript.

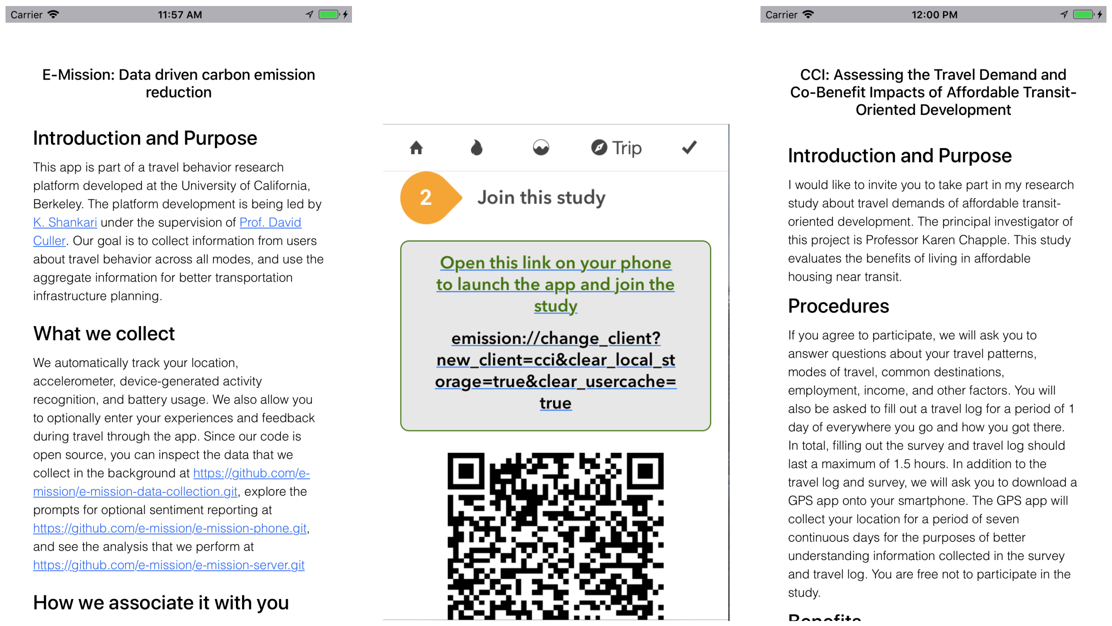
Because the UI is built using web components, it can be updated without deploying a new app to the stores. The e-mission platform supports multiple UI *channels*, meaning that practitioners can ask survey participants to install the standard e-mission app and then switch to the study-specific channel. A channel can be selected in the UI or by following a special URL or QR code. As soon as a user joins the channel, they are presented with study-specific information, consent, and login choices.

Such extensions are shared with the community as new branches on the `e-mission-phone` GitHub repository.[8]

3.7.2.2 Medium: Extending the phone app using existing plugins

e-mission is built using the Apache Cordova mobile app framework, which allows easy re-use of existing plugins. Functionality like reading a user's calendar or allowing

[8]https://github.com/e-mission/e-mission-phone



E-Mission: Data driven carbon emission reduction

Introduction and Purpose

This app is part of a travel behavior research platform developed at the University of California, Berkeley. The platform development is being led by K. Shankari under the supervision of Prof. David Culler. Our goal is to collect information from users about travel behavior across all modes, and use the aggregate information for better transportation infrastructure planning.

What we collect

We automatically track your location, accelerometer, device-generated activity recognition, and battery usage. We also allow you to optionally enter your experiences and feedback during travel through the app. Since our code is open source, you can inspect the data that we collect in the background at https://github.com/e-mission/e-mission-data-collection.git, explore the prompts for optional sentiment reporting at https://github.com/e-mission/e-mission-phone.git, and see the analysis that we perform at https://github.com/e-mission/e-mission-server.git

How we associate it with you

2 Join this study

Open this link on your phone to launch the app and join the study

emission://change_client?new_client=cci&clear_local_storage=true&clear_usercache=true

CCI: Assessing the Travel Demand and Co-Benefit Impacts of Affordable Transit-Oriented Development

Introduction and Purpose

I would like to invite you to take part in my research study about travel demands of affordable transit-oriented development. The principal investigator of this project is Professor Karen Chapple. This study evaluates the benefits of living in affordable housing near transit.

Procedures

If you agree to participate, we will ask you to answer questions about your travel patterns, modes of travel, common destinations, employment, income, and other factors. You will also be asked to fill out a travel log for a period of 1 day of everywhere you go and how you got there. In total, filling out the survey and travel log should last a maximum of 1.5 hours. In addition to the travel log and survey, we will ask you to download a GPS app onto your smartphone. The GPS app will collect your location for a period of seven continuous days for the purposes of better understanding information collected in the survey and travel log. You are free not to participate in the study.

Figure 3.7. Launching a custom UI through a configuration link. The new UI can be tailored to the survey's summary, consent and login choices. (l-r): base app consent, customization link, custom consent. Upon clicking the link from the phone or scanning the QR code, the app is launched with the new UI, so users never have to consent to the base app.

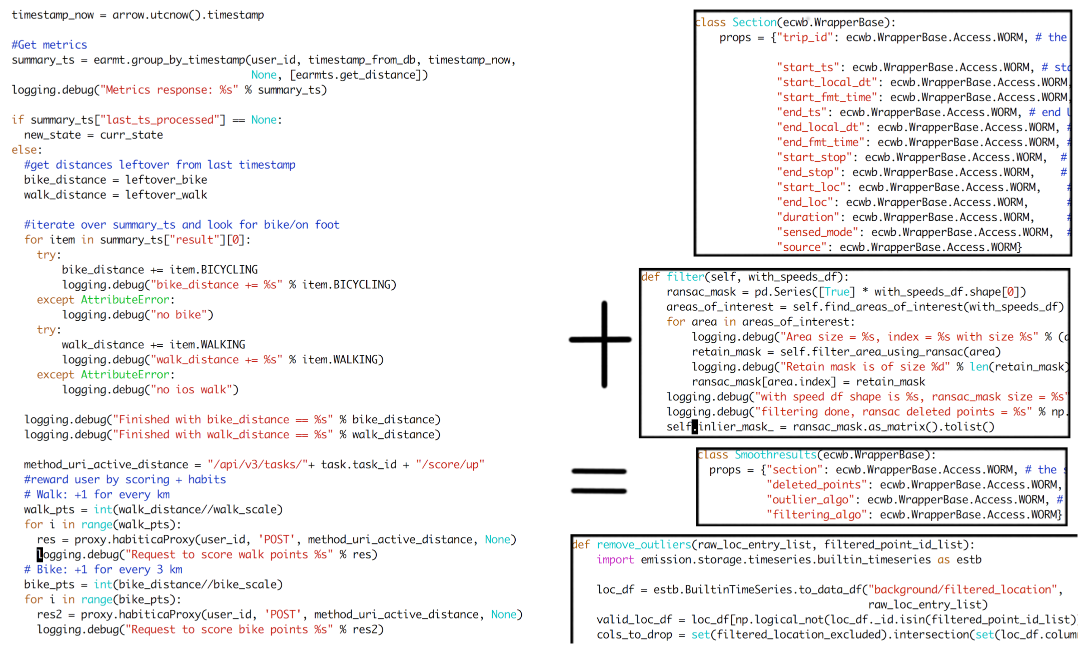
users to take photos can be added in this way. Cordova plugins are controlled using Javascript.

A phone app that has been extended through the addition of new plugins cannot be updated via the UI channels. Instead, a new app would need to be submitted to the stores with a new name and signing key. For iOS, the app must pass the App Store review process. The resulting app would have no obvious connection to the e-mission platform — it could have its own logo, and would be marked as owned by the organization that is submitting it.

Code for such enhancements can be made available to the community by forking the `e-mission-phone` GitHub repository and pushing changes to the fork. Once the project is complete, the enhancement could even be added to the standard e-mission app (in a new UI pane, for example). This would be done by submitting a pull request to the master branch of the `e-mission-phone` repository.

3.7.2.3 Hardest: Writing a new native plugin

Some projects may want to use sensing capability that is not currently supported in the Cordova ecosystem, for example by integrating with a sensor that measures stress from sweat, or using ambient noise to determine whether a car trip is shared



```
timestamp_now = arrow.utcnow().timestamp

#Get metrics
summary_ts = earmt.group_by_timestamp(user_id, timestamp_from_db, timestamp_now,
                                      None, [earmts.get_distance])
logging.debug("Metrics response: %s" % summary_ts)

if summary_ts["last_ts_processed"] == None:
  new_state = curr_state
else:
  #get distances leftover from last timestamp
  bike_distance = leftover_bike
  walk_distance = leftover_walk

  #iterate over summary_ts and look for bike/on foot
  for item in summary_ts["result"][0]:
    try:
        bike_distance += item.BICYCLING
        logging.debug("bike_distance += %s" % item.BICYCLING)
    except AttributeError:
        logging.debug("no bike")
    try:
        walk_distance += item.WALKING
        logging.debug("walk_distance += %s" % item.WALKING)
    except AttributeError:
        logging.debug("no ios walk")

  logging.debug("Finished with bike_distance == %s" % bike_distance)
  logging.debug("Finished with walk_distance == %s" % walk_distance)

  method_uri_active_distance = "/api/v3/tasks/"+ task.task_id + "/score/up"
  #reward user by scoring + habits
  # Walk: +1 for every km
  walk_pts = int(walk_distance//walk_scale)
  for i in range(walk_pts):
    res = proxy.habiticaProxy(user_id, 'POST', method_uri_active_distance, None)
    logging.debug("Request to score walk points %s" % res)
  # Bike: +1 for every 3 km
  bike_pts = int(bike_distance//bike_scale)
  for i in range(bike_pts):
    res2 = proxy.habiticaProxy(user_id, 'POST', method_uri_active_distance, None)
    logging.debug("Request to score bike points %s" % res2)
```

```
class Section(ecwb.WrapperBase):
    props = {"trip_id": ecwb.WrapperBase.Access.WORM, # the

             "start_ts": ecwb.WrapperBase.Access.WORM, # st
             "start_local_dt": ecwb.WrapperBase.Access.WORM,
             "start_fmt_time": ecwb.WrapperBase.Access.WORM,
             "end_ts": ecwb.WrapperBase.Access.WORM, # end
             "end_local_dt": ecwb.WrapperBase.Access.WORM, #
             "end_fmt_time": ecwb.WrapperBase.Access.WORM, #
             "start_stop": ecwb.WrapperBase.Access.WORM,  #
             "end_stop": ecwb.WrapperBase.Access.WORM,    #
             "start_loc": ecwb.WrapperBase.Access.WORM,    #
             "end_loc": ecwb.WrapperBase.Access.WORM,      #
             "duration": ecwb.WrapperBase.Access.WORM,     #
             "sensed_mode": ecwb.WrapperBase.Access.WORM,  #
             "source": ecwb.WrapperBase.Access.WORM}
```

```
def filter(self, with_speeds_df):
    ransac_mask = pd.Series([True] * with_speeds_df.shape[0])
    areas_of_interest = self.find_areas_of_interest(with_speeds_df)
    for area in areas_of_interest:
        logging.debug("Area size = %s, index = %s with size %s" % (
        retain_mask = self.filter_area_using_ransac(area)
        logging.debug("Retain mask is of size %d" % len(retain_mask)
        ransac_mask[area.index] = retain_mask
    logging.debug("with speed df shape is %s, ransac_mask size = %s"
    logging.debug("filtering done, ransac deleted points = %s" % np.
    self.inlier_mask_ = ransac_mask.as_matrix().tolist()
```

```
class Smoothresults(ecwb.WrapperBase):
  props = {"section": ecwb.WrapperBase.Access.WORM, # the
           "deleted_points": ecwb.WrapperBase.Access.WORM,
           "outlier_algo": ecwb.WrapperBase.Access.WORM, #
           "filtering_algo": ecwb.WrapperBase.Access.WORM}
```

```
def remove_outliers(raw_loc_entry_list, filtered_point_id_list):
    import emission.storage.timeseries.builtin_timeseries as estb

    loc_df = estb.BuiltinTimeSeries.to_data_df("background/filtered_location",
                                               raw_loc_entry_list)
    valid_loc_df = loc_df[np.logical_not(loc_df._id.isin(filtered_point_id_list)
    cols_to_drop = set(filtered_location_excluded).intersection(set(loc_df.colum
```

Figure 3.8. Options for server extensibility proportional to effort. (l-r): contributing plugins, modifying existing pipelines.

or not.

This would require writing native code (in Java and Objective-C or Swift) that reads the appropriate sensors, buffers them, and performs the inference either on the phone or on the server. Such projects can reuse the authentication, buffering, and communication components of the e-mission platform. They can also use the notification component to obtain additional information from the user.

Integration with the e-mission platform would allow the new travel data to be placed in a spatiotemporal context without having to re-write the location tracking and post-processing components. On iOS, restrictions preclude most sensors from being read in the background, but using the e-mission platform would allow plugins to attach themselves to the location tracking callbacks in order to read other sensor data.

Such an extension can be shared with the community by structuring the code as a Cordova plugin and publishing it on GitHub. Projects can then add the plugin like any other.

3.7.3 Extending the server functionality

This section outlines a range of customizations, from adding new offline analyses, to improving the implementation of analysis algorithms, to running a custom server, that are possible in the server and analysis tiers.

3.7.3.1 Easy: Adding queries or analyses

Aspects of the server software not related to the core outputs are structured as plugins, where new functionality can be added by simply writing a standalone Python script. Some examples are queries to find users who are targets for platform-initiated surveys or notifications to inform users about things related to their travel patterns. New analyses can be added to e-mission by generating a pull request from a fork of the "e-mission-server" repository.[9]

3.7.3.2 Medium: Modifying data pipelines

The existing pipelines for creating travel diaries are open to improvement. Practitioners may want to modify the segmentation, smoothing, or mode inference algorithms used by the core platform. These improvements will be more complex to integrate into the core platform, because we need to ensure that they are empirically valid and enough of an improvement to make the default. So while these changes can be contributed using a standard pull request, additional testing will be required before the changes can be merged. The reproducibility enabled by the analysis layer (Section 3.6) enables multiple rounds of testing and comparisons against the original code.

3.7.3.3 Hardest: Running a custom server

Some projects may have data storage and privacy requirements that differ from the core platform and are best achieved by running their own server. Projects that need special external integrations — with an Open Street Maps editor, for example — would also want to run their own server. Projects that modify the core data pipelines could also run a custom server to avoid integrating their changes with the core e-mission platform.

The e-mission server software can run on any Linux, macOS, or other Unix-like system. However, to manage a production backend, you need to be comfortable setting up SSL, obtaining the correct keys for authentication, and monitoring the pipeline logs for errors. Changes to the server software can be shared with the community by publishing the forked code so that it can be used to inform other projects that require similar integrations.

3.8 Conclusion

Human Mobility Systems (HMS) can form the basis for applications in domains ranging from travel behavior studies to crowdsourcing initiatives to identify structural barriers to transportation. We generalize an open-source platform, E-MISSION from

[9]https://github.com/e-mission/e-mission-server

use cases in these domains. *Deployers* can use this platform to instantiate customized systems for their own domains. In order to bridge the gap between *deployers* and *builders* of such systems, we also outline a clear platform architecture that describes the client, server and architecture tiers, and the components in each tier.

We now arrive at the technical core of the thesis, which delves more deeply into the sensing (Chapter 4) and analysis (Chapter 5) modules of the architecture.

Chapter 4

Background sensing using virtual sensors

Smart phones are ubiquitous, both in the developing and developed world. Ever since the early smart phones were introduced, researchers have been interested in the opportunities for gathering data by using smartphones as sensor platforms. But cell phones are not just a collection of sensor in a convenient package. They are also devices that provide real utility to users — that's why the adoption rates are so high. This means that users are sensitive to high rates of power drain, so energy efficient sensing has been a research focus for almost as long.

Historically, the two main mobile phone OSes — iOS and Android — have approached the power/utility trade-off differently. iOS has prioritized user interaction while eschewing background operation and multi-tasking. Android has provided a more traditional, multi-threaded, preemptive operating system. However, as the platforms mature, they have realized that: (i) background operation is critical to providing a good user experience; (ii) background operation done poorly is an energy hog; and (iii) application developers are rushed and will rarely take the time to optimize their background operations. Both platforms have converged towards a model in which background operations are permitted, but with OS-imposed restrictions that aim to manage the associated power drain.

At the same time, much of the prior research work in this space seems to have made its way into the shipping OSes as *virtual sensors*. For example, the recommended Location APIs on Android no longer require developers to choose a provider. The OS dynamically chooses the provider based on the desired accuracy and the context. Similarly, iOS automatically and continuously tracks motion activity, whether requested or not, and manages the power drain by using a separate low power co-processor to offload the collection and processing of sensor data.

This chapter explores the use of such virtual sensors for continuous background sensing of mobility data on both Android and iOS. Using the virtual sensors instead of re-implementing custom versions (e.g., [PFJM19, p.133]) reduces development and testing time by leveraging the preexisting engineering effort. However, both virtual sensor implementations and any associated background restrictions change with new releases and require adaptations to the sensing code.

The sensing trade-offs in this chapter are from 2015, when the sensing modules were first designed. The second round of evaluation in 2019 (Chapter 7), found that the virtual sensor implementations had been improved in the interim. The changes, and the work required to address them, are detailed in Section 4.5.

The rest of this chapter is structured as follows: Section 4.2 provides an overview of the behavior of the platform APIs on a limited set of mobility patterns, Section 4.3 maps this behavior into the motivation for our approach, and Section 4.4 constructs a simple 3-state model and generalizes the results to a wider range of patterns. Section 4.5 compares virtual sensors to custom implementations wrt both accuracy and maintenance, while Section 4.6 concludes.

4.1 Restrictions on background processing

The early days of fine-grained location tracking used dedicated devices with GPS sensors (Section 2.4). These devices provided raw data, such as the number of satellites for a fix, and used a simple sensing algorithm that collected data at the specified frequency.

Although smartphones also include GPS sensors, their usage model is not as simple. Instead, background operation is limited, either at the CPU level or at the sensor level, in order to increase battery life.

This section briefly summarizes the support for continuous background sensing, as of 2015, for both background scheduling and location detection on both the Android and iOS platforms. Consistent with the theme of this chapter, this section focuses on sensing API limitations. The analysis chapter (Chapter 5) focuses on the resulting data quality.

4.1.1 Android

On Android, background operation is permitted, unless the system turns on context-sensitive throttling. It also provides an accuracy-based location virtual sensor with a time filter and an accelerometer-based activity API.

4.1.1.1 Background scheduler

The Android OS provides a fairly standard preemptive scheduler. In particular, processes *can* be scheduled to run at time based intervals (every 30 secs) in the background. The OS provides specialized frameworks for cooperative scheduling — the `SyncAdapter/JobScheduler` interface for batching network operations is an example.

The 6.0 Marshmallow release moved from suggesting to forcing cooperative scheduling using *doze mode*. This is a low power mode that is activated when the phone is not plugged in, and has been inactive for a while (screen off, stationary). While in this mode, the OS suspends the regular scheduler, including processes scheduled by time, and performs all activities in regular maintenance windows.

4.1.1.2 Location APIs

The location interface consists of a "classic" `LocationManager` API which provides a time based access to a variety of location sensors such as GPS and the network, and what appears to be a context sensitive, rate adaptive `fused` API supplied through Google Play Services (GMS). For either API, a time filter can be specified to regulate the sample rate. The time filter is a hint — updates can be received more or less frequently based on interaction with other apps and the scheduler. GMS also supports a `geofence` API that monitors dwelling within a particular location in very low power mode.

4.1.1.3 Activity APIs

GMS supports a native, accelerometer-based [ZWHI15] API for activity recognition that can also be the basis for duty cycling. Applications can register for periodic activity updates, but there are no guarantees that the updates will be delivered at the requested rate.

4.1.2 iOS

On iOS, background operation is restricted by default. Apps that wish to operate in the background must explicitly list their reason in their manifest, and the reason should be compatible with their published description. iOS also provides an accuracy-based location virtual sensor with a distance filter and an always-on accelerometer-based activity API.

4.1.2.1 Background scheduler

The iOS background scheduler has the philosophy that power should be conserved for interaction with the user. This results in very impressive battery life on a stock phone, but places severe restrictions on background operation that require creative workarounds.

In general, processes **cannot run in the background**. This means that in general, application *cannot schedule tasks at specified time intervals* unless the app is in the foreground and the user is actively interacting with it. A small set of background operations can be enabled if permitted by the user, including two modes that can be used to request periodic wake ups. `Background fetch` is scheduled locally, but prior testing on iOS7 and iPhone 4 indicated that it is not reliable since the OS would duty cycle it based on network signal strength and user interaction patterns. `Remote push` wakes up the app to handle messages pushed from a server through a messaging service. While it is not guaranteed to be reliable either, it is fairly reliable in practice.

As an aside, the restrictions are severe enough that there is speculation that developers have resorted to playing blank sounds (playing music is a supported background operation) to keep their apps active in the background [1].

4.1.2.2 Location APIs

Fortunately, location tracking is one of the supported background operation modes. This means that the application can receive location updates even when it is running in the background.

The `standard` Location APIs on iOS do not allow users to specify a provider — instead, similar to the GMS `fused` API, users specify a desired accuracy (best, 10 m, 100 m, 1 km, 3 km), and the OS automatically picks a provider or set of providers. The sampling rate is controlled by a distance filter — there is no time filter, maybe because too many developers were using periodic location updates as a background timer.

If an application has requested location updates using the standard API, they will not be delivered if the application has been terminated due to memory pressure. A second `significant location changes` API can restart the app to deliver updates, but it is very coarse and does not support any configuration parameters. Updates are received when the OS determines that there has been a *significant change* — a term with no precise definition. This means that the error model for iOS tracking could include large gaps in tracking for which there is no workaround. Fortunately, this appears to be rare in practice.

iOS also supports a `geofencing` API similar to the one on Android. It also supports a visit detection API that can detect both trip starts and trip ends.

4.1.2.3 Activity APIs

iOS also supports a native activity recognition API that can provide periodic updates. However, the updates are **NOT** delivered when the application is suspended. This means that this API cannot be the basis for duty cycling based on activity, since we cannot reliably detect when the user is in motion again and turn on location tracking.

[1]Background data and battery usage of Facebook

4.2 Initial exploration of power accuracy trade-offs

Phone OSes already include sophisticated, context sensitive power management features and virtual sensors (Section 4.1). This raises the alluring prospect of assuming that energy-efficient background sensing is a solved problem, and using a simple sensing design that collects data continuously. This section explores the feasibility of this naïve sensing design by answering three questions:

1. Do phones really do nothing well?
2. Is continuous sensing on mobile phones a solved problem?
3. Does the virtual sensors in the phone OS (e.g., geofencing) help?

The questions are answered through an empirical evaluation. This empirical evaluation was a precursor to the improved evaluation procedure conducted later (Chapter 6). It incorporates the concept of evaluation across multiple OS states, and of direct comparison across multiple phones carried at the same time. However, it does not include repeated trips, spatial ground truth or an accuracy control, so it is much closer to the Toronto evaluation [HSHM] conducted by domain experts.

4.2.1 Experimental setup

The phone OS can dynamically adapt its behavior based on patterns of user activity and interaction, therefore it is not possible to extrapolate from a short sample to a long one. Evaluating the behavior of the phone OS over a day, requires both measuring the power drain over the course of a day and ensuring that it enters various operating states during that time. Further, since it is not known whether the operating states are deterministic, a direct comparison between data collected on various days is not known to be accurate.

Therefore, our experiment setup consisted of three identical phones for each platform — three iPhone6s and three Nexus 6 phones. We installed the data collection regimes that we wanted to compare on the three phones simultaneously, and carried each set of phones from the same platform in the same pocket.

In order to ensure that the phones had the chance to move through a variety of states, we divided each data collection day into “day” and “night” cycles, each of which was roughly 12 hours long. We also took several short trips throughout the “day” cycle. The trajectories for the trips were not identical, although their cumulative duration was roughly identical, and the “night” cycle occurred at different offsets in the day. This means that the data collected across days is not comparable.

4.2.1.1 States

We exercised the following states as part of the data collection:

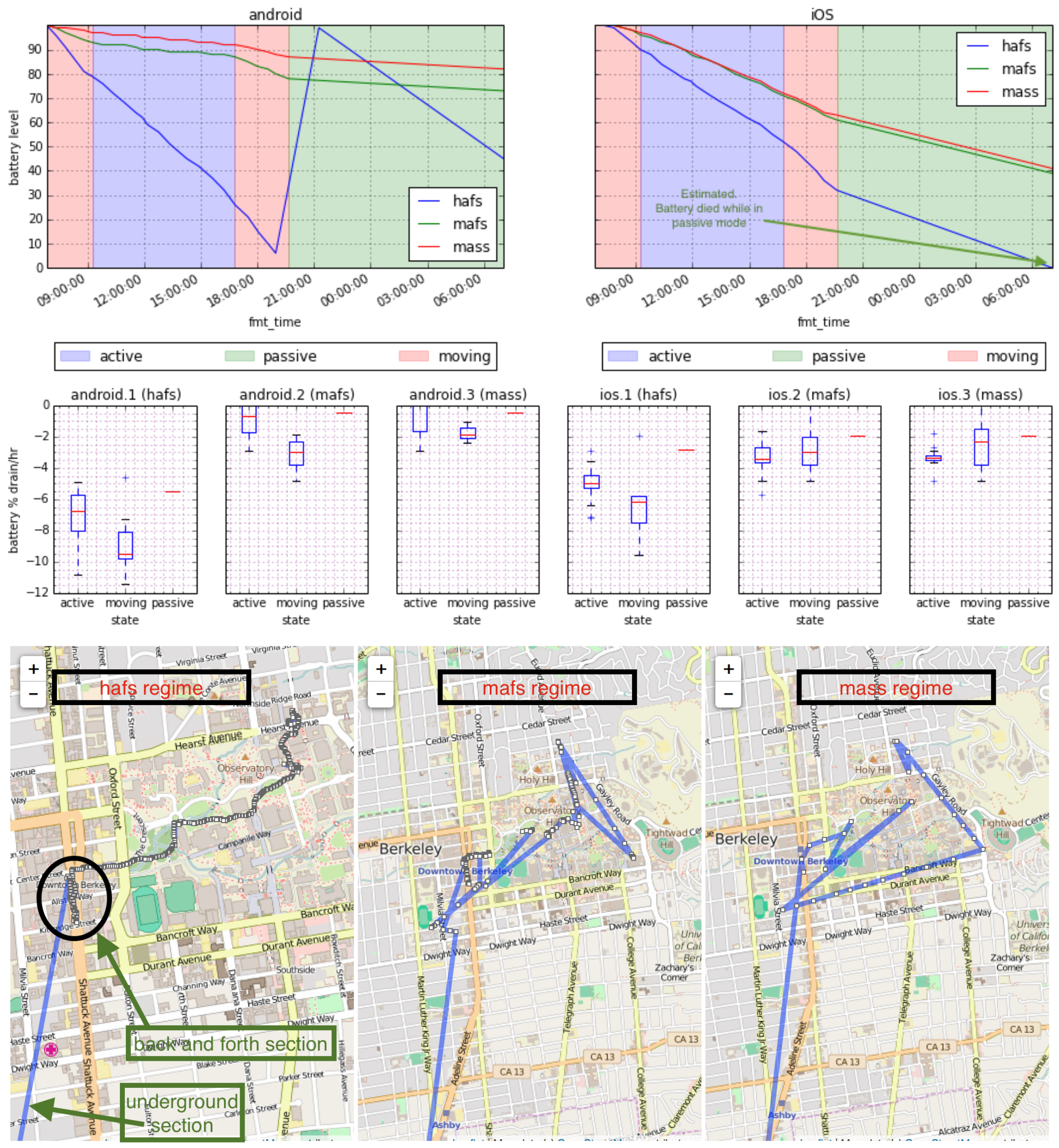


Figure 4.1. Comparison of three existing data collection regimes with no geofencing. Regimes are high accuracy fast sampling (`hafs`), medium accuracy fast sampling (`mafs`) and medium accuracy slow sampling (`mass`). The top graph shows the change in battery level over 24 hours. The middle graph shows the rate of drain in %/hr in the three states. The bottom map shows the data points collected by each regime and provides an intuition of what the different accuracy levels correspond to.

1. **Passive** The phone is not being actively used — it is stationary with the screen off. We expect that the phone will be in this state while the user is sleeping, for example. This corresponds fairly closely with the Android Doze mode. We expected that the phone would be in the passive state for most of the "night" cycle.

2. **Active** The phone is being actively used, but the user is not traveling. This state is key to our evaluation, since we can turn off tracking in this state and reduce background operation. Note that we detect that the phone is active even if the user is walking, as long as she does so within a small radius, like that of a building. We expect that the phone will be in this state for most of the "day" cycle.

3. **Moving** The user is taking a trip while carrying the phone. We expect that the phone will be in this state when we take trips during the day.

4.2.1.2 Data collection regimes

We primarily use the following data collection regimes to explore the range of behavior in each of the states above. Note that the details of the regimes are slightly different on iOS and on Android, since they use different filters. Each of the sampling regimes above is evaluated both with and without geofencing, which gives us six different data collection regimes overall.

1. **High accuracy, fast sampling** (2 s on Android, 5 m on iOS) (`hafs`)

2. **Medium accuracy, fast sampling** (2 s on Android, 5 m on iOS) (`mafs`)

3. **Medium accuracy, slow sampling** (30 s on Android, 100 m on iOS) (`mass`)

4.2.1.3 Metrics

Since our technique trades off power and startup accuracy, we use the following metrics to evaluate the two aspects of the trade-off.

1. **Power drain** We measure the power drain across different regimes running in parallel on the three phones. We look at both the *final battery level* at the end of 24 hours, and the *power drain in various states* under the regime. The power drain is represented using box plots — the center line is the median, the box represents the 25th to 75th percentile, the whiskers represent the interquartile range (IQR) and outliers are represented by individual points lying outside.

2. **Accuracy** We look at the distance between the actual start of the trip and the geofence exit location. We also inspect the distances between the geofence exit location and the first few points in order to estimate the loss in accuracy at the start of the trip.

Each of the power drain result figures represents data collected over one day to compare regimes against one another. The top graph in each figure represents the change in battery level over the course of the day, and the middle graph contains the box plots for the power drain in the various states. These graphs correspond to the power drain metrics above. The third row has a set of maps which provide a visual representation of the accuracy of the data collected.

4.2.1.4 Recording measurements

One of our challenges was to develop a technique for measuring battery levels that would not perturb the measurement. For example, it was not clear that we could use a power meter to measure power drain, since the OS only puts the phone into Doze mode when it is unplugged. Automatically polling for the battery life, in active or passive states, for example, would introduce new background processing at a time when our goal is to reduce or eliminate it. And automatic sampling means that states may overlap with samples, which makes it harder to calculate drain.

Therefore, we used the following low-tech solution to record the power drain.

1. **moving** state: Every 30 minutes, view the battery level on the phone screen and manually record it in a csv file. In addition, record entries at the start and end of every trip so that each time interval has only one associated state.

2. **active** state: We use the same technique as the **moving** state. This has the added advantage that it simulates the user interacting with her phone periodically, and ensures that the phone remains in the active state.

3. **passive** view and record the battery level at the beginning and end of the passive period. This ensures that there is no additional interaction with the phone, although it means that it is harder to isolate outlier points.

4.2.2 Exploration results

This section analyses the empirical results to answer the questions about the feasibility of the naïve sensing design. It shows that the sophisticated, context sensitive power management features work well in the absence of energy hungry background sensing, but they fail when confronted with high power background sensing. Virtual sensors such as geofences can be used duty cycle sensing at the app level.

4.2.2.1 Do phones really do nothing well?

Yes. Figures 4.2, 4.3 and 4.4 compare the power drain of continuous data collection against geofenced data collection and no data collection. So the "nd" line on the top-most graph in each figure represents the power drain with no data collection

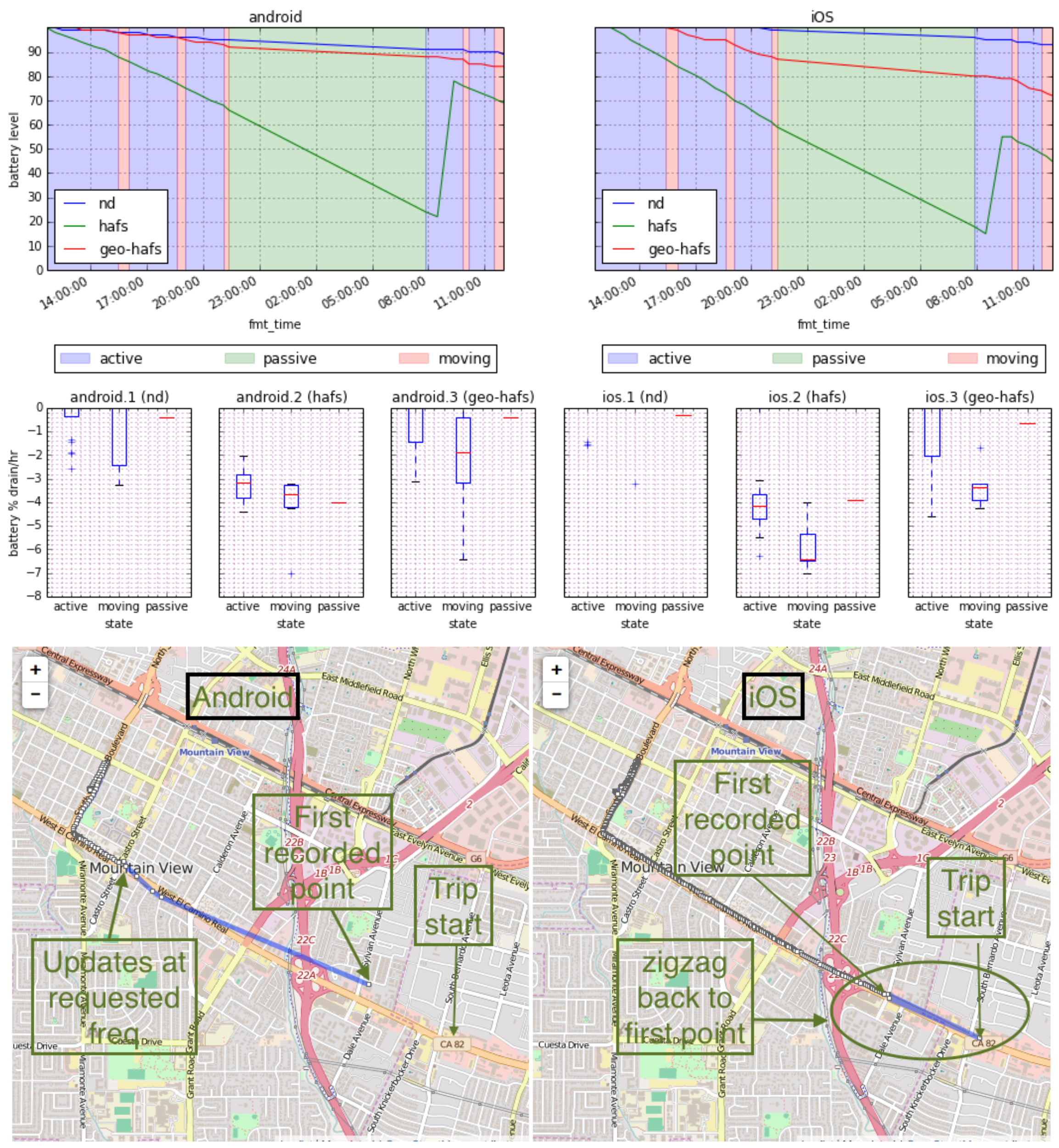


Figure 4.2. Evaluation of geofencing while collecting data with high accuracy. Regimes are no data collection (`nd`), high accuracy fast sampling (`hafs`) and geofenced high accuracy fast sampling (`geo-hafs`). The top graph shows the change in battery level over 24 hours. The middle graph shows the rate of drain in %/hr in the active, moving and passive states. The bottom map shows the extent of the error on each platform at the beginning of the trip.

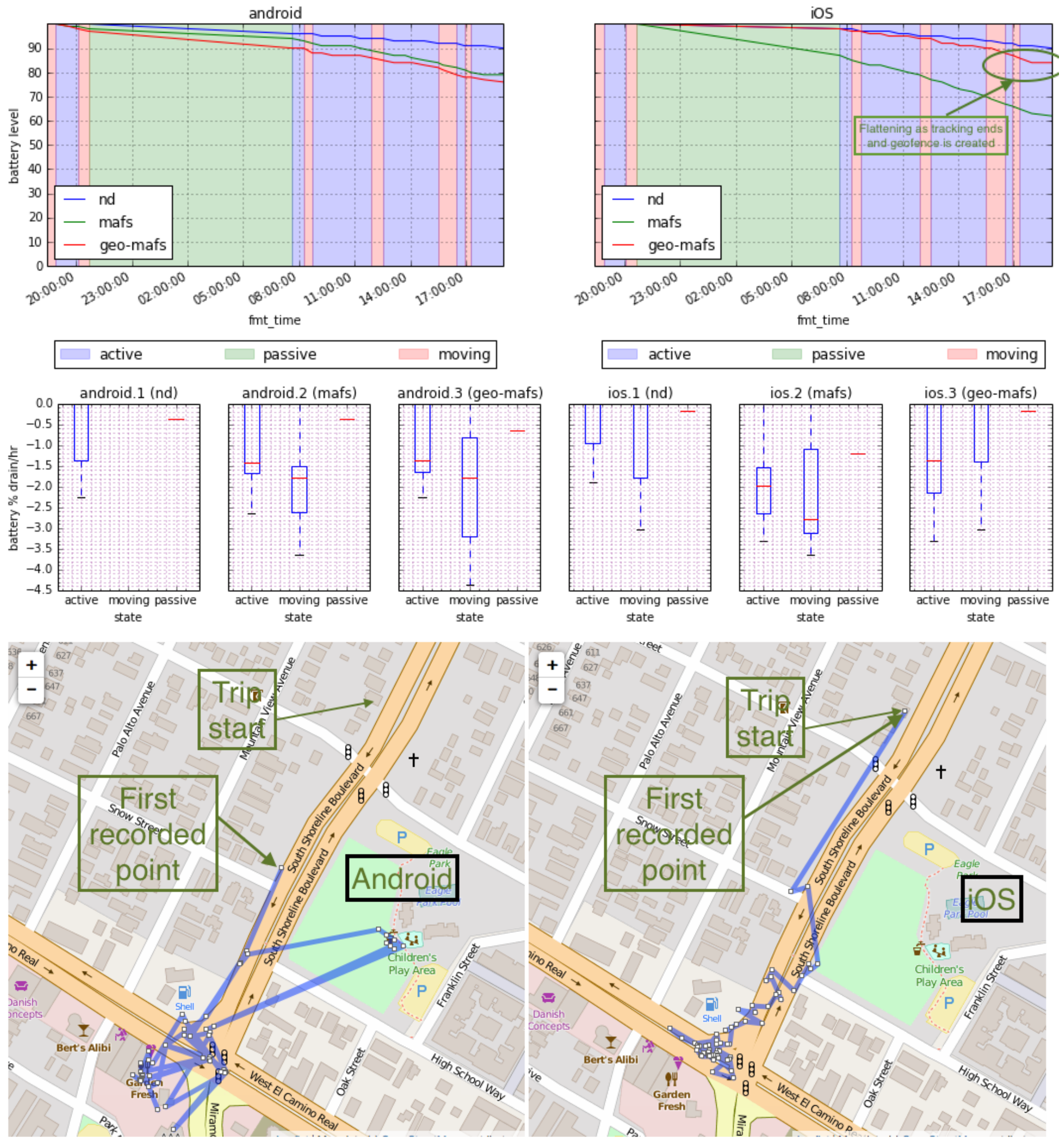


Figure 4.3. Evaluation of geofencing while collecting data with medium accuracy, but a fast sampling rate. Regimes are no data collection (`nd`), medium accuracy fast sampling (`mafs`) and geofenced medium accuracy fast sampling (`geo-mafs`). The top graph shows the change in battery level over 24 hours. The middle graph shows the rate of drain in %/hr in the active, moving and passive states. The bottom map shows the extent of the error on each platform at the beginning of the trip.

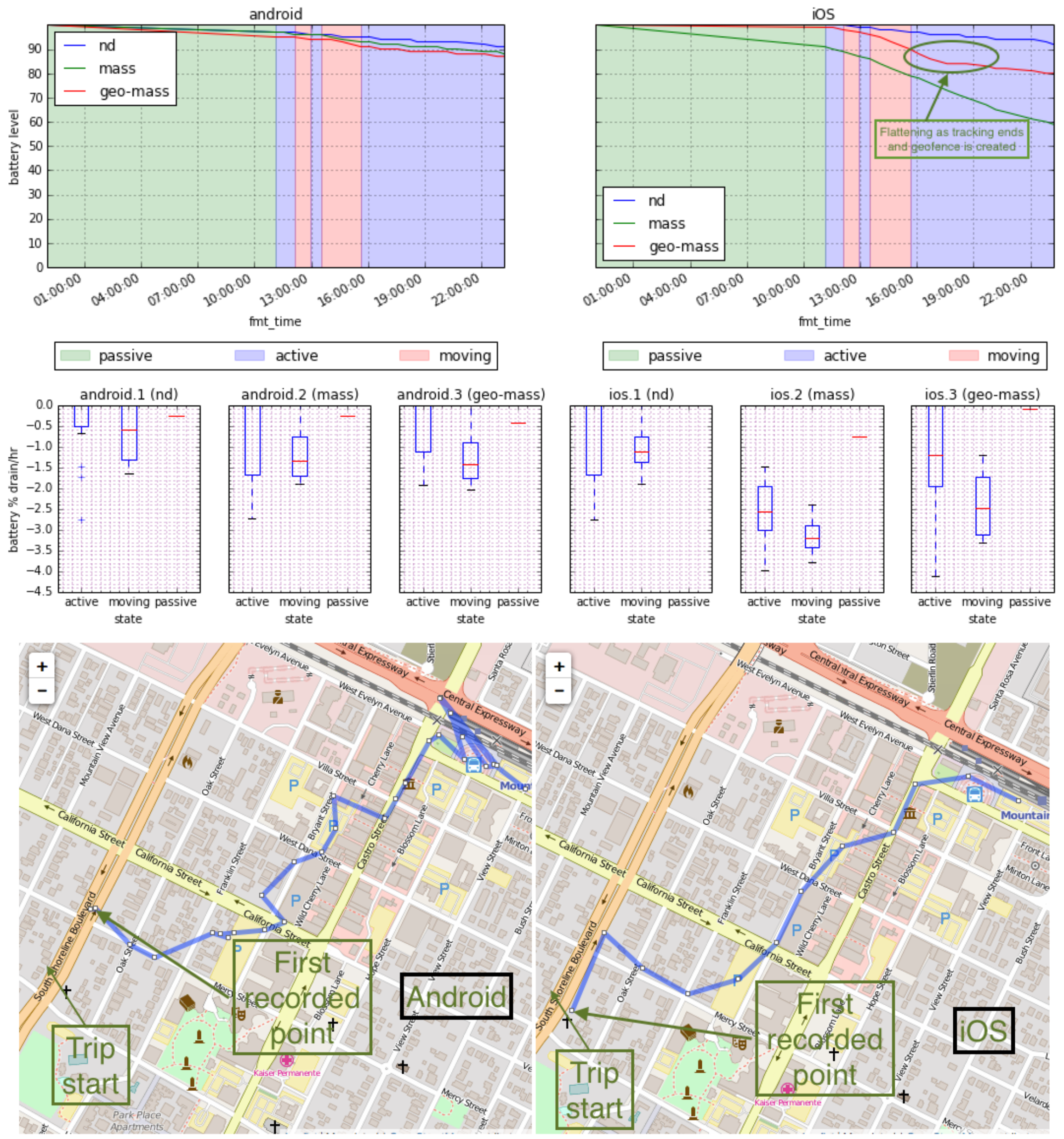


Figure 4.4. Evaluation of geofencing while collecting data with medium accuracy, and a slow sampling rate. Regimes are no data collection (`nd`), medium accuracy fast sampling (`mass`) and geofenced medium accuracy fast sampling (`geo-mass`). The top graph shows the change in battery level over 24 hours. The middle graph shows the rate of drain in %/hr in the active, moving and passive states. The bottom map shows the extent of the error on each platform at the beginning of the trip.

(stock phone) over 24 hours. We can see that the values for iOS and Android are 7% and 11% respectively. The 2019 results are 2x worse with similar drain over half the time ($\approx$ 12 hours (Figure 6.2)). This may be due to: (i) additional 4 years of battery deterioration, (ii) new system-level services, or (iii) coarse timer wakeups for automated battery readings in the new evaluation procedure (Section 6.3). But even 10% drain over a workday corresponds to $<$ 1% per hour, which seems to correspond with working well.

4.2.2.2 Is continuous sensing on mobile phones a solved problem?

Not with high accuracy. Figure 4.1 compares the power drain of three different continuous sampling regimes over the course of the same day. Note that moving from high accuracy to medium accuracy makes a significant difference. The high accuracy fast data collection resulted in out of battery on both platforms, but the medium accuracy sampling did not. It is also interesting to note that the filter size does not appear to make any difference on iOS — the lines for the fast and slow sampling are almost indistinguishable. On Android, the sampling rate seems to matter primarily during the *moving* state. During the *passive* state, the slopes of the lines are very close, and the divergence in the active state is small.

We can also see this from the box plots of the power drain rate - in all the Android regimes, the moving rate is noticeably higher than the active rate, which is in turn significantly higher than the passive rate. On iOS, we see a similar pattern for the high accuracy case, but for medium accuracy, there is not much difference in the power drain across sampling rates, or across states within the same medium accuracy regime.

Figure 4.1 also shows the trade-off in lower accuracy of the collected data points. The three trajectories shown were recorded at the same time on three identical iPhones. As expected, the high accuracy data collection is an accurate representation of ground truth — the duplicated points on Shattuck represent an actual back and forth section of the trip.

While the high accuracy data collection is clearly superior to both medium accuracy data collection regimes, the location accuracy required depends on both the algorithms that are used to process it, and the final application for which it is used. A determination of the optimal accuracy and sampling for different applications is outside the scope of this chapter and is addressed in Chapter 7.2 instead. We content ourselves with evaluating the effect of geofenced duty cycling on each of the three sampling regimes here.

4.2.2.3 Does geofencing help?

Yes. Figure 4.2 shows the effect of geofencing on the power drain with high accuracy sampling. With geofencing, we can obtain high location accuracy during the trip with a power drain that is close to no data collection. In fact, high accuracy data collection is not possible without duty cycling on either platform. Both platforms are remarkably consistent — duty cycling makes high accuracy data collection possible.

The picture is less clear if we are willing to tolerate medium accuracy. Figures 4.3 and 4.4 illustrate the differing ways in which geofencing affects medium accuracy sensing on the two platforms. Medium accuracy data collection is very efficient on Android — it appears to be similar to the power drain of geofencing. In fact, at the end of the day, the non-geofenced solution actually has a lower power drain than the geofenced solution. From the box plot, we can see that power drain in the active state is almost identical for both geofenced and non-geofenced operation. In the passive state, the power drain is actually slightly higher with the geofence. While the passive state has only one data point, the active state has several points and the result is consistent across both sampling rates. It looks like Android has figured out how to perform continuous, medium accuracy data collection very cheaply.

The story is very different on iOS, where geofencing is significantly cheaper than continuous sensing. Even with medium accuracy, the difference in battery level between geofenced and non-geofenced operation at the end of 24 hours is between 25% and 30%. The box plot shows that geofencing is essentially free in the passive state. The drain appears to be high in the active state, but at least part of that is because we are unable to detect the trip end immediately and have to wait until the next hour to stop tracking. A set of short trips causes us to spend a lot of time in *active but tracking* state, which increases the power drain. For example, note that the big drop in battery level during the moving state in Fig. 4.4 continues beyond the end of the trip, before flattening out as the tracking stops and the geofence is re-established.

Finally, although geofencing enables high accuracy data collection during the trip, it loses some accuracy at the start of the trip. Location tracking is started only after the geofence boundary has been crossed, so the points traversed to reach the boundary are lost. An interesting observation that we can see what looks like rate adaptive GPS tracking on Android in the `hafs` regime — after the geofence is exited, the points are generated at exponentially shorter distances, until they settle into the configured frequency. We do not see similar behavior in iOS high accuracy mode, and the low accuracy data on both platforms is so noisy that it is hard to determine what the correct points are, let alone their frequency.

4.3 Cross-platform duty cycling implementation

Results from an empirical evaluation of context sensitive power management features indicate that these features do not work well when confronted with high power background sensing. Instead, smartphone apps that want to capture high granularity, high accuracy location data need to perform their own duty cycling using

stat	drain % /hr	reporting service launches/hr
High accuracy, no filter	8.42	2722
Medium accuracy, no filter	0.25	4

Table 4.1. Comparison of Android sensing regimes when phone is stationary

virtual sensors. A finite state machine (FSM) can capture the transitions required for such duty cycling. This section uses the empirical results (Section 4.2) to outline the design of Android and iOS-specific FSMs for automatically triggered duty cycling.

4.3.1 Motivation

As we can see from Section 4.2.2.2 and Figure 4.6, there is significant power drain on iOS for ongoing tracking, even with a large distance filter to throttle updates. The power drain ranges from 1%/hr for the medium accuracy 100 m filter to almost 4%/hr for a high accuracy 100 m filter (measured separately) even when the phone was stationary on a desk. This would imply a 24-hour power drain of 24% for medium accuracy and 79% for high accuracy even when the location is not changing. The additional power drain for geofencing over stock phone operation is essentially zero.

On Android, the situation is more complex, since the medium accuracy mode uses Wi-Fi, and Wi-Fi scans are suspended in doze mode except for the maintenance windows. Table 4.3.1 shows that ongoing sensing is effectively duty cycled by default when the phone is in doze mode. So duty cycling would primarily help when high accuracy sensing is needed.

We also considered two other duty cycling approaches from the literature.

1. *Leave the accelerometer turned on and duty cycle other sensors if the user is stationary for a certain period.* Unfortunately, this will not work on iOS. Reading the accelerometer and/or activity detection results are not supported background modes. By piggybacking on the location tracking, we can potentially detect when the user has stopped moving, but we cannot detect when the user has started moving again.

2. *Change the location filter properties based on user speed.* Again, this is not likely to help in iOS because changing the distance filter does not appear to appreciably reduce the power drain. Reducing the sampling rate also does not appear to appreciably change the power drain on Android while in medium accuracy, although the power drain is low to begin with. While this might be an acceptable strategy for Android, we wanted to explore a consistent strategy across both platforms in which we just turn everything off.

4.3.2 Our design and some challenges

In our design, we use simple finite state machines to perform duty cycling (Figure 4.3.2). The core of the FSM are the `waiting_for_trip_start` and `ongoing_trip` states.

1. We use location updates to detect when the user is *loitering* or *dwelling* at a location. We will detect loitering even if the user is walking, as long as all movement is within the location radius. So even if the user is walking around the office, she is dwelling in the office.

2. Create a geofence at the current location and turn off all tracking.

3. When the geofence is exited, resume all tracking.

The other states deal with non-routine conditions such as the user manually turning tracking on and off, either through the app controls, or directly in the phone OS by turning off location services or location permissions.

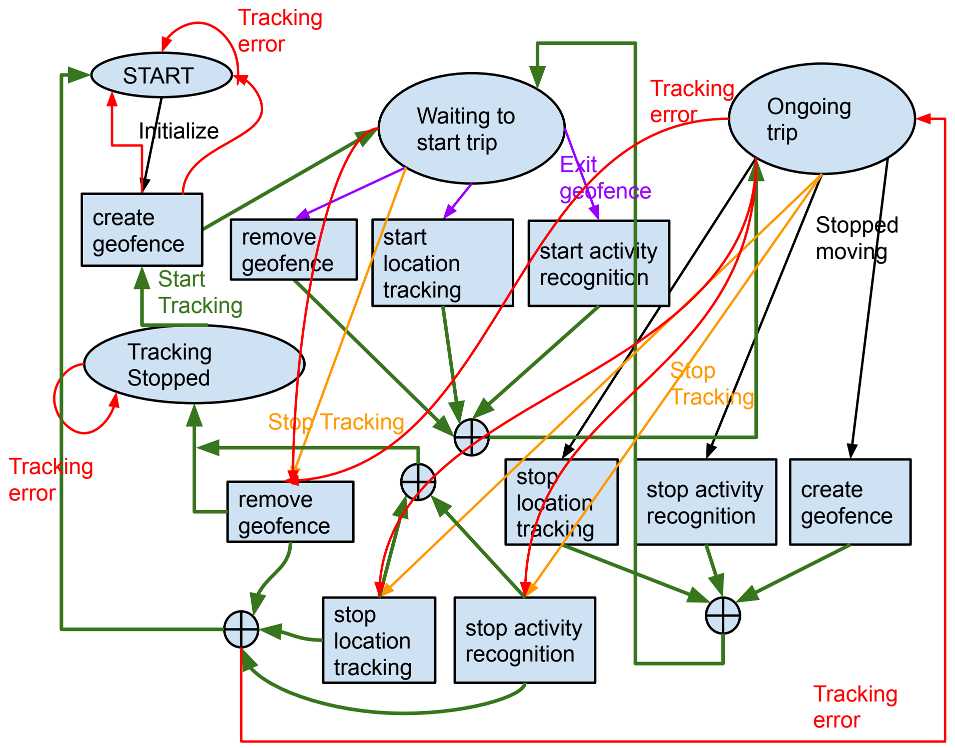
Now, we discuss three additional design challenges and their solutions.

4.3.2.1 Detecting dwelling with a distance filter

Detecting the end of a trip with a time filter is pretty straightforward. If the user has not moved more than distance d in the past t minutes, then end the trip. This has been the approach taken by most prior work, based on data from GPS devices. But on iOS, the only supported throttling option is a *distance* filter. So once a trip has ended, we will simply stop getting updates. The next update will occur when the user has traveled more than d. Depending on the value of d, this could well be at the start of the next trip, which means that no duty cycling will occur. If there was support for scheduling jobs at a future time, we could schedule a job to be run after t minutes, which would end the trip if there were no recent updates, but as we have seen, that is not a supported background mode. So we use remote pushes, to wake the app up periodically and check to see if the last received location was t minutes ago. Since remote pushes are scheduled on the server, and we do not want to require a network call for each location update, the remote pushes are scheduled to run every hour, with no app triggers or communication.

4.3.2.2 Detecting dwelling in the presence of noise

We originally assumed that the dwell-time algorithm would be robust to noise because the noise would die down eventually and the geofence would be created. This assumption about the noise model was incorrect, specially in medium accuracy mode. On Android, the medium accuracy collection would sometimes repeat the last known point if it had no new data. This would cause the algorithm to terminate trips



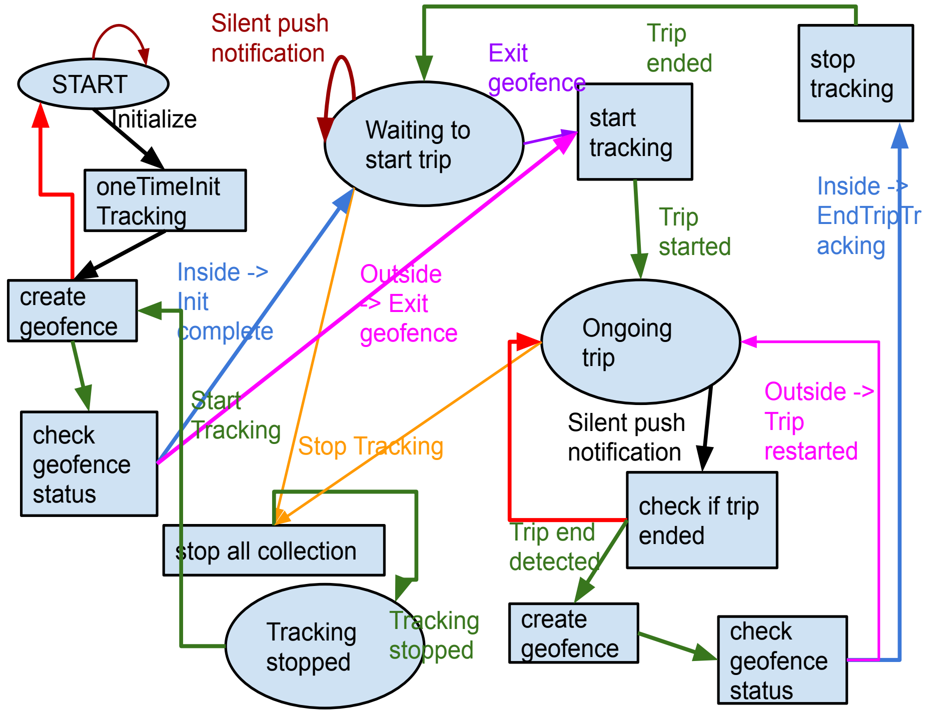


Figure 4.5. Finite State Machines for duty cycling on android (up) and iOS (down)

whenever we were traveling underground, for example. On iOS with medium accuracy tracking, spurious, low accuracy points would be generated outside the distance filter, which would cause a continuous set of updates between the current location and the spurious location. Also, we only check for trip end every hour, noise showing up at the wrong time can lead to tracking for an additional hour and wasted power. We address this by filtering both noisy points and duplicates on the phone before checking for the trip end.

4.3.2.3 Geofence creation quirks

1. iOS creates a geofence but does not start monitoring it in the background. So we need to wait for the geofence creation to complete before returning.
2. on iOS, creating a geofence while in motion is tricky because if the phone is outside the geofence by the time creation is complete, no exit event is generated. So after creation, we need to check whether we are inside or outside and transition states accordingly.
3. creating the geofence at the "current" location has some drawbacks — if the current location has low accuracy, we might create the geofence some distance away, and not trigger it while leaving.
4. android deletes all geofences when the location services are disabled. So if the traveler turns location services off when she is not moving, then tracking will stop and the FSM will be stuck in `WAITING_FOR_TRIP_START` forever. This does not change even if the user turns location services on later. To avoid this, we need to periodically recreate the geofence to ensure that it exists.

Additional details of the implementation are in the BSD-licensed library on github [2].

4.3.2.4 Summary

To summarize, designing a robust, efficient, cross-platform pervasive sensing app is a significant challenge, full of philosophical differences (background processing), subtle quirks (geofences) and undocumented error models (medium accuracy) that can only be discovered by implementation and testing. We now generalize the behavior of this solution to a wide variety of usage patterns and determine the population-wide ranges for sensing overhead.

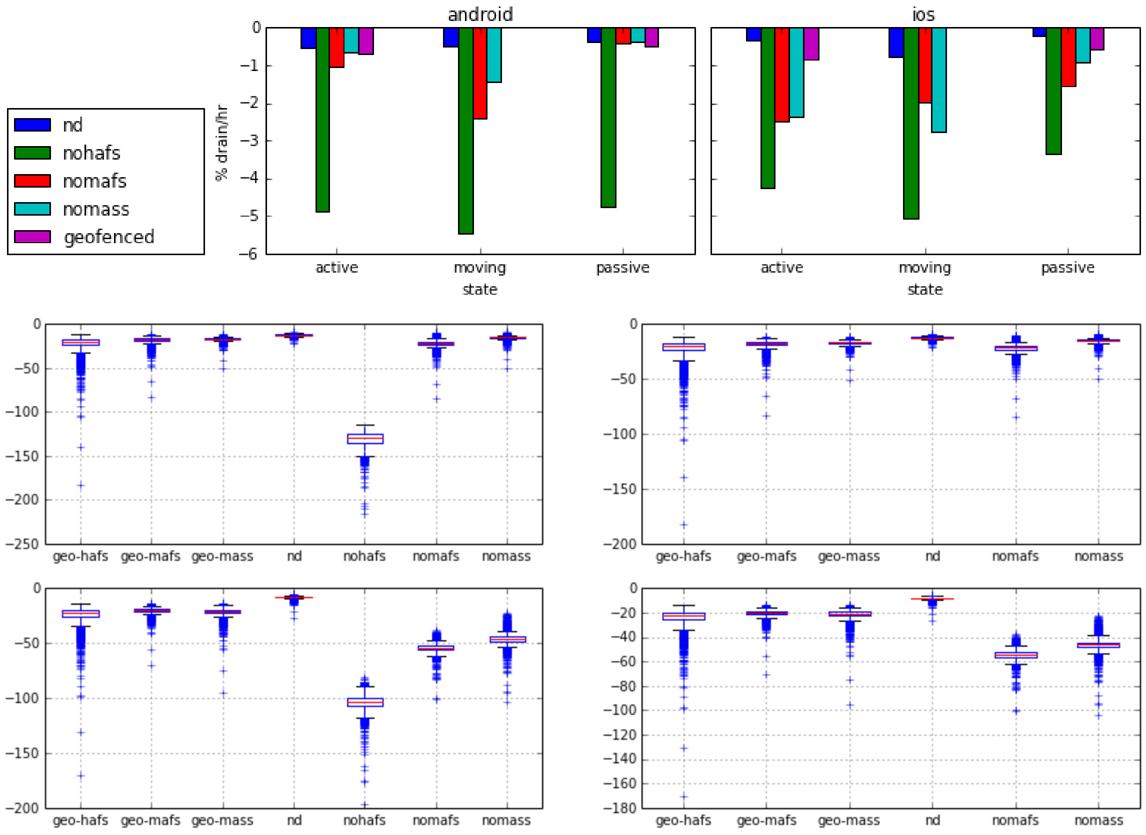


Figure 4.6. Results for generalizing the results to a broader variety of activity patterns. Top: a simple model for estimating the power drain as a factor of the no data collection `nd`, tracking using high accuracy fast sampling `nohafs`, tracking using medium accuracy fast sampling `nomafs`, tracking using medium accuracy slow sampling `nomass` and geofenced `geofenced` states. Bottom: Distributions of power drain (%/hr) on android and iOS generated by applying the top model to the ATUS dataset. L: with `nohafs`, R: without `nohafs` and an expanded scale

4.4 Modeling and generalization

The experimental results above are for a small, restricted set of travel patterns. Since our approach consists of lowering the power drain in the active and passive states, its performance is heavily dependent on the time spent in each state — if a user spends the whole day traveling, there will be no difference in power drain between our solution and the continuous data collection solutions.

So in order to complete our evaluation, we need to extend the results from the four data collection days above to typical days in the life of the general population.

We do this by building a model of the power drain in each state for different regimes and applying it to a large set of user activity patterns collected as part of the American Time Use Survey (ATUS).

The ATUS is a publicly available dataset collected by the Department on Labor that consists of a set of activity diaries which include coded activities, their start and end times and their duration for a randomly selected sample of the population. The 2014 ATUS data contains data from **11592** individuals, whose activities are coded into 17 major codes. The codes include both sleeping (code 1) and all forms of transport (code 18) — there is no category for code 17. For simplicity, we assume that people are interacting with their phones at any time that they are sleeping and not traveling, so we can easily map the major codes to our states.

Next, we combine the data collected above to build a composite model that has a power drain coefficient for each state. We do this by combining the entries from all time periods when that regime was active and calculating the overall mean. In particular, each of the continuous sensing regimes is modeled by considering data from the *accuracy versus sampling rate* data, the *ongoing regime* and the "moving" sections of the *geofenced regime*. Note that this results in 5 coefficients, since the drain for the geofenced modes is a combination of geofencing for active and passive, and a selected sensing mode for moving.

The resulting model is shown in Figure 4.6. The model appears to be consistent with our observations in Section 4.2 — the biggest power drain is in the `hafs` regime, geofencing is essentially free on iOS, and geofencing doesn't show much improvement over medium accuracy on Android.

We can then estimate the power drain over the day for every user for a particular regime by: (i) mapping the activity codes to states, (ii) summing up the durations in each state to obtain the percentage of the day spent in each state, and (iii) multiplying by the coefficients and summing to obtain the power drain across the entire day.

Note that for the geofence regimes, we use the geofence coefficient for the passive and active states, and the selected sensing regime for the moving state.

This gives us the distribution of power drains across the set of users for each regime. A box plot of these distributions is shown in Figure 4.6. Unsurprisingly, the

[2] https://github.com/e-mission/e-mission-data-collection

`nohafs` regime runs out of battery for almost all users on both platforms. In order to get more visibility into the details of the other regimes, we re-plotted the graph after excluding `nohafs`. From that graph, we can observe that:

1. The graphs are actually fairly consistent across platforms — the median drain for all geofenced regimes on both platforms is around 20%.

2. The major difference between the geofenced regimes is in the spread of the data — the medium accuracy regimes with geofencing have tighter bounds than the `geo-hafs` regime, and fewer outliers. Some outliers in the `geo-hafs` case are greater than 100%, indicating that in a few cases, the phone will run out of battery even with geofencing turned on.

3. The non-geofenced regimes are noticeably different — the median on iOS is around 50%, while the median on Android is at the same (15 – 20%) as the other data collection methods. This implies that by using medium accuracy on Android, it might be possible to get away without duty cycling. But it is clear that for iOS, any reasonable data collection solution must use some form of duty cycling.

4.5 Custom implementations and virtual sensor updates

Our sensing design (Section 4.3) uses virtual sensors such as geofences or the visit detection APIs on iOS for duty cycling. This section justifies the decision to use virtual sensors by comparing: (i) the flexibility vs. development cost trade-off, (ii) the liberation from restrictions for system services, and (iii) the maintenance and upgrade cycle.

4.5.1 Virtual sensors are easier to use but less flexible

Using the virtual sensors instead of re-implementing custom versions (e.g., [PFJM19, p.133]) reduces development and testing time. It leverages preexisting engineering efforts from the phone OS providers. Prior android-only comparisons between custom implementations and the built-in API indicate that the built-in API is comparable [CCC+14b] or slightly better (Figure 5, [CCC+14a]).

However, the built-in implementations can be less flexible than custom implementations. For example, turning off location access to the app can delete all associated geofences (Section 4.3), so apps that use geofences need to poll and re-create geofences when the coarse timer is triggered (Section 3.4). This could cause some trips to be lost. The OS does not delete explicit location tracking requests so location updates will be resumed without any gaps. Such challenges lead to a flexibility/power trade-off for virtual sensor use.

4.5.2 Virtual sensors have fewer restrictions, at least on iOS

Since the virtual sensors are part of system services, they may also have access to private APIs or be subject to fewer background restrictions than regular apps. For example, the motion activity API on iOS uses the M-series co-processor for low power operation[3]. User-level apps cannot schedule operations on the coprocessor, only the OS can, therefore even if user-level apps were not subject to background restrictions, they would likely have higher power drain than the built-in OS sensing. Similarly, iOS does not expose Wi-Fi scans to apps[4], so apps cannot create a custom implementation of the fused location sensors.

4.5.3 Virtual sensor implementations change over time

Both the virtual sensor implementations and any associated background restrictions change with new releases. For example, the *location virtual sensor* on Android in 2019 appears to have much more aggressive built-in location duty cycling (Figure 6.2) than 2015 (Figure 4.1). Similarly, the Android update to version 26 started delivering location updates only a few times an hour unless the data was requested by a foreground service[5].

Any sensing app that uses virtual sensors will need to be modified periodically to adapt to such changes. A modular architecture with a dedicated data collection module (Section 3.4) that communicates with loosely coupled notification modules (Section 3.3.2) allows e-mission to isolate other modules from these changes. A well-defined finite state module (Section 3.4.1, Section 4.3) simplifies the process of making such changes and reduces the chances of regressions.

4.5.4 Maintenance and upgrade cycle

For a concrete example, we outline the changes required to migrate from API 18 to API 26 on android. The OS changes[6] related to virtual sensors were: (i) runtime permissions, (ii) background service restrictions, and (iii) implicit broadcast restrictions.

The related changes were all in the client tier (Section 3.4), and in the sensing module (Figure 3.3) only. The complete update, including non-sensing changes, took

[3]`https://en.wikipedia.org/wiki/Apple_motion_coprocessors`

[4]cell tower signal strength, Wi-Fi signal strength

[5]`https://developer.android.com/about/versions/oreo/background-location-limits`

[6]https://developer.android.com/about/versions/oreo/android-8.0-changes.html

roughly a month of developer time and the related issue logged over 100 updates[7]. The changes to the sensing module were fairly involved, but only ≈ 500 lines of code (LoC) were affected[8]. They involved: (i) more error handling in the FSM, (ii) conversion of background services to foreground, and (iii) switching to explicit broadcasts.

None of all these changes are specific to *virtual sensors*. The runtime permissions change would have affected custom implementations, and they would have needed to convert their sensing services to foreground as well. This implementation had more such changes because of the number of sensors that it listened to. Similarly, a sensing method with only one sensor may be tightly coupled and not use broadcasts so may not need the explicit broadcast change. However, such tightly coupled implementations might have found it harder to reason about the additional error conditions without a well-defined FSM.

Although invocation sites for virtual sensors need periodic maintenance, the changes are reasonable (≈ 500 LoC and one month of work for an update after ≈ 10 releases) and do not appear to be significantly greater than similar custom implementations.

4.6 Conclusion and Future Work

Background sensing on smartphones requires duty cycling to be practical. There are multiple duty cycling methods that can work on Android but using geofencing as the trigger is the only feasible option in iOS. Geofencing allows high accuracy, high frequency data collection at an ≈ 15% penalty for the vast majority of travel patterns although there are large outliers. Virtual sensors have less flexibility, but lower development and maintenance cost. Since there are few guarantees, their behavior can change unexpectedly as part of the upgrade cycle. A well-defined finite state machine can clarify the duty cycling design and ensure that updates are robust.

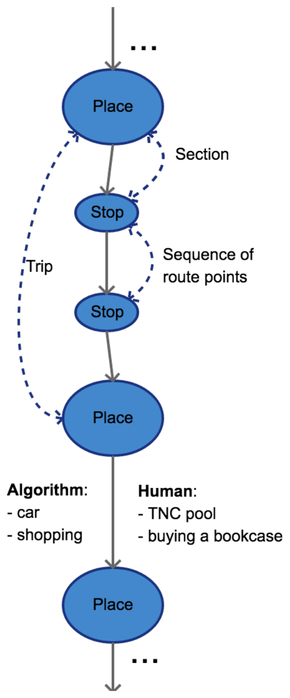
This chapter has focused on the sensing implementation, the restrictions on background processing and how to balance power and accuracy in background data collection. The next chapter continues this technical focus by examining the modules and algorithms required to convert this imperfect, noisy data into the mobility diary at the root of Computational Mobility.

[7]https://github.com/e-mission/e-mission-docs/issues/325
[8]https://github.com/e-mission/e-mission-data-collection/pull/170

Chapter 5

Wrangling noisy data into a mobility diary

The mobility, or travel, diary is the canonical data structure on which many Computational Mobility applications and models are based. It is generated by analyzing input data, both sensed and surveyed. The analysis consists of multiple steps, each of which has an algorithm associated with it. The steps are typically linked into an analysis *pipeline*. Some intake algorithms, such as mode inference, have been much better studied than others, such as segmentation. Like all data processing algorithms, the intake algorithms will never be perfect. However, reproducible analysis architectures can allow us to to quantify the error, and to compare algorithms against each other. In this section, we first outline the mobility data inputs, pipeline steps, and outputs in detail, with a special focus on data modeling for reproducibility (Section 5.1). Next, we outline the input data characteristics in detail, with a special focus on the unpredictable noise (Section 5.2). Next, we outline our adaptations to algorithms (i) that have been extensively studied in the literature, such as mode inference, (ii) that have been studied briefly, such as segmentation, and (iii) that do not appear to have been studied, such as dealing with untracked time. Finally, we outline some use cases for reproducible analytics as part of the conclusion (Section 5.4).



5.1 Reproducible data pipeline for diary creation

The mobility diary is the foundational data structure for computational mobility (Section 1.4.1), and represents longitudinal travel patterns for a single traveler. The input to the mobility diary is a combination of sensed and surveyed data, and the output is a linked sequence of *trips* and *places*, where each trip can potentially have multiple *sections* separated by *stops* for multi-modal travel The trips can have *algorithm-generated* or *user-specified* annotations. The diary is generated using an *intake pipeline* consisting of multiple self-contained processing *steps*.

5.1.1 Intake pipeline algorithm steps

The state-of-the-practice for complex, multi-step processing algorithms is to create small, self-contained processing steps and combine them into a *pipeline*, where the output of every step is the input into the next step. e-mission defines such a pipeline for converting raw data into a complete mobility diary.

The pipeline architecture and data model principles have already been outlined in the architecture chapter (Section 3.6). This section focuses on the concrete analysis steps included in e-mission, their inputs and outputs, and how they are composed to convert raw inputs into inferred outputs (Fig. 5.1).

More steps can be added later if the data collection is enhanced to include additional sensors. The algorithms for the steps, and the challenges associated with them, are discussed in greater detail in subsequent sections (Sections 5.3).

`segment_current_trips`: Converts the raw inputs and splits them into a linked sequence of *raw trips* and *raw places*. If the underlying sensing algorithms automatically turn tracking on and off (Section 4.3, Section 4.2), then the first few points in the trip will be missing. Also handles the detection of *untracked* time, which could be due to the user using various software settings to turn off tracking, or by turning the phone completely off.

`segment_current_sections`: Converts trips into a linked sequence of *raw sections* and *raw stops*. Each section represents travel by one mode — multi-modal trips will consist of multiple sections while unimodal trips will consist of only one section.

`filter_current_sections`: Location data can frequently be very noisy, particularly when the GPS sensor is turned on and is attempting a fix from satellites. When fused virtual sensors are used, this may happen multiple times during a trip, particularly during underground sections. This step identifies these erroneous points so that they can be removed. It is run after the section segmentation step because it uses speed outliers to determine invalid points, and speeds can only be expected to be consistent within a particular section. Unlike prior

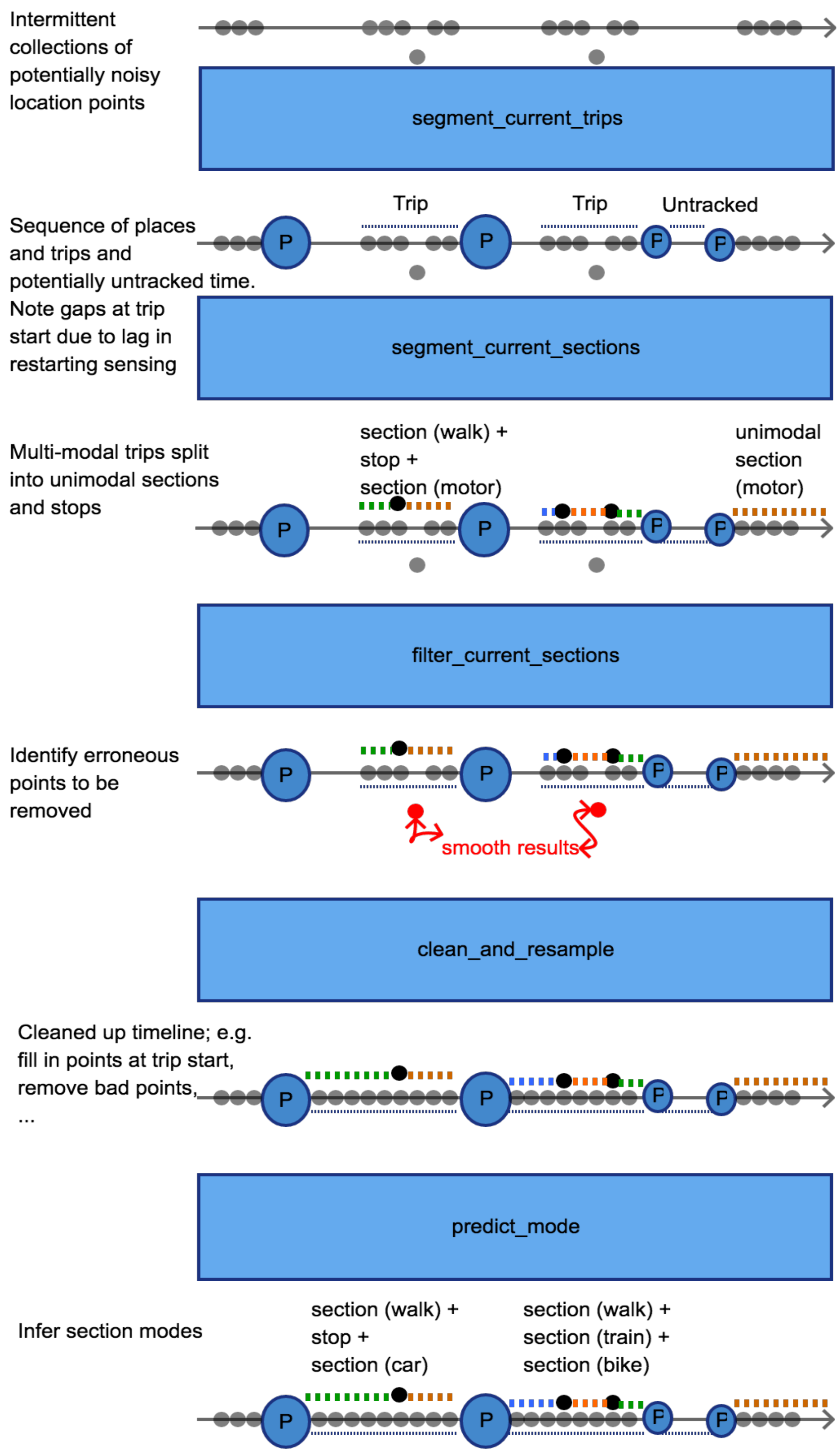

Intermittent collections of potentially noisy location points
segment_current_trips
Trip
Trip
Untracked
Sequence of places and trips and potentially untracked time. Note gaps at trip start due to lag in restarting sensing
P
segment_current_sections
section (walk) + stop + section (motor)
unimodal section (motor)
Multi-modal trips split into unimodal sections and stops
filter_current_sections
Identify erroneous points to be removed
smooth results
clean_and_resample
Cleaned up timeline; e.g. fill in points at trip start, remove bad points, ...
predict_mode
Infer section modes
section (walk) + stop + section (car)
section (walk) + section (train) + section (bike)

Figure 5.1. Illustration of how the timeline evolves as it proceeds through the pipeline.

work [YCP+13], we do not have domain knowledge of the moving object before section segmentation.

`clean_and_resample`: Puts the results of the previous steps together to generate spatiotemporally consistent mobility diary by filling in gaps if necessary. This diary can be used for additional inference about the trips (e.g., mode) and places (e.g., purpose). In addition to purely spatiotemporal data, since mobility mode transitions were determined during section segmentation, the resulting sections can be tagged with basic mobility modes.

`predict_mode`: Use inference algorithms to determine the mode for each section in the mobility diary. The accuracy of inference algorithms typically varies widely across modes; walking is easy to detect since walking speeds are limited, and significantly slower than the alternatives. Mobility modes with similar speed characteristics (e.g., bicycling and buses on city streets) are much harder to distinguish. And some modes, such as carpooling or ride-hailing are essentially indistinguishable through location data alone.

5.1.2 Other steps: data manipulation and use-case specific

In addition to the high level pipeline steps above, the e-mission pipeline has steps for data manipulation and specific use cases. While these are not necessarily generalizable, we briefly outline them here for completeness. They are also likely to change as we pare down the core and simplify the architecture.

5.1.2.1 Data manipulation steps

The data manipulation steps mainly deal with formatting the data during input and output and moving it between the input and output caches.

moveToLongTerm: Formats incoming messages based on the message type and moves them from the input cache into the timeseries database.

storeViewsToCache: Exports a geojson representation of the most recent days in the timeline to an outgoing cache. This representation is periodically pulled by the phone for faster local access.

5.1.2.2 Use case specific steps

The use case specific steps deal with particular use cases that we thought would be relevant but did not turn out to be so in practice.

filterAccuracy: If the duty cycling is turned off on the phone, then we don't need to filter the locations on the phone and the phone app only uploads the `background/location` stream. But the rest of the pipeline expects the filtered stream. So in case the phone doesn't filter, this step does the filtering. This is surprisingly slow, and most clients don't use it, so it is currently only enabled via a config file option.

give_points_for_all_tasks: One of the use cases for e-mission involved integrating with Habitica, an open source Role Playing Game (RPG). The RPG allows travelers to create point-awarding *tasks* which can then be automatically checked off by the integration with e-mission. This step does the check and awards RPG points if necessary.

5.1.3 Data model for reproducibility

The pipeline data model consists of immutable input data which is transformed through the pipeline steps to generate the final output. Each stage assumes that its input is immutable. The output of each stage is the input to the next stage (Figure 5.1). Types for each data object are represented in the metadata, by the `metadata.key` field.

Input types: Sensed input types are typically represented by `background/*` keys, e.g., `background/location`, `background/battery`, `background/motion_activity`. Manual input types are typically represented by `manual/*` keys, e.g., `manual/incident` or `manual/mode_confirm`.

Initial segmentation: Preliminary segmentation outputs are represented by `segmentation/*` keys, e.g., `segmentation/raw_trip`. These represent a first pass at segmentation so that we can detect and remove outliers. Note that the segmentation outputs include `segmentation/*_untracked` which represents time at which tracking was turned off. Tracking could be turned off because the user disabled it, or because the device was turned off.

Outlier detection: The outlier detection step is a bit different since it stores the erroneous points for a section. This implies that the final segmentation will use inputs from multiple previous stages — initial segmentation and outlier detection. The outlier detection results are stored in `analysis/smoothing`.

Final segmentation: Final segmentation outputs are represented by `analysis/cleaned_*` keys, e.g., `analysis/cleaned_trip`. These are the entries that most consumers will use. In addition to segmentation results, we also generate filtered, resampled data points, represented by `analysis/recreated_location`

Mode inference: The results of the mode inference are stored as a separate object `analysis/inferred_section`. The correspondence between inferred and cleaned sections is 1:1.

5.1.4 Pipeline states

While pipelines are common in traditional data models, the examples typically work on static datasets. In our case, our focus on data collection implies that data comes in periodically, and the pipeline needs to be run periodically to process this fresh data. We enforce this through the notion of a pipeline *state*. Further, since the data model regenerates analysis results from immutable input data, we need to be careful about duplicate entries. Therefore, pipeline states are used for three main purposes:

avoiding slowness: the state needs to include an indication of how much data has already been processed. Without this state, the pipeline would need to run from the beginning every time. This would quickly lead to unacceptable performance in the face of continuously increasing data.

avoiding concurrent runs: the state needs to indicate whether this would be a concurrent run — whether the pipeline is attempting to execute the same step with the same algorithm for the same user. Without this state, the next stage would see multiple trips from the same algorithm for the same time range, which is likely to confuse it.

avoiding re-processing: maintaining the state of how much data has already been processed is tricky. The naive solution would assume, on every run, that that all data upto the current time. But that doesn't account for delayed or asynchronous data transfer. Consider the case in which a user completes a trip at 11am but does not have network connectivity at her destination. Her data is only uploaded after she returns home at 5pm. But the pipeline has run at noon, 1pm, . . . 4pm. So the naive approach would assume that all data up to 4pm had been processed, when in fact, the 11am data had not even arrived at the server and could not have been processed. In order to avoid this, each pipeline step needs to store the timestamp of the last data point that it successfully processed. As a bonus, this also ensures that if there is an error or exception in a pipeline step, processing will continue from the correct point once the error is fixed.

5.2 Input virtual sensor data characteristics

Much of the prior work on travel diary creation has involved manually implementing context-sensitive data collection [SAMF16b, ZGP^{+}15] or activity detection [FT13, LV16, YYW^{+}14].

Possibly based on the prior work, smartphone OSes have now incorporated closed source implementations of several of these techniques and exposed them as *virtual sensors* for apps to use. In some cases, the virtual sensors are the only interface available; in others, they provide a convenient default implementation for common functionality.

For the initial version of this platform, we have focused on using the virtual sensors wherever possible. This allows us to leverage existing engineering effort and potential freedom from system restrictions (Section 4.5) in the basic sensing implementations. A rigorous evaluation (Section 7.2) shows areas for improvement similar to [ZWHI15] but may involve trade-offs in terms of additional edge computing. The platform's modular architecture (Chapter 3) allows us to evaluate and incorporate incremental improvements as they are developed.

The sensing deep-dive (Section 4.1) described virtual sensors in terms of their exposed APIs and documented restrictions on usage. This section describes the resulting data quality based on empirical results and exposes the challenges that the analysis algorithms need to overcome.

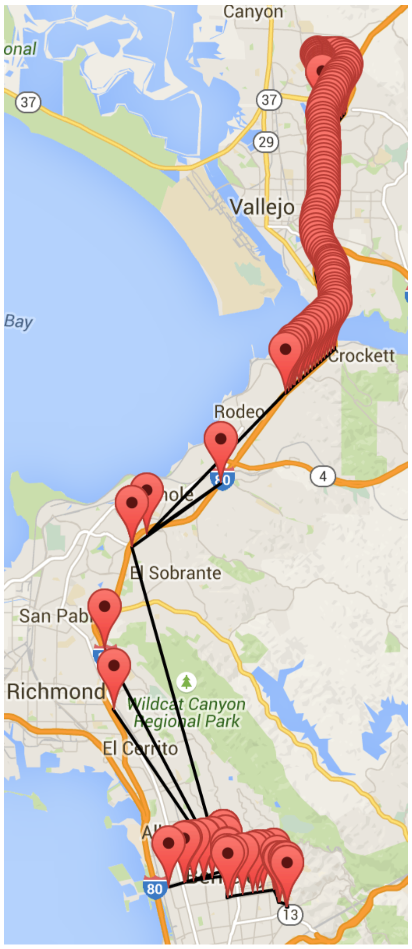


Figure 5.2. Quality change

5.2.1 Fused location

The fused location APIs provide access to phone location data based on a combination of localizable sensor data. This typically involves A-GPS, Wi-Fi and cell towers, although they may support Bluetooth as well. API users can specify the desired accuracy and frequency. Android uses a time filter to specify the frequency (e.g., `1000 ms`) but on iOS uses a distance filter (e.g., `10 meters`). This makes trip end detection harder on iOS (Section 4.3).

Both the configuration options are "hints". The location API performs sensor batching under the hood and the points exposed depend on the other apps requesting location at the same time. For example, in Figure 5.2, the traveler launched a navigation app while collecting data at medium accuracy/medium frequency, causing the data quality to dramatically improve without changing the configuration. The points have an accuracy radius, but it can be incorrect, specially around underground sections (Figure 6.1). The generated points do not explicitly indicate the source of location data (e.g., GPS or Wi-Fi).

5.2.2 Trip start/end

Geofencing The geofencing APIs on both android and iOS can be used to determine trip start. The sensing regime can creating a geofence at the end of a trip and listen for the exit. Geofences are recommended for use [1] since they are very power efficient (Figure 4.2). Geofences on android can be configured with radius and responsiveness settings which control the sensitivity/power trade-off. In keeping with the lack of time controls on iOS, geofences on iOS only support a radius setting. Further testing shows that the radius setting is also only a hint — e.g., Figure 7.3 does not report a geofence exit even after several km of travel.

Visit detection iOS also has a separate API for detecting the start and end of a visit to a place. This can also be used to detect trip start and end, since a visit end is a trip start and vice versa. The visit detection API has no configuration settings and appears to use a different implementation than the Geofencing API since they don't trigger synchronously (Figure 7.3).

5.2.3 Motion activity

Both Android and iOS also support activity detection APIs that can distinguish between active transportation modes (walk, run, bike) and a single motorized mode.

On Android, motion activity updates are provided to the app when tracking is on, and the app can specify the update frequency. On iOS, updates are only provided to the app in real-time when it is already executing; activity updates are not a supported background operation (Section 4.1). The OS also stores a history of activity updates which can be queried by the app when it does get a chance to run. It does not have a setting for the update frequency. Although android has an update frequency and iOS does not, the update frequency on android is also a "hint". Empirical testing shows that, at least on recent versions of the API, updates are only generated on activity transitions (Figure 7.10).

The raw accuracy of these sensors is poor. Prior work has established this for android [ZWHI15], but the API for iOS has not been studied before. Empirical evaluation (Figure 7.9) shows that iOS accuracy is also poor and is characterized by multiple flip-flops, specially for bicycling and public transit trips (Table 7.5).

5.3 Adaptations to classic algorithms

Most of the known post-processing algorithms have been designed for designated GPS devices(Section 2.4). Using smartphones instead requires adapting them to

[1]https://developer.android.com/training/location

match the changed data characteristics. These changes can be both beneficial (e.g., built-in activity detection) and challenging (e.g., poor quality location trajectories with variable frequencies). To the best of my knowledge, while prior work has used smartphone location entries, this thesis is the first to use the full range of virtual sensors in travel diary processing.

This section summarizes the key algorithm changes for each of the pipeline steps. The entire implementation is open source[2] and can be consulted for further details.

5.3.1 Trip segmentation

At a high level, the trip segmentation mainly uses a standard dwell time algorithm with a configurable dwell time. Some additional challenges encountered, and the ad hoc heuristics to solve them are:

untracked time The user can turn off tracking at any time, either (i) through the in-app option, (ii) by turning off phone level location services, or (iii) by turning the phone completely off. Detecting in-app disabling is trivial, but the other two are more complicated. Android receives an event when the phone is rebooted, but iOS does not.

If there is a large gap between two consecutive points, it is challenging to determine whether it was due to invalid entries or because tracking was turned off. Incorrectly assuming that it was due to invalid entries could lead to trips that span multiple days, but incorrectly assuming that it was due to the tracking being turned off could lead to aggressive segmentation and prove problematic for GIS integrations. We use a reasonable travel time heuristic to determine untracked time from such gaps and store it as a *trip-like* object.

Distance filter Using a dwell-time algorithm with a distance filter is as challenging on the server as it is on the phone. Concretely, we are not guaranteed to receive points after we have stopped moving until the next trip starts, which means that the last trip of the day may not be processed until the next day. We use visit notifications as a secondary source and terminate trips on visit starts, even if they don't match the dwell filter.

Switching phones The traveler may buy a new phone and end up switching from a distance filter to a time filter or vice versa. We handle this by: (i) creating two separate modules for time and distance filtering, (ii) storing the filter type for each location point, and (iii) filtering the locations to be sent to each module based on the type.

[2]`https://github.com/e-mission`

5.3.2 Section segmentation

At a high level, section segmentation uses the transitions between modes in the activity entries (Figure 7.10) as the section segmentation boundaries. We ignore irrelevant modes such as `TILTING` and only consider entries above a certain confidence threshold. Some additional challenges encountered, and the ad hoc heuristics to solve them are:

Matching activities to locations The activity transitions are generated in a completely different stream than the location points, so the two sets of entries are not synchronized.

1. We handle this by treating a stop as another segment with enter and exit locations. The mode changed somewhere within that range, but we lack the precision to determine where.
2. For short trips at low frequencies, the activity transitions and the location points are sometimes completely disjoint. In this case, we need to extend the activity to the extent of the trip.

Boundary selection Once we have detected the transitions, we need to determine whether and how they should correspond to sections. For example:

1. If we detect a change at m_{i+1}, do we end the section at m_i or at m_{i+1}? We define some rules around this based on ad hoc data contributed by users. For example, for transitions from *motorized* $\rightarrow$ *non-motorized*, we transition at m_{i+1}, for the reverse, we transition at m_i.
2. How do we deal with flip-flops? Sometimes, we get multiple transitions during what should be a single section. These are typically very short, in the order of seconds or minutes. We need to smooth over them but not miss real mode transitions. At a high level, our algorithm for this involves: (i) identifying flip-flops (ii) combining them into streaks, and (iii) merging the streaks to generate an unbroken section. We use heuristic rules such as the length of the section and the speed differential for these steps.

5.3.3 Trajectory smoothing

At a high level, trajectory smoothing (i) uses the interquartile range of the speeds in a particular section to determine outlier points and (ii) selectively removes them using a heuristic that can resolving zigzags (Figure 5.3).

The first step is straightforward. The second is complex because the zigzags are frequently between fairly short point subsets (e.g., 1–2 points) on each side and the outlier detection only determines that some subsets have to be filtered out.

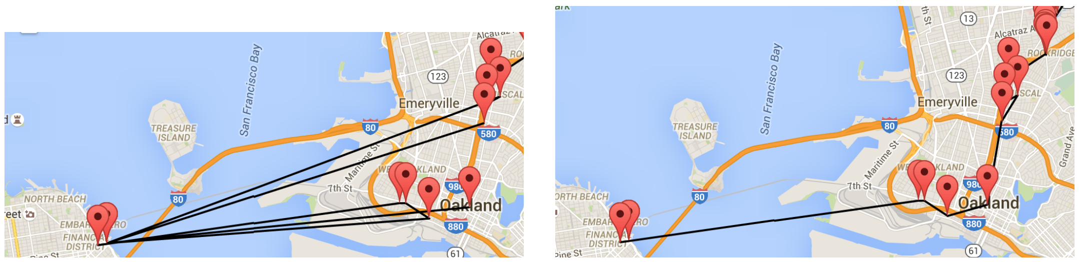


Figure 5.3. **Example of zigzag removal**. l: before smoothing, r: after smoothing

The zigzag smoothing algorithm leverages the structure characteristic of the zigzag — we expect to find a set of i good points, followed by a jump to a set of j bad points, followed by another jump to a set of k good points, ... So our algorithm becomes: (i) find the jumps, (ii) find the segments between the jumps, (iii) categorize the segments into good and bad, (iv) remove the bad segments. Its main limitation is with jumps at the beginning of a section, where we do not have sufficient travel history to determine a jump or categorize segments.

5.3.4 Cleaning and Resampling

The cleaning and resampling algorithms perform several finishing steps, such as:

Smoothing and resampling applies the results of the previous smoothing algorithm to remove outliers and then resamples the trajectory at a fixed interval for downstream consistency.

Extrapolating trip/section start extends the start of the trip, and by extension, of the first section to account for the gap in detecting the trip start. This has to be done after the smoothing so that we can determine a consistent speed to use. The trip must start from the place where the prior trip ended, so we can adjust the start time based on the additional distance and the section speed.

Detecting and squishing spurious trips Older versions of Android and Google Play Services had a persistent issue with spurious geofence exits. These were self-correcting, since we would quickly reach the dwell-time threshold, but such spurious trips could be confusing to end users. The spurious trips typically included a jump, which should be smoothed out by the outlier detection algorithm. Therefore, this stage can detect such spurious trips and squish them. So one `cleaned_place` can represent multiple `raw_places`.

5.3.5 Mode inference

We support two different mode inference algorithms — random forest and rule + GIS based.

1. The **random forest** is the more traditional option, with feature selection largely based on [ZCL+10]. It can distinguish between the classic 5 modes — walk, bike, bus, car and train. It requires a trained model, which is hard to obtain in the absence of a large reference dataset. However, once it is provided with a model, it can perform all computation locally and is very fast.

2. The **rule+GIS based** algorithm uses different techniques for motorized and non-motorized sections.

   (a) The *non-motorized* sections can easily be distinguished based on speed.

   (b) The *motorized* sections are classified by querying OpenStreetMap (OSM) using the Overpass API to determine whether the start and end points correspond to transit stops. Additional checks on the routes serving those transit stops reduce false positives and allow more fine-grained classification. Using the OSM labels on the routes, this algorithm adds support for tram, metro and subway modes as well.

   This approach clearly generates higher quality inference, but it requires access to an external service and is subject to all the reliability and robustness challenges that remote communication entails.

5.4 Conclusion

Reproducible data analysis involves designing a well-defined analysis pipeline such that the same analyzed results are generated for multiple runs with the same inputs (Section 3.6). The benefits of such data analysis include:

comparative evaluation multiple potential analysis algorithms can directly compared against each other by running them on the same inputs and comparing the results against standard metrics (Section 7.2),

ensemble methods instead of choosing "the best" algorithm, the results can be combined using ensemble methods such as bootstrapping. Alternatively, the ensemble result can be used to decide when to prompt the user for clarification.

debugging if there were errors in processing, such as unexpected inputs, the related data can be submitted to the algorithm expert, debugged carefully, and the fix tested before checking in. This has been a popular use case of our pipeline in the real world [3].

[3]e.g., https://github.com/e-mission/e-mission-docs/issues/461, https://github.com/e-mission/e-mission-docs/issues/322

We have built a reproducible analysis pipeline for spatiotemporal data. The inputs to the pipeline are *virtual sensor* readings from both android and iOS. The algorithms are modified versions of the classic travel diary processing algorithms, modified for smartphone sensors. They include

trip segmentation using a dwell-time filter with heuristics for determining untracked time,

section segmentation using the activity recognition phone APIs with flip-flop smoothing,

trajectory smoothing using a novel algorithm to detect and remove zigzags, and

mode inference with random forest and GIS rule engine implementations.

The innovations enabled by reproducibility require standardized datasets to operate on. While our initial sensing regime was also data-driven, the evaluation procedure and metrics were ad hoc (Chapter 4). In the next two chapters, we use the lessons learned from building a reproducible pipeline on ad hoc data to define a reproducible evaluation procedure (Chapter 6) and clear metrics (Section 7.2).

Chapter 6

A technique for evaluating mobility sensing

Inspired by the popularity of smartphone-based personal fitness tracking, the transportation community aims to build Human Mobility Systems (HMSes) that can automatically track and classify multi-modal travel patterns. Such systems can replace expensive and infrequent travel surveys with long-term, largely passive data collection augmented with intermittent surveys focused on perceptual data.

While there has been much work on building HMSes, both in academia and in industry, the procedure to *evaluate* them has largely been an afterthought. Careful evaluations are critical as we move from the personal to the societal domain. Users who make decisions based on self-tracking have an intuition of its accuracy based on their experienced ground truth. The decisions are low-stakes lifestyle changes, which may be personally meaningful, but are not societally contentious. However, a Metropolitan Transportation Agency picking projects and allocating millions of dollars in funding needs to know the accuracy of the data before making its decisions [VS15].

Computational Mobility (CM) (Chapter 1) can frame approaches to evaluate accuracy. Consistent with interdisciplinary research principles, we can apply computational concepts to shift the focus from ad hoc techniques to more rigorous concepts for evaluating software systems and algorithms. These concepts include: (i) *artificial workloads*, from Operating Systems, (ii) *standard datasets*, from Machine Learning, and (iii) *handling transient effects*, from Networking.

The rest of this chapter is structured as follows. We start with an intuitive description of the challenges and the solution in Section 6.1. Next, we outline the evaluation requirements in Section 6.2 and outline an experiment procedure that meets them in Section 6.3. We discuss some alternative approaches in Section 6.4 and describe the reference implementation in Section 6.5, concluding with Section 6.6.

6.1 Intuition for challenges and solution

The typical HMS evaluation procedure (e.g., Quantified Traveler [JAC+15], prior versions of our work [SYCK15, SBM+18], Chapter 4) is ad hoc and also functions as a pilot — a small ($\approx$ 3-12) set of the author's friends and family are recruited to install the app component of the HMS on their phones, and go about their daily life for a few days or weeks while annotating the trips with "ground truth". The ground truth annotation can either directly happen on the app, or through a recap at the end of the day. Conscientious researchers may ensure that the set of evaluators are demographically diverse, in an attempt to evaluate against a richer set of travel patterns.

While this procedure imposes little additional researcher burden, it conflates the *experimental* procedure (understanding human travel behavior in the wild) with the *evaluation* procedure (evaluating the instrument that will measure the human travel behavior). The first is trying to understand *behavior*, so it needs human diversity. The second is trying to understand *sensing* parameters, so it needs diversity of *trip types*. The human functions as a phone transportation mechanism during evaluation and could be profitably replaced with a self-navigating robot if one was available.

An analogy with classic physical measurements may be useful. Consider the situation in which a researcher wants to collect data on the weight distribution of the population in a particular region. Since there are currently no certifying bodies for travel diaries, let us pretend that she cannot purchase a pre-certified scale. How would she evaluate the available scales before starting her experiment?

The analog to ad hoc evaluation procedure would involve recruiting several of her friends and family to weigh themselves on the scales and compare the reported weight with their true weight. This analogy clearly reveals some limitations of the ad hoc procedure: (i) How does she trust that the self-reported weights are "true"? (ii) If all her friends are adults weighing 55 kg – 75 kg, how does she know how the scales perform outside that range? She can overcome the range limitations by recruiting a broader set of testers, e.g., through an intercept survey. However, that modification makes the ground truth limitation worse, since it is less likely that strangers will reveal their true weight. A further modification might pay contributors to improve the self-reported accuracy, but at this point, she is essentially running the experiment.

A more robust evaluation procedure would involve choosing known weights across a broad range (e.g., 0 kg to 300 kg in 10 kg increments) and comparing them to the reported weights. Since no instrument is perfect, there is likely to be some variation in the values reported. She would likely repeat the experiments multiple times in order to establish error bounds.

HMS evaluation procedures need to be more sophisticated than simple physical measurements since: (i) their operation is based on prior behavior (e.g., HMS duty cycling, android doze mode) and the potential for feedback loops makes it important to control the *sequence* of evaluation operations, (ii) unlike a physical scale, which has a fixed one-time cost, they have an ongoing, variable cost in terms of battery drain, so the evaluation must assess the power/accuracy trade-off, and (iii) unlike scalar weight data, HMSes generate strongly correlated timeseries data, which is extremely hard to anonymize.

Therefore, the main contributions in this chapter are:

1. We propose an **evaluation procedure** for HMSes based on predefined, ground truthed, artificial trips and outline how it addresses the above challenges,

2. We describe the design of a cross-platform **evaluation system** that can be used to perform such evaluations reproducibly and publish the results.

3. We highlight some lessons learned during this process that future groups might want to take into account while designing their experiments.

6.2 Requirements for evaluating Human Mobility Systems (HMSes)

Human Mobility Systems (HMSes) need a methodology for rigorous evaluation that allows users of the data to understand their limitations and their accuracy in various settings. Before establishing such a method, we need to understand the evaluation requirements, and the challenges associated with meeting each requirement. Establishing a clear set of requirements also allows us to understand the limitations of the proposed method and provides a roadmap for future improvements.

6.2.1 Holistic evaluation: power vs. overall accuracy

Using smartphones for collecting background sensed location data leads to higher battery drain. Travelers are very sensitive to battery life [FDK11, RQZ07], and there is a clear power/accuracy trade-off for smartphone sensing (Chapter 4). Naïve high accuracy sensing drains the battery for almost all users (Section 4.4), but techniques to lower battery drain also lower the accuracy (Section 4.2) So it is critical that the evaluation consider both power and accuracy together. For example, if an HMS can get 95% accuracy, but runs out of battery in 2 hours, the deployer needs to adjust the incentives offered (e.g., money, utility, ...) accordingly.

6.2.2 Privacy preserving

The data collected by HMSes includes location traces, which are inherently privacy sensitive. While a common privacy technique is to *de-link* datasets by replacing Personally Identifiable Information (PII) with a code [MGS10], location traces allow re-identification from the raw data alone [ZB, dHVB13]. Intuitively, from the location traces, we can find the *places* where people spend most of their time, which allows us to discover their home and work locations and uniquely identify them. This implies that the evaluation methodology must address privacy concerns.

6.2.3 Ground truthed

In order to fully evaluate the data collected, we need ground truth for not just the mode, but also the trip start and end times, section start and end times and the travel trajectory. Labeling trips through prompted recall is a low effort technique to collect mode ground truth, but it depends on accurate trip and section segmentation. Segmentation ground truth requires recalling the start and end *times* at the end of the day, which is likely to be unreliable [SSLA15, p. 206-207]. Similarly, for evaluating trajectories, travelers can potentially draw out spatial ground truth but spatiotemporal ground truth is almost impossible to obtain after the fact.

6.3 Controlled Evaluation of context-sensitivity

Instruments are typically evaluated by repeatedly exposing them to controlled inputs to determine their error characteristics. In the case of complex systems such as HMSes, the evaluation needs additional controls for feedback loops, cost/accuracy trade-offs and privacy considerations. In this section, we outline EM-EVAL — a procedure for HMS evaluation that addresses these concerns with two techniques that are novel in this domain:

(i) predefined, **artificial trips** that support spatial ground truth, preserve privacy, increase the breadth of *trip types* and support repetitions for establishing error bounds, and

(ii) power and accuracy **control phones** carried at the same time as the experimental phones, that can cancel out context-sensitive variations in power and accuracy.

6.3.1 Artificial timeline

The core of the experimental procedure is the predefined specification of a sequence of artificial *trips*, potentially with multiple *legs* or *sections* per trip. The trajectory and mode of travel is also predefined. The *data collector* completes the timeline trips by strictly following the specified trajectory and mode while carrying multiple phones that collect data simultaneously using different configurations.

The specified predefined trajectories provide spatial ground truth. We do not predefine temporal ground truth since it is extremely hard to control for differences in walking speeds, delays due to traffic conditions, etc. We use manual input from the data collector to collect *coarse* temporal "ground truth" of the transitions along the timeline. We do not use manual input for *fine-grained* temporal ground truth along the trajectory because: (i) human response times are too slow for fine-grained temporal ground truth during motorized transportation, and (ii) distracting the data collector during active transportation can be risky.

Using an artificial timeline addresses several of the unique challenges associated with HMS evaluation.

Privacy Since the trips are artificial, they preserve the data collector's privacy. Even if his adversaries would download the trips, they would not be able to learn anything about his normal travel patterns.

Spatial ground truth Since even high accuracy (GPS-based) data collection has errors, predefining spatial ground truth allows us to resolve discrepancies (Section 6.4) and compute the true accuracy.

Breadth and variety of trips Artificial trips allow efficient exploration of the breadth of the trip space. For example, the trips could include novel modes such as e-scooters and e-bikes, or specify different contexts for conventional modes, such as express bus versus city bus.

Repetitions Since the trips are predefined, they can be repeated exactly. This allows us to use standard variance and outlier detection to estimate error bounds on the measured values.

6.3.2 Control phones

The artificial trips give us spatial ground truth, but they do not give us cost (power consumption) or temporal ground truth.

We control for the cost through the use of the use of multiple phones, carried at the same time by the data collector. The phones carried by the data collector are divided into control phones and experiment phones. The control phones represent the baseline along each of the axes in our trade-off and the experiment phones implement a custom sensing regime that is at some intermediate point. The evaluation procedure allows us to determine those points.

Power The power control phone captures the baseline power consumption of a phone that is not being used for tracking by a HMS. This does not mean that the phone is idle — phone OSes (e.g., iOS or android) are complex, context-sensitive systems that perform their own location tracking (e.g., "Find my iPhone") and their own duty cycling (Figure 6.1). Using a power control allows us to identify the **additional power** consumed by the HMS, even if it is context sensitive.

Accuracy The accuracy control captures the upper bound on the accuracy of a particular class of smartphones given sensor and OS limitations. While we would like to compare the experimental accuracy to ground truth, (i) all sensors have errors, so ground truth is not achievable in practice, (ii) artificial trips give us spatial but not temporal ground truth, and (iii) GIS-based trajectory specifications do not have an associated power trade-off. Using an accuracy control allows us to compare the experimental data collection against the best achievable data collection, in addition to the ground truth.

6.4 Discussion of alternative procedures

While EM-EVAL (Section 6.3) addresses the complexities of HMS evaluation, it also imposes a much higher researcher burden than the ad hoc method. This raises the question of whether all these controls are necessary or merely sufficient. In this section, we discuss some alternative approaches and highlight the unexpected behavior that they would miss. This list is not comprehensive but provides a flavor of the arguments without tedious repetition.

6.4.1 No artificial trips

Creating predetermined trips requires an upfront investment in effort, and requires the data collector to take trips just for data collection. An alternative would use multiple phones, but allow data collectors to go about their regular routines and tag the modes only. We could use the accuracy control phones to determine the ground truth trajectory.

No privacy Capturing the data collector's regular routines compromises their privacy. Even if the data does not include their name or phone number, a list of their commonly visited places and trips can form a unique fingerprint that can uniquely identify them [dHVB13, ZB]. This sensitivity precludes evaluation data from being published and used for reproducible research.

No repetition The behavior of the same phone with the same configuration can vary over time, both for power and for accuracy (Figure 6.1). Repeating the same trip multiple times allows us to detect and remove outliers. With ad hoc trips, it is unclear whether any difference in behavior is real or caused by context-sensitive variation. And without predetermined trips, it is challenging to repeat the same trips and trajectories over time.

No spatial ground truth No sensor is perfect and even the accuracy control phones can have sensing errors. If we see a divergence between an experiment phone and the accuracy control, it is unclear which one has the error (Figure 6.1).

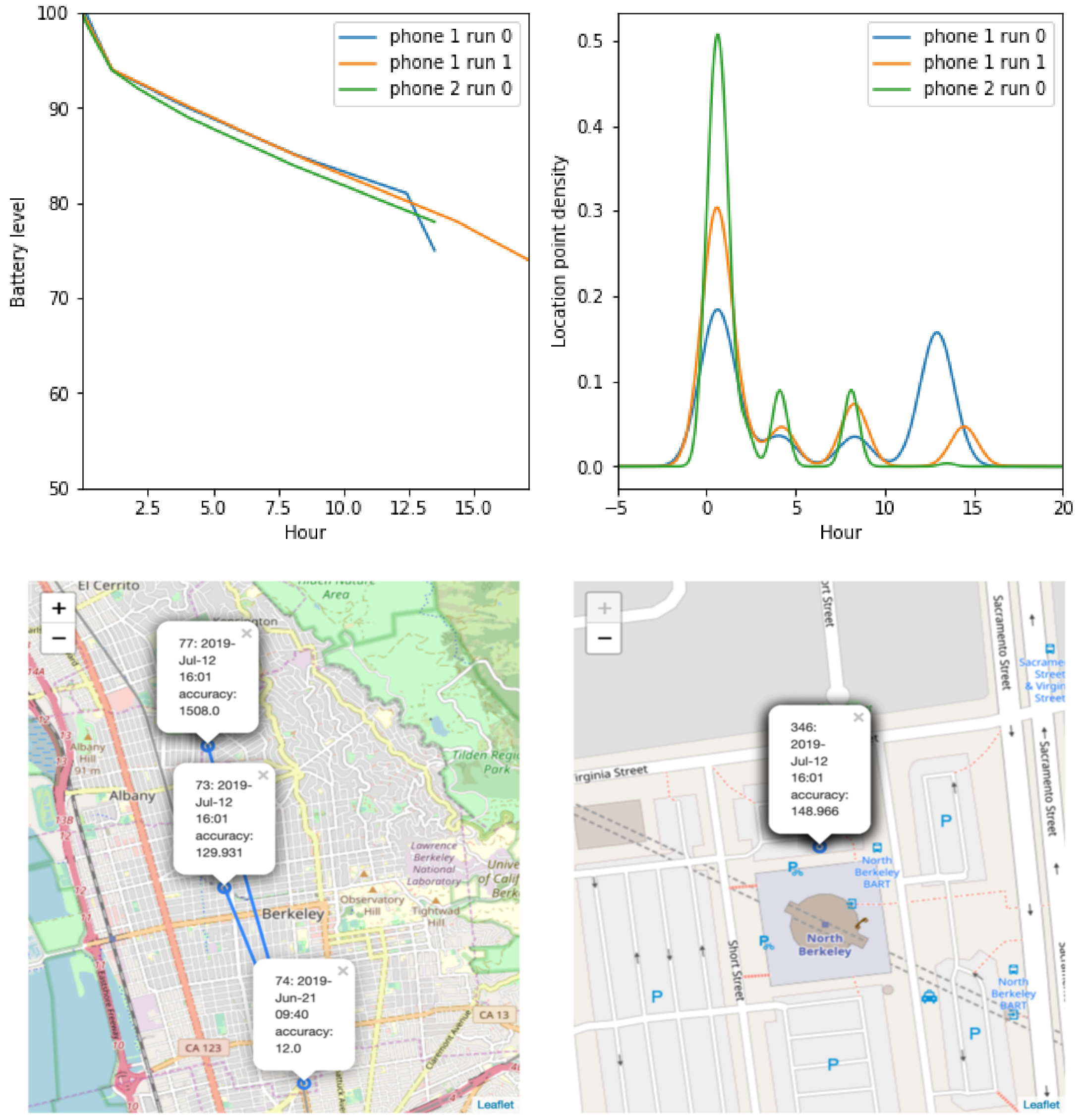
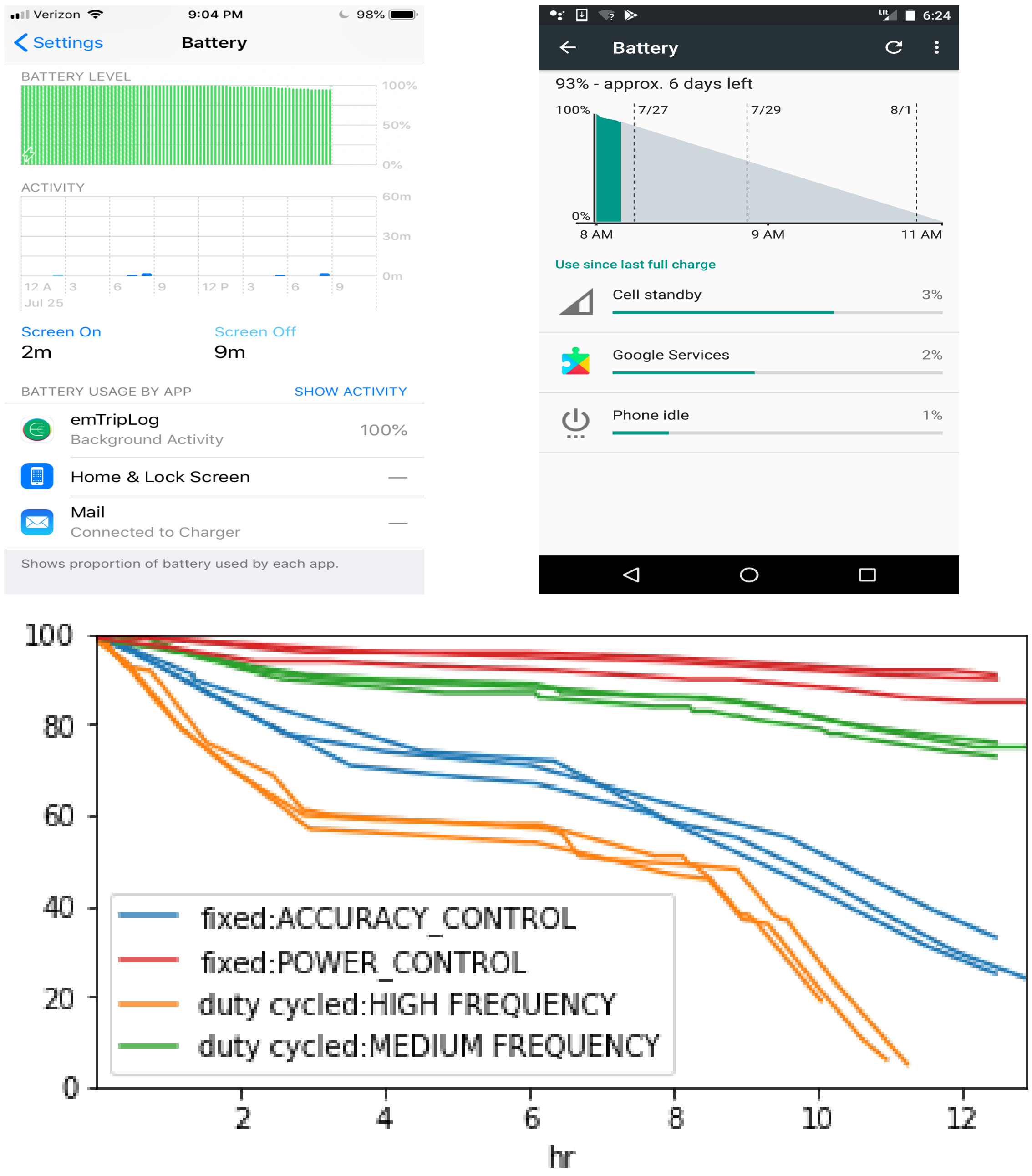


Figure 6.1. **Top:** Power variation illustrated by duty cycling on android. All the phones were configured identically, and placed in the same environment. The built-in duty cycling on android switches all phones to low power mode at around 1 hr. However, phone 1, on run 1 alone, switches back to high power mode at around 12.5 hours. Repeating experiments allows us to distinguish the first consistent duty cycle and the second outlier. **Bottom:** Accuracy variation illustrated by mismatched timestamps during trajectory data collection. Both trajectories are collected from identical phones during a subway trip. Point 74 has an accuracy radius of only 12, but its timestamp is in June instead of July! Spatial ground truth allows us to sort out the varying accuracies here.



Verizon 9:04 PM 98%
Settings Battery
BATTERY LEVEL
ACTIVITY
Screen On 2m
Screen Off 9m
BATTERY USAGE BY APP SHOW ACTIVITY

emTripLog Background Activity	100%
Home & Lock Screen	—
Mail Connected to Charger	—

Shows proportion of battery used by each app.

6:24
Battery
93% - approx. 6 days left
Use since last full charge

Cell standby	3%
Google Services	2%
Phone idle	1%

Figure 6.2. **Top left**: iOS power control phone with the sensing app consuming 100% but of a power drain of only 2% over the entire day. **Top right**: android phone showing Google Play services as a separate power consumer. **Bottom**: Explicit duty cycling causes increased power drain at high frequencies, possibly due to greater CPU power consumption. Note that the battery drain flattens out on all curves during the middle, stationary part. The main difference is in the rate of power drain while moving — the checks for android's built-in duty cycling appear to be optimized to be more efficient than our simple implementation.

6.4.2 No control

Using control phones requires the researcher to purchase multiple phones of the same make, model and approximate age. While used smartphones are relatively cheap (USD 50 – USD 100), 4 android phones and 4 iPhones combined will still cost USD 400 – USD 800. An alternative would be to use one phone each for each OS, perform the timeline trips, and look at the app-specific power consumption reported by the phone OS.

Sensor access attribution and the meaning of the % Sensor access in modern phone OSes (android and iOS) is also context-sensitive, making it unclear how it is counted for per-app consumption. For example, if multiple apps request a sensor reading, the OS delays returning a result until it can batch related requests and serve all of them with a single sensor access (e.g., Figure 5.2). This is why the OSes treat the sensing frequency as a hint instead of a guarantee. Second, if sensor access is mediated by a service (e.g., fused location in Google Play Services), it is unclear whether the sensor access is counted for the service or the app (Figure 6.2). And finally, although android reports per app consumption as a % of the battery *capacity*, iOS does so as a % of the battery *consumption*. This indicates that on dedicated phones, the HMS under test will always show close to 100%, whether it is the power control or the accuracy control (Figure 6.2). Using a control phone for the power will cancel out these context-sensitive effects and estimate the difference in power drain with and without the HMS app component installed.

Custom duty cycling increases power drain Sensing is not the only source of power consumption — CPU usage can also have a significant impact on power usage. HMSes can use smart local processing to reduce local sensing, but the increased power consumption from the CPU can cancel out the savings from the sensing. Including an accuracy control showed that, unlike in 2015 (Section 4.2), the basic duty cycling algorithm in our experiment paid for itself in low frequency sensing but actually increased power usage for high frequency sensing (Figure 6.2).

6.5 Evaluation system design

The EM-EVAL procedure (Section 6.3) allows us to estimate the power/accuracy trade-off of various sensing configurations used in Human Mobility Systems (HMSes). One of the novel components of the procedure involves the specification of pre-defined, artificial trips with ground-truthed trajectories and modes.

This section explores the nuances of implementing such a procedure. We first describe a publicly available reference implementation of a system – EM-EVAL-ZEPHYR

– that can be used perform this procedure. We then discuss challenges encountered while using the system to perform an experiment in the San Francisco Bay Area (Section 7.2). Some of these challenges were addressed by system improvements, while others are documented as best practices for future data collectors.

6.5.1 System overview

EM-EVAL is a generic *procedure* for HMS evaluation – it does not actually collect any data. To use it, we need a concrete system that configures data collection based on the spec configurations, collects coarse temporal ground truth, periodically reads battery levels and stores data for future analysis.

We built a system – EM-EVAL-ZEPHYR – that combines our prior work on power evaluations [SFC+] with our existing HMS platform [SBM+18] and supports performing the EM-EVAL procedure. The system consists of three main parts:

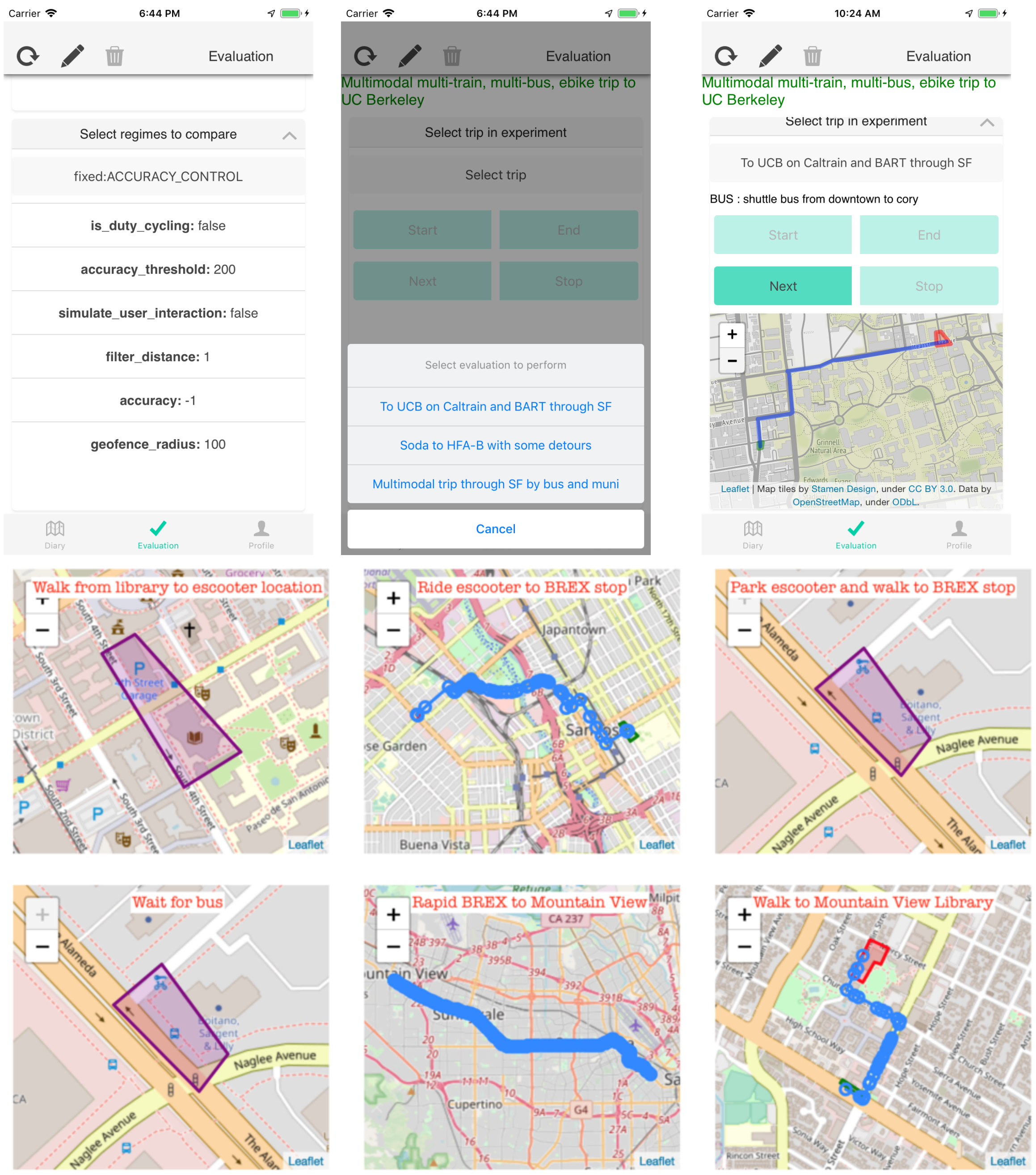
Evaluation Specification The *spec* describes an evaluation that has been performed or will be performed in the future. In addition to mode and trajectory ground truth, it includes the app configurations to be compared and the mapping from phones to evaluation *roles*. The spec automatically configures both the data collection app and the standard analysis modules.

To reduce evaluator burden, we provide preprocessing functions to fill in trajectory information based on route waypoints for road trips and OSM relations for public transit. We also provide sample notebooks to verify timelines and their components before finalizing and uploading the evaluation spec.

Auto-configured Smartphone App We have generated a custom UI skin for our E-MISSION platform [SBM+18] that is focused on evaluation. It allows evaluators to select the current spec from the public datastore, and automatically downloads the potential comparisons to be evaluated, the role mappings and the timeline.

Since the e-mission platform data collection settings are configurable through the UI, the sensing configurations defined in the spec are automatically applied based on the phone role when the data collector starts an experiment. For example, when starting an experiment to compare high accuracy (HAHFDC) versus medium accuracy (MAHFDC) data collection, the second experiment phone will automatically be set to MAHFDC settings. Finally, when the data collector performs the trips, he marks the transition ground truth in the UI, and the app automatically displays the next step in the timeline (Figure 6.3).

Public Data + Sample Access Modules Since there are no privacy constraints, EM-EVAL-ZEPHYR uploads all collected data to a public instance of the E-MISSION server. The associated repository contains sample notebooks that can



Carrier 6:44 PM

Evaluation

Select regimes to compare

fixed:ACCURACY_CONTROL

is_duty_cycling: false

accuracy_threshold: 200

simulate_user_interaction: false

filter_distance: 1

accuracy: -1

geofence_radius: 100

Diary Evaluation Profile

Carrier 6:44 PM

Evaluation

Multimodal multi-train, multi-bus, ebike trip to UC Berkeley

Select trip in experiment

Select trip

Start End

Next Stop

Select evaluation to perform

To UCB on Caltrain and BART through SF

Soda to HFA-B with some detours

Multimodal trip through SF by bus and muni

Cancel

Carrier 10:24 AM

Evaluation

Multimodal multi-train, multi-bus, ebike trip to UC Berkeley

Select trip in experiment

To UCB on Caltrain and BART through SF

BUS : shuttle bus from downtown to cory

Start End

Next Stop

Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

Diary Evaluation Profile

Walk from library to escooter location
Ride escooter to BREX stop
Park escooter and walk to BREX stop
Wait for bus
Rapid BREX to Mountain View
Walk to Mountain View Library
Japantown
Buena Vista
Naglee Avenue
Alameda
Cupertino
Leaflet

Figure 6.3. **Top**: Spec components in EM-EVAL-ZEPHYR include configuration, timeline and trip details. **Bottom**: Sample spec for a multi-modal trip, including transfers and waits for public transit.

download, visualize and evaluate the data associated with a particular spec. All the data used in this paper is publicly available, and the notebooks can be manipulated interactively using binder[1].

Note that although the EM-EVAL **procedure** is general, the current implementation of the EM-EVAL-ZEPHYR **system** is integrated only with the E-MISSION platform. Using the procedure with other HMSes will require re-implementing the EM-EVAL procedure with the other HMS, or using a combination of systems for the evaluation. For example, EM-EVAL-ZEPHYR can still read the battery level periodically, display the trip sequence to the data collector, and be used to mark the transition ground truths. However, the evaluator needs to configure the settings for the app being tested manually, and to download, clean and analyze the resulting data.

6.5.2 System iterations and lessons learned

As we started collecting data, we had to resolve some ambiguities around exactly when the transition ground truth should be collected. We also discovered best practices that increased the likelihood of successful data collection. This section outlines these lessons learned.

System change: capture transition complexity One of the big promises of using HMSes for instrumenting human travel is that we don't have to focus only on the primary mode. Instead, with fine-grained data collection, we can understand the full complexity of end to end travel.

In fact, the only true unimodal trips are walking trips. Everything else is multimodal. Thus, a significant change to the system was to restore the hidden complexity that is elided from user descriptions of travel diaries. For example, consider the trip description "Drive from Mountain View Library to Los Altos Library". Although that appears to be a unimodal trip, it is actually a multimodal trip which involves implicit walk access sections to and from the car at the source and destination respectively.

It is not possible to predetermine the ground truth for these walk access sections since we cannot control which parking spaces are available when we perform the trip. We address such issues by adding *shim sections*, and expanding the start and end from points to ≈ 100 m polygons. We can then relax the constraints around ground truth within the polygon by only using the reference dataset, but still check the accuracy of the mode inference (Figure 6.3).

Best practice: Pilots are critical In spite of reviewing the predetermined trajectories ahead of time as part of the validation process, and also having them

[1] `https://mybinder.org/`

displayed on the EM-EVAL-ZEPHYR UI, we found that we frequently made small mistakes, during the first round of data collection for a new timeline. Sometimes, we found that the predefined routes, potentially suggested by Open Source Routing Machine (OSRM), felt unsafe to bicycle on. We had to tweak the specification to pick safer routes. The second repetition generally resolved these issues. In order to avoid a stressful data collection experience, we suggest running through a new timeline with a trial run before starting full-featured data collection.

Best practice: Mindfulness Remembering to mark the transition ground truths was one of the hardest parts of the ongoing data collection and really highlights the challenges of ground truth collection. In spite of the fact that she was performing artifical trips to collect data for her own project, one of the authors forgot to mark wait $\rightarrow$ move transitions during the pilot for the long multi-modal timeline because she had started checking her email while waiting. It is important to be present in the moment and pay attention to the context while collecting data.

6.6 Conclusion

Human Mobility Systems (HMSes) are complex software systems that run on equally complex smartphone operating systems (OSes). This complexity implies that there is rarely a simple linear relationship between their inputs and outputs, which complicates their evaluation.

We outline a procedure, based on repeated travel over predefined artificial *timelines* carrying experiment and control phones, to control this complexity. We show that it can control for outliers, and also reveal meaningful signals about the behavior of smartphone virtual sensors that are relevant to instrumenting human travel data.

The procedure is privacy-preserving, so it does not need human subjects approval. It focuses on *trip* diversity, not *demographic* diversity, so it can be undertaken by a small research group, or even a single researcher, as a pre-pilot before recruiting study participants. It uses predetermined trips and modes, so it can efficiently explore complex or newly emerging travel patterns and modes, such as e-scooters. The procedure, and the associated reference implementation can simplify the testing required before a study is launched.

The next stage is evaluation, which adopts this procedure to evaluate the sensing accuracy for various settings. In addition to assessing the architecture usage, we also gauge the ability of the analysis algorithms (Chapter 5) to boost the sensed accuracy.

Chapter 7

Performance Evaluation

In prior chapters, we have outlined the architecture of a platform for instrumenting human travel data, described its functionality, and indicated how it can be extended to meet various use cases. We have also developed an evaluation procedure for analyzing the related power/accuracy trade-offs, and collected ground truth data using the procedure. In this chapter, we perform a quantitative evaluation of the platform in both these areas.

For the architecture and extensibility, we examine the usage of the architecture modules in the context of three use cases from different domains — (i) a classic travel diary, (ii) a crowdsourcing initiative for accessibility metrics, and (iii) a behavioral study on incentivizing sustainable transportation. We show that our use cases used an average of 64% of the features of the platform, with $\approx$ 3-4 months of part-time CS undergraduate time for each new case. Every use case contributed at least one extension, primarily client-related, back to the platform. This serves as an existence proof that it is possible to build a nascent community around open source HMSes.

Since the data collection settings of the platform are configurable, instead of evaluating a single implementation, we examine the power/accuracy/analysis trade-off for a variety of configurations. These results can assist deployers in choosing the settings suitable for their application. We identify three timelines with varying travel times and evaluate them under high/medium accuracy and frequency settings. We show that the power consumption on android is primarily affected by frequency and on iOS, by accuracy. The sensing configurations primarily affected the trajectories, but not the segmentation or classification, which use other virtual sensors (Section 5.2). The post-processing analysis improved trajectory and section segmentation significantly, but did not affect the other metrics.

7.1 Evaluation of architecture and modularity

In this section, we evaluate our architecture under the context of bridging the builder–deployer gap (Section 3.1). Our evaluation is based on its use in three separate use cases (or "apps") from deployer projects. We show that although the use cases initially appear different, they re-use several common modules without modification, and are able to extend other modules to meet their needs. We also show that the development time for the projects is much shorter than building one-off apps from scratch. Finally, these projects show the ability to overcome the social challenges associated with interdisciplinary platform building. However, in the absence of a rigorous user study, we have no knowledge of negative cases (e.g., deployers who are unconvinced by platforms). Further, although the platform enhancements have reduced as the platform matures, requests for documentation, particularly from non-developer deployers, are increasing with adoption. Therefore, the long-term viability of such platforms is still an open question.

7.1.1 Metrics

When presenting the idea of a platform to deployers, there was skepticism about the benefits of a platform. Some questions that have been raised, and the metrics that we used to answer them, are:

Q: Is there enough common functionality that it can be abstracted out?

extensibility: We examine the platform components identified by the architecture, and see how they are used by the systems instantiated from it. For example, does the architecture ensure that that common functionality is reused and all customization is restricted to customizable modules?

Q: What is the difference between an app and a platform? What is wrong with a one-off project? How much time will using a platform actually save?

utility: We compare the time required to create a one-off app from scratch with the time required to customize a platform.

Q: Will non-developer communities embrace open source platforms? Why not continue to use consultants instead?

adoption: We measure external contributions to both core modules and customizations, especially if the customizations were re-used by other projects. Consultants can use open source platforms too.

Q: What about application-specific metrics such as survey responses or app launches?

application specific metrics: We do not include these because the platform does not control application-specific settings, and the metrics suitable for one application may not be suitable for others.

7.1.2 Use cases

Abstracting a set of specific systems into a platform involves striking a delicate balance between breadth and compactness. The platform should be *broadly applicable* to a wide variety of applications, or otherwise deployers will continue to build one-off systems. However, if the platform is too broad, it loses the clear structure that makes it useful. Striking that balance is more an art than a science, and the balance can shift over time. Most platform builders use a small ($n \approx 3$) set of canonical use cases to define the platform. The E-MISSION platform uses three such canonical use cases — a classic travel diary, an infrastructure crowdsourcing project, and a behavior change study (Figure 7.1).

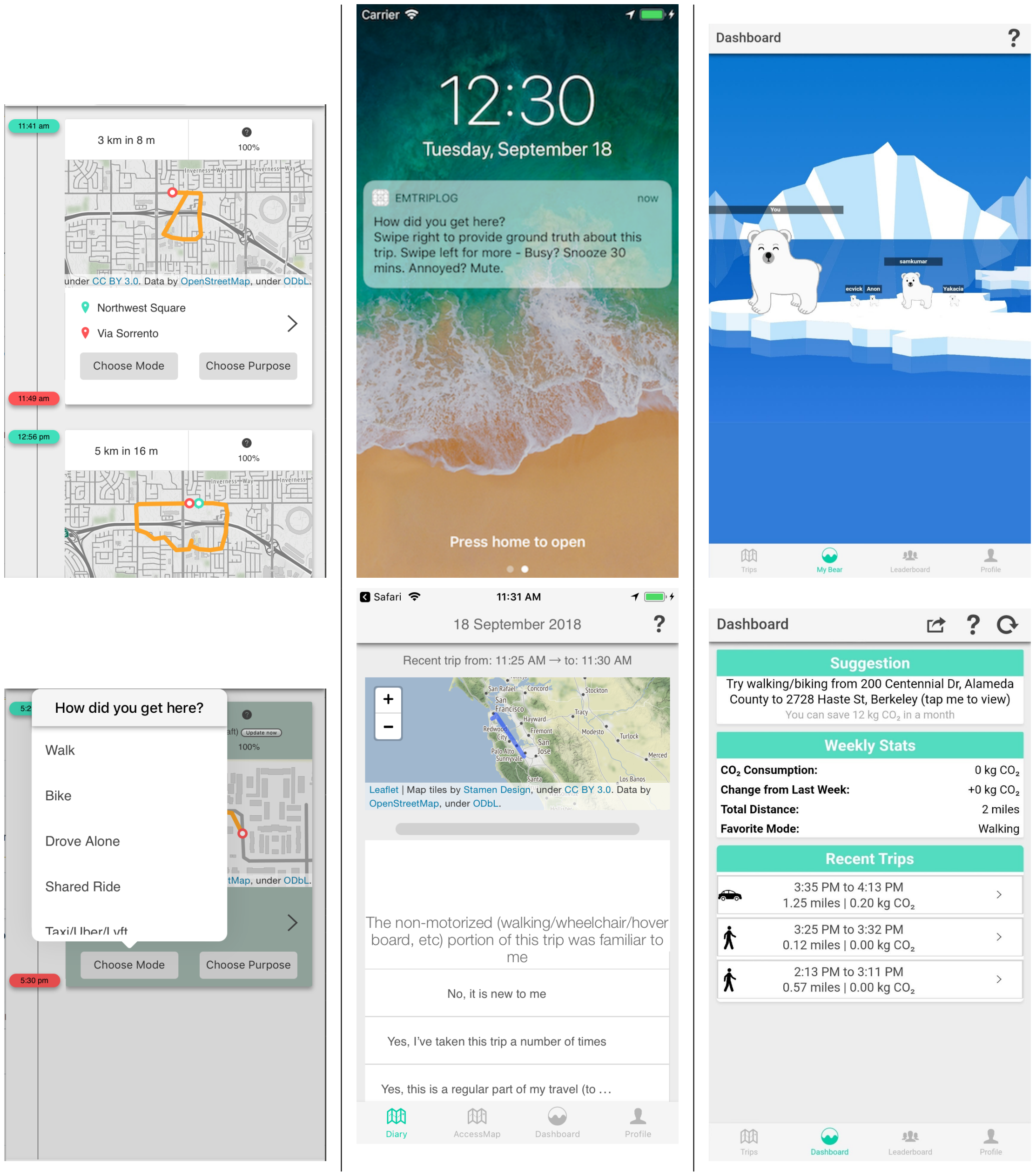
All the projects provisioned their own server and collected their own data. All of them used a UI channel on top of the E-MISSION base app. The use cases typically used 16 out of 25 features (64%), although the exact set varied according to the use case. In all three cases, the actual customization was done by undergraduates with computer science (CS) backgrounds; the undergraduates were from three different universities. The undergraduate who worked on the `cci-berkeley` project had worked with E-MISSION the prior summer; the others had no prior direct experience.

classic travel survey, `cci-berkeley` Classic travel surveys are used for instrumenting mobility patterns in a population. They are the most recent evolution of the traditional Household Travel Survey (HTS) which have shifted from paper logs to telephone surveys to GPS devices and finally, to smartphones. They are primarily focused on unobtrusive data collection and currently provide monetary incentives for participation. They have to be careful about providing services to travelers in order to avoid the Hawthorne effect.

The Center for Community Innovation (CCI) is instrumenting mobility patterns of low-income households in order to study the effects of gentrification on overall Vehicle Miles Traveled (VMT).

They use a classic travel survey with a stripped down UI that only includes the travel diary. They also removed several of the controls from the profile, notably, the option for "Medium accuracy", and the entire Developer Zone. Since they had graduate students recruit participants in person, they chose to hand out a unique, randomly generated token for authentication instead of having users sign in with an email ID.

They added the ability for users to specify mode and purpose ground truth from the diary screen, including a rich set of modes such as carpool, shared ride, etc.



under CC BY 3.0. Data by OpenStreetMap, under ODbL.

How did you get here?
Swipe right to provide ground truth about this trip. Swipe left for more - Busy? Snooze 30 mins. Annoyed? Mute.

Recent trip from: 11:25 AM → to: 11:30 AM

Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

The non-motorized (walking/wheelchair/hover board, etc) portion of this trip was familiar to me

No, it is new to me

Yes, I've taken this trip a number of times

Yes, this is a regular part of my travel (to …

Try walking/biking from 200 Centennial Dr, Alameda County to 2728 Haste St, Berkeley (tap me to view)
You can save 12 kg CO_2 in a month

CO_2 Consumption:	0 kg CO_2
Change from Last Week:	+0 kg CO_2
Total Distance:	2 miles
Favorite Mode:	Walking

3:35 PM to 4:13 PM
1.25 miles | 0.20 kg CO_2

3:25 PM to 3:32 PM
0.12 miles | 0.00 kg CO_2

2:13 PM to 3:11 PM
0.57 miles | 0.00 kg CO_2

Figure 7.1. Screenshots of the three different use cases (L-R: `cci-berkeley`, `opentoall`, `tripaware`)

They initially used the event notifier to pop-up a survey at the end of every trip, but turned it off after negative feedback during the pilot. They also added a survey that would link the user token to the user UUID, but ended up not using it when they switched to token-based authentication.

They did not run the analysis pipeline on the data collection server. Instead, the data analysts pull subsections of the data onto their own laptops and run the analysis on an ad hoc basis.

crowdsourced infrastructure, `opentoall` Infrastructure shortcomings such as missing bicycle lanes, bumpy sidewalks and confusing intersections can be crowdsourced from concerned travelers. This data could either be continuously annotated (e.g., label the quality of each segment along your route) or intermittently triggered when the user actually encounters a barrier. Linking the collected data to robust mechanisms for (i) notifying group members about newly reported issues, and (ii) ensuring that responsible institutions address hotspots can motivate participation by users.

The Taskar Center for Accessible Technology (TCAT) is documenting barriers to accessibility — bumpy or non-existent sidewalks, blocked routes, etc.

While they include the classic trip diary, they prompt the user at the end of every trip for their experience of the trip, including any barriers that they encountered that are not already in the `opentoall` dataset. They use OpenID connect, linked to their own keystone server for authentication. This allows them to associate trips taken by any user with trips recommended by the `opentoall` trip planner.

They are interested in gamification to prompt crowdsourcing of barriers, as well as adding local processing for bumpy sidewalk detection using the accelerometer.

behavior change, `tripaware` While the classic travel survey is sufficient to *detect* changes in travel behavior, mobility data can also be used to *suggest* behavior changes. Apps could provide services, propose more sustainable alternatives, or influence user emotions to nudge them towards more sustainable behavior. There are many potential design choices for modifying behavior change. We expect the most effective designs for a particular area to be heavily influenced by local cultural preferences.

A group of undergraduates participating in a research apprentice program studied the difference between emotion (moody polar bear) and information (suggestions for alternate modes) in motivating sustainable behavior.

They conducted a Randomized Controlled Trial (RCT); participants were randomly assigned to the emotion, information or control channels, and automatically downloaded the appropriate UI for their group.

They retained the classic trip diary for the control group. For other groups, they added a leaderboard, and modified the summary dashboard based on intervention. For the information group, they provided summary statistics and a set of suggestions for alternatives. For the emotion group, they showed a polar bear that grew or shrank, and was compared to the others in the same leaderboard tier.

7.1.3 Extensibility + adoption

The community involvement metrics (Figure 7.2) indicate strong interest and a significant contributor base. Digging deeper, the usage matrix (Table 7.1) indicates that most of the components were used without modification in a majority of the projects. Most changes were to customizable modules. The only external enhancement to the core modules was the addition of a new auth by the `opentoall` project. Further, many of the contributions to customizable modules can re-used by other projects. For example, the enhancement that allowed users to specify mode and purpose, introduced as part of the `cci-berkeley` project was adapted for use in the `opentoall` project.

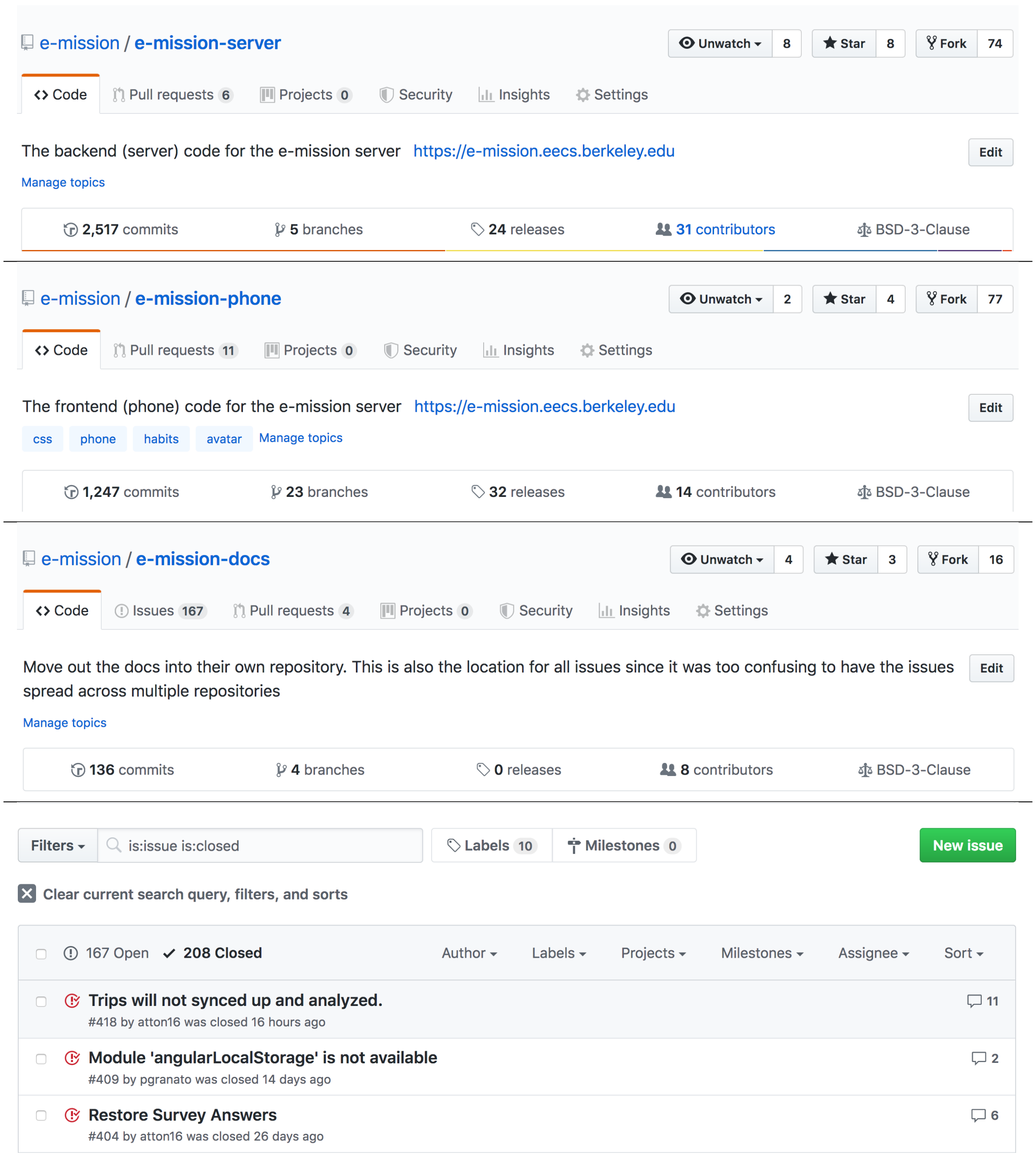
Notable exceptions to these general results include:

Auth Every project used a different authentication mechanism. Having a configurable authentication mechanism allows deployers to easily switch between mechanisms, as well as allowing projects to contribute auth plugins that they needed for later re-use.

Coarse timer/Push notify 2 out of 3 projects did not turn on the silent push notification based coarse timer on iOS. Since the data can also be uploaded at trip end, the data collection still worked since both projects were based in the United States, which has reasonable connectivity. They also did not use targeted push notifications.

Algorithm extensions No group has yet contributed algorithm extensions. The CCI group is actively analyzing their collected data and might contribute improvements if they develop any. Since the architecture and data model are now clearly documented, we hope that researchers who work on inference algorithms in the future will contribute them to the platform.

Aggregate metrics Since all the projects so far have been focused on small-scale data collection, they have not explored the aggregate analyses possible. The `opentoall` crowdsourced dataset could be an instance of such analysis once the study is complete.



e-mission / e-mission-server

Unwatch 8 | Star 8 | Fork 74

Code | Pull requests 6 | Projects 0 | Security | Insights | Settings

The backend (server) code for the e-mission server https://e-mission.eecs.berkeley.edu

Edit

Manage topics

2,517 commits | 5 branches | 24 releases | 31 contributors | BSD-3-Clause

e-mission / e-mission-phone

Unwatch 2 | Star 4 | Fork 77

Code | Pull requests 11 | Projects 0 | Security | Insights | Settings

The frontend (phone) code for the e-mission server https://e-mission.eecs.berkeley.edu

Edit

css | phone | habits | avatar | Manage topics

1,247 commits | 23 branches | 32 releases | 14 contributors | BSD-3-Clause

e-mission / e-mission-docs

Unwatch 4 | Star 3 | Fork 16

Code | Issues 167 | Pull requests 4 | Projects 0 | Security | Insights | Settings

Move out the docs into their own repository. This is also the location for all issues since it was too confusing to have the issues spread across multiple repositories

Edit

Manage topics

136 commits | 4 branches | 0 releases | 8 contributors | BSD-3-Clause

Filters | is:issue is:closed | Labels 10 | Milestones 0 | New issue

Clear current search query, filters, and sorts

167 Open ✓ 208 Closed | Author | Labels | Projects | Milestones | Assignee | Sort

Trips will not synced up and analyzed. — 11
#418 by atton16 was closed 16 hours ago

Module 'angularLocalStorage' is not available — 2
#409 by pgranato was closed 14 days ago

Restore Survey Answers — 6
#404 by atton16 was closed 26 days ago

Figure 7.2. Basic community involvement statistics from github for the project. Top to bottom: server repo, phone repo, docs repo, and closed issues

	Feature	`cci-berkeley`	`opentoall`	`tripaware`
Client	Local buffering	✓	✓	✓
	Local processing	✓	✓	✓
	Location state machine	✓	✓	✓
	Consent	✓	✓	✓
	Auth	Pre-created token ↑	OpenID connect ↑	Google auth
	bi-directional sync	✓	✓	✓
	protocol client	✓	✓	✓
	Coarse timer	×	×	✓
	Event notifier	Removed after pilot	✓	×
	Setup	✓	✓	✓
	UI update	✓	??	✓
	Push notify	×	×	✓
	UI channel	✓	✓	✓
Server	Input timeseries	✓	✓	✓
	Analysis timeseries	Offline, on laptop	✓	✓
	K-V store	✓	✓	Added leaderboard tier position ↑
	Incoming buffer	✓	✓	✓
	Webapp	✓	✓	New API endpoint for suggestions ↑
	Push notify	×	×	✓
	Integrations	GIS for mode	GIS for mode, `opentoall` trip planner	GIS for mode
Analysis	Pipeline usage	Analyst runs offline ✓	✓	New stage for *tiers, happiness*
	Reproducibility	Multiple analysts work with subsets of data ✓	×	Investigate errors in mode inference ✓
	Algorithm Extensions	×	×	×
	New data model objects	`mode_confirm` ↑	survey result ↑	×
	Aggregate metrics	×	×	×

Table 7.1. Three projects and their usage of various components of the architecture. Usage key - ✓: no modification, × not used, ↑ enhancement contributed

7.1.4 Utility

The utility metric is difficult to assess because one-off deployer projects that did not publish source code do not publish their development time either. The commercial rMove app [GFHG16] took five months to develop, but the development team size is unknown. The one-off Quantified Traveler project [JAC$^+$15] involved a development team of five in addition to the authors, but the details of contribution and time taken are unclear. DataMobile [PF16] is open source, but it only recently (June 2018) created github repositories through code bulk upload, so we are unable to see the commit history.

As shown below, all the E-MISSION changes so far have taken < 3 months with CS undergraduates working part-time. Less ambitious changes are possible with one undergraduate, RCTs with multiple UIs need a larger team.

cci-berkeley $\approx$ 6 weeks of full-time work by one CS undergraduate with prior E-MISSION experience + $\approx$ 2 weeks of part-time work by another CS undergraduate to change text and colors.

opentoall $\approx$ 1 month of full-time work by one CS undergraduate for extending auth + $\approx$ 3 months of extremely part-time effort for UI changes + integration.

tripaware $\approx$ 3 months of 6-10 hrs/wk by 6 CS undergraduates to design 3 custom UIs for RCT + server changes for leaderboard and polar bear

An advanced UI is planned to be developed for Sydney area to be completed in one month time by a professional programmer and a research student, to collect the travel diary of Sydney residents for two weeks. Further, the platform is being used for collecting information about the route choice behavior of pedestrians in dense urban area of Sydney CBD.

Note that this estimate only accounts for deployer, not builder effort. While the pace of platform enhancements has slowed as it has matured, requests for clarification and documentation are increasing. These requests are particularly numerous when non-CS deployers are involved — for example, although `cci-berkeley` UI changes took only $\approx$ 2 months, it took another month and a half for the CCI group to perform routine non-platform-specific system administrative tasks.

Integrating with ongoing *software carpentry*[1] efforts may result in documentation that meets the needs of non-CS audiences without overwhelming platform builders, especially in resource-constrained research environments.

7.1.5 Application specific metrics

The evaluation does not include application specific metrics. This is mainly because (i) the platform is designed to be extensible, so it does not control the ap-

[1]`https://software-carpentry.org/`

plication configuration, and (ii) the metrics, such as the number of questions in the survey, may not be broadly applicable.

no control: The metrics are influenced by a mix of factors, few of which the platform controls. For example, app retention is likely to be influenced by monetary incentives, app functionality and power drain. The platform does not control either of the first two options, and the third is configurable by the app. So it is possible for applications with similar functionality, built on the same platform, to have drastically different retention rates. Thus, retention rate is not an appropriate metric for evaluating the platform.

application dependent concerns: Other metrics might be heavily application dependent. For example, metrics for travel surveys could include the number of questions for each trip and the response rate. These metrics are not meaningful for gamification, where users are typically not surveyed about trips, and might even be prevented from providing additional information in order to avoid cheating. Likely gamification metrics are number of app opens and the length of time on each screen.

7.2 Evaluation of data collection and analysis

Since e-mission is a platform, systems that use it can configure it to meet their needs. Therefore, this evaluation focuses on evaluating the performance at various settings and presenting the trade-offs for deployers to use. The key findings are:

power The power consumption on android is primarily affected by frequency and on iOS, by accuracy. At the lowest power levels tested, the power drain of sensing was only significant for daily travel time > 3 hours, and it was $\approx 15\%$ even for 6 hours of daily travel.

accuracy The sensing configurations only affected the trajectories and the end trip detection on android. The other metrics were largely consistent across configurations. This indicates that unless fine-grained trajectory information is necessary, the lower power levels are sufficient.

analysis The post-processing analysis improves all metrics, some more than others. The maximum spatiotemporal error dropped from 40 km to 15 km on android and from 20 km to 5 km on iOS. The maximum number of additional sections dropped from 10 to 4 on android and from 10 to 2 on iOS. The post-analysis median trip count difference was consistently non-zero only for the most complex timeline. However, the start/end trip and section times did not improve significantly.

7.2.1 Experiment design

EM-EVAL is a generic evaluation procedure and can be used with any kind of HMS. EM-EVAL-ZEPHYR is a reference implementation of EM-EVAL that used to evaluate many experimental settings relevant to HMSes over any set of trajectories. The evaluator can pick her settings based on her research goals (Chapter 6). In this section, we outline our goals for this evaluation, and use them to define three timelines that cover 15 separate modes, including recently popular modes such as e-scooter and e-bike.

Dwell time Instead of focusing only on trips, we wanted to evaluate a timeline that included significant dwell time. We could see from our calibration runs that android appears to have built-in duty cycling (Figure 6.2). Including significant dwell time would allow us to capture the impact of this context sensitive behavior. Therefore, we structured our timeline trips as round trips to libraries with an intermediate dwell time $\approx 3\times$ the mean travel time to the location.

Broad range of modes HMS evaluations should cover a broad spectrum of trip types. Since we are creating artificial trips, we can structure them to maximize mode variety. In order to efficiently cover this space, we tried to ensure that no mode was repeated. We only had to include commuter rail twice since there were few other transit options to reach the starting point chosen.

Multi-modal transfers Detecting multi-modal transfers in a HMS is tricky because there isn't a clear signal similar to a trip end. We ensure that there are many transition examples by emphasizing multi-modal transfers.

With those goals in mind, we decided on three artificial timelines of varying lengths that cover a total of 15 separate modes. We chose each timeline to be round trips to libraries to not include identifiable location data (e.g., home location) in our experiments. A description of each timeline with the associated modes and dwell times is given in Table 7.2.

7.2.2 Evaluation parameters

The experiment evaluates the power, accuracy and analysis trade-offs along 3 timelines of varying lengths. It first establishes the relationship between power and the *sensed accuracy* of the raw data received from the phone. The raw data consists of location, trip transitions and motion activity detection (Section 5.2). Android does not expose a trip end sensor, so the evaluation uses includes a custom dwell based implementation. Finally, it explores the improvements to overall accuracy by analysis algorithms.

id	Description	Outgoing trip modes	Incoming trip modes	Dwell time	Overall time
`unimodal trip car bike mtv la`	Suburban round trip	car (city streets)	bike	1 hrs	3 hrs
`car scooter brex san jose`	Downtown library	car (freeway)	escooter Bus Rapid Transit	3 hrs	5.5 hrs
`train bus ebike mtv ucb`	Multi-modal trip across the bay	commuter train subway city bus	ebike (shared) express bus downtown walk light rail commuter rail	6 hrs	12.5 hrs

Table 7.2. Brief description of timelines, covered modes, dwell times and overall times

7.2.2.1 Built-in, black-box sensing parameters

Modern smartphones include closed source APIs for

1. *fused location sensing*, which determines location with the specified accuracy based on a combination of GPS, Wi-Fi and other sensors,

2. *trip start/end detection*, which uses low power sensing to detect when a trip starts or ends, and

3. *motion activity detection*, which uses low-power sensors to determine whether the traveler is walking, bicycling or in a car.

Since the APIs are black-boxes to HMS builders, we evaluate the accuracy at various sensing settings. Configurations are a combination of these settings, so `HAMFDC` stands for High Accuracy, Medium Frequency, Duty Cycled collection.

High accuracy vs. Medium (HA vs. MA) High accuracy will tend to favor GPS and result in high power consumption.

High frequency vs. Medium (HF vs. MF) High frequency will sense and process more often so is likely to have higher spatiotemporal accuracy (e.g., will hug corners) but with higher power consumption.

Duty cycling vs. Always on (DC vs. AO) Duty cycling allows for high accuracy, high frequency sensing with low power drain, but with sensing gaps at trip start due to delay in detecting the trip start.

7.2.2.2 Trip end detection during sensing

Note that there is currently no built-in, app-invokable trip end detection API for android. We implement a naïve trip end detection algorithm to fill this gap for the sensing evaluation. For each sensed point, the algorithm reads the data from the last 5 minutes, computes the distances from the current point, and checks to see if the max distance is below the trip end detection threshold. This allows us to control for noise in the data and avoid spurious trip end detection. The computation cost for this algorithm depends on the density of the collected points since we run the algorithm more frequently and check more points on each run. More efficient algorithms that run less frequently or check fewer points will have lower computational needs but less sensitivity. Our results show that Google appears to have implemented a duty cycling algorithm for android that is more efficient than our naïve algorithm. This is consistent with our reasons for using virtual sensors where possible (Section 4.5). If this algorithm were exposed for third party apps to listen to, we could use virtual sensors for all our sensing needs.

7.2.2.3 Processing algorithms for improving accuracy

We can attempt to improve this *sensed* or *input* accuracy by running processing algorithms for smoothing, outlier detection and interpolation. We consider two possible analysis algorithms:

`master`: uses dwell time + visit detection for trip segmentation, unfiltered motion activity values for section segmentation, zigzag detection for trajectory smoothing and random forest for mode inference.

`gis-based`: Enhances segmentation to smooth out flip-flops in motion activity, GIS integration for enhanced mode inference.

7.2.3 Metrics

The post-processing steps can be classified into three broad themes, each of which can be evaluated using multiple metrics (Section 5.3).

Segmentation Splits a stream of sensed values into meaningful *segments* — e.g., *trips* and *sections*.

Trajectory tracking Detects outliers in spatiotemporal trajectories caused by erroneous sensing and removes them.

Classification Assigns labels to the segments. The most common classification task, and the only one we will evaluate here, is the determination of the travel mode for every section.

We now outline the common error conditions for each algorithm type, and define the metrics that can be used to characterize the error. Additional concrete examples of error characteristics can be found in the interactive notebooks of the evaluation repository[2]

7.2.3.1 Segmentation

The main error conditions for segmentation algorithms are:

1. the algorithm detects the correct number of segments, but the start and end transitions don't match the ground truth (Figure 7.3, top)
2. the algorithm detects more segments than the ground truth, flip-flopping during a single real segment (Figure 7.3, bottom)
3. the algorithm detects fewer segments than the ground truth

More formally, let the set of ground truth segments be GTS and the set of sensed segments be SS. Each $gts \in GTS$ can match a set $SS_{gts} = \{ss_{gts_1}, ss_{gts_2}, \ldots, ss_{gts_n}\}$ of $ss \in SS$. We can then measure these error characteristics using the following metrics:

$\boldsymbol{\Delta start\_ts}$ $\forall_{gts} \min(|ss_{gts_1}.start\_ts - gts.start\_ts|, T_t)$. Lower is better. Metric is capped at a static threshold of T_t. This captures the first kind of error.

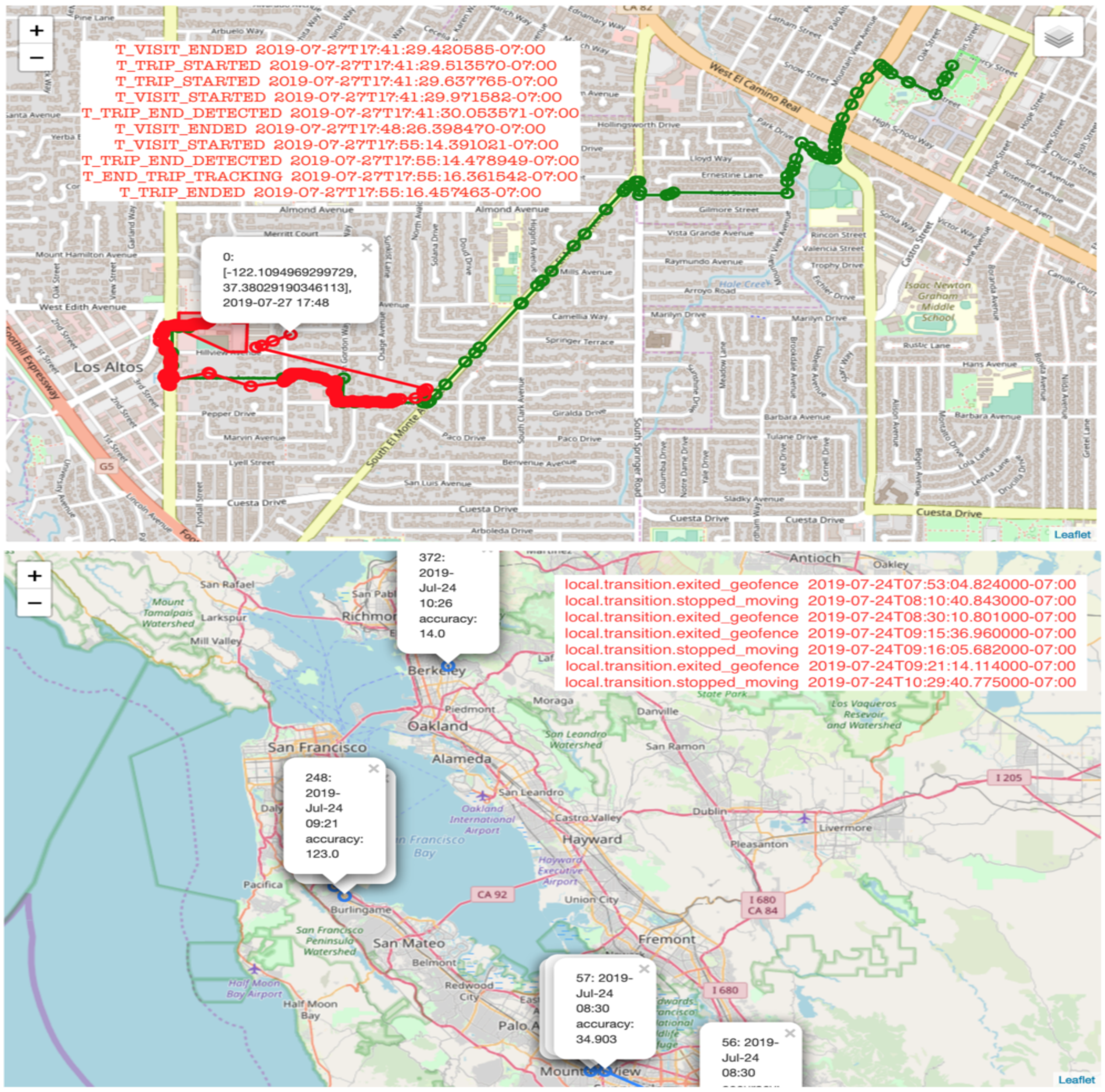
$\boldsymbol{\Delta end\_ts}$ $\forall_{gts} \min(|ss_{gts_n}.end\_ts - gts.end\_ts|, T_t)$. Lower is better. Metric is capped at a static threshold of T_t. This too captures the first kind of error.

$\boldsymbol{\Delta count}$ $\forall_{gts} \min(|n-1|, T_c)$. Lower is better. Metric is capped at a static threshold of T_c. This captures the second and third kinds of errors.

Matching algorithm for evaluation In order to evaluate these metrics, we need to come up with an algorithm that can generate the ss_{gts} given a gts. This is an evaluation algorithm that will be used to evaluate the performance of more complex post-processing algorithms. In order to avoid infinite recursion, it should be simple and deterministic and not involve exhaustive evaluation of its own.

Our proposed matching algorithm has two steps.

[2]https://github.com/e-mission/e-mission-eval-public-data



```
T_VISIT_ENDED 2019-07-27T17:41:29.420585-07:00
T_TRIP_STARTED 2019-07-27T17:41:29.513570-07:00
T_TRIP_STARTED 2019-07-27T17:41:29.637765-07:00
T_VISIT_STARTED 2019-07-27T17:41:29.971582-07:00
T_TRIP_END_DETECTED 2019-07-27T17:41:30.053571-07:00
T_VISIT_ENDED 2019-07-27T17:48:26.398470-07:00
T_VISIT_STARTED 2019-07-27T17:55:14.391021-07:00
T_TRIP_END_DETECTED 2019-07-27T17:55:14.478949-07:00
T_END_TRIP_TRACKING 2019-07-27T17:55:16.361542-07:00
T_TRIP_ENDED 2019-07-27T17:55:16.457463-07:00
```

```
local.transition.exited_geofence 2019-07-24T07:53:04.824000-07:00
local.transition.stopped_moving 2019-07-24T08:10:40.843000-07:00
local.transition.exited_geofence 2019-07-24T08:30:10.801000-07:00
local.transition.exited_geofence 2019-07-24T09:15:36.960000-07:00
local.transition.stopped_moving 2019-07-24T09:16:05.682000-07:00
local.transition.exited_geofence 2019-07-24T09:21:14.114000-07:00
local.transition.stopped_moving 2019-07-24T10:29:40.775000-07:00
```

Figure 7.3. **Examples of errors in segmentation captured by the segmentation metrics**. *Top: large error in the trip start time for an iOS **HAHFDC** run.* The green line is the ground truth, red line is the sensed data. We got a visit end (trip start) transition at 17:41, but we detected a trip end within 30 ms so we did not sense any data. The next trip start was at 17:48, when we did start reading values, but this was almost the end of the trip.*Bottom: error in segmentation trip count for an android **HAMFDC** run.* On one multi-modal trip, we get multiple trip start and end transitions, largely corresponding to transit transfers. Note also that there at 8:30, there are two consecutive geofence exits (08:30 and 09:15) and an erroneous point in San Jose.

1. The first step, which is only applicable while evaluating raw sensor data, converts a sequence of *transitions* (e.g., `VISIT_ENDED, VISIT_STARTED`) into candidate *ranges* by matching start and end transitions. This is not applicable while evaluating post-processed data, since the output of the post-processing step will already generate segments.

   Input Set of transitions (S—E)* with some potentially missing or duplicated

   Output Pairs of (S, E) transitions that define the sensed ranges

   Implementation For each S, find the first corresponding E. Any intermediate unexpected transitions are ignored — e.g., $\{S_0, S_1, E_0, E_1, E_2, E_3\} \rightarrow \{S_0, E_0\}$

2. The second step, which is always applicable, matches the ground truth trip or section segment with an arbitrary number of sensed ranges from the previous step.

   Input $GTS = \{gt_1, gt_2, \ldots\}, SS = \{ss_1, ss_2, \ldots\}, \forall ss, ss = (S, E)$

   Output $SS_g \subseteq SS \forall g \in GTS$

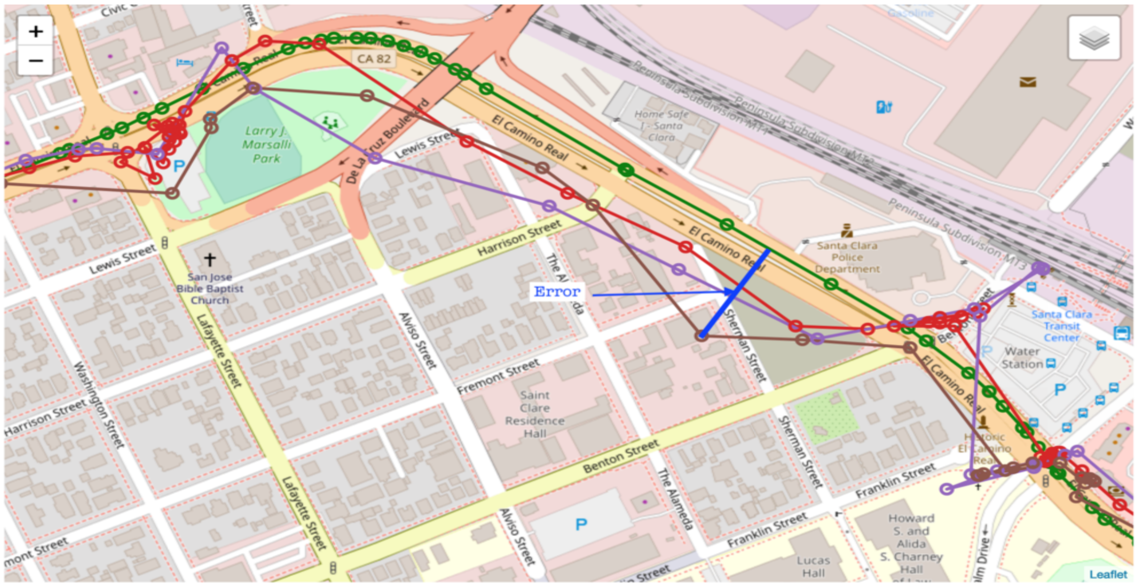
   Implementation For each g find the ss_s with the closest start timestamp and the ss_e with the closest end timestamp. Both matches have threshold of T_c beyond which we will not match any entry. Then, $SS_g = \{ss_s, \ldots ss_e\}$. Note that we match each ground truth segment in isolation, so it is possible for a particular ss to match two separate g. However, because of the threshold on the match, we expect this to be unlikely.

7.2.3.2 Trajectory tracking

The main error conditions for tracking algorithms are:

1. the sensed points are spatially offset from the real trajectory (Figure 7.4, top). The metric for this error condition is fairly straightforward, since we know the spatial ground truth for each evaluation timeline. We can simply compute the error for each sensed point as the shortest distance from the point to the ground truth trajectory. Note that since we compute the error for each sensed point, this metric does not capture large gaps in the sensed data - e.g., the delay in sensing at the start of every trip. Those errors are captured by the segmentation metrics.

2. the sensed points have temporal inconsistencies (Figure 7.4, bottom). It is much harder to determine a metric for this error condition since we do not have spatiotemporal ground truth for trajectories. Computing the spatial distance alone will not capture the error, since the error was caused by repeatedly returning



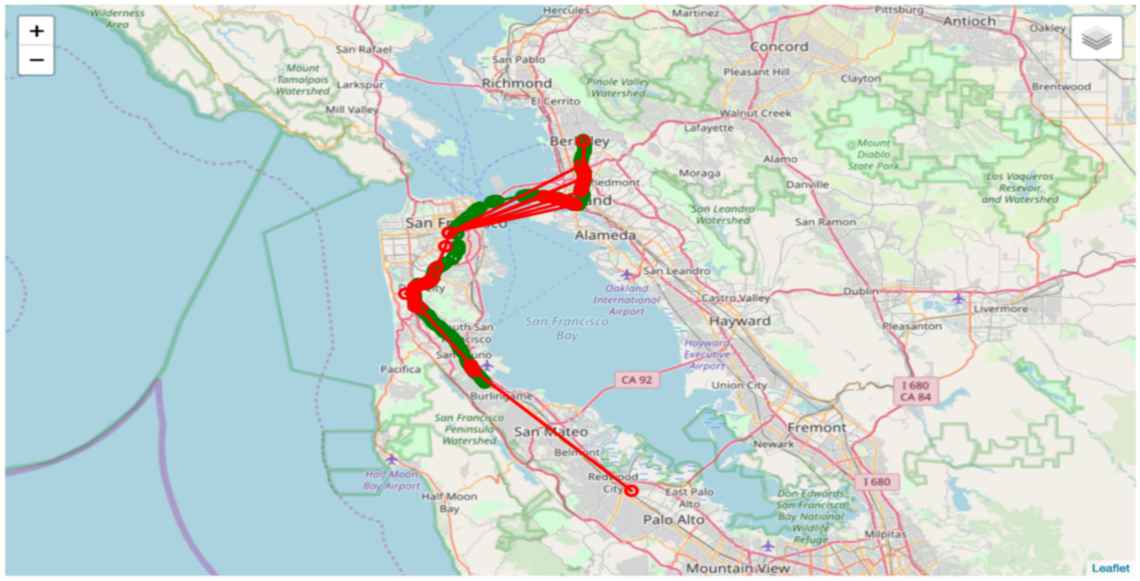


Figure 7.4. **Examples of errors in trajectory captured by the trajectory tracking metrics**. *Top: Spatial tracking errors from multiple iOS* ***MAHFDC*** *runs.* The green line is the ground truth, other colors are the sensed data. For each sensed trajectory, the spatial error is the shortest perpendicular distance to the spatial ground truth line (i.e., the thick blue line from the brown point to the green line. *Bottom: Temporal errors caused by backtracking to previous spatially valid point, based on an android* ***HAMFDC*** *run.* The sensed points in red are largely along the spatial ground truth trajectory in green, but they periodically return to a previous point in the trajectory, generating zigzags. In this case, the spatial error of the repeated points is small, but the temporal error, encompassing cross-bay jumps, is large.

to an earlier point. For this metric, we generate a spatiotemporal reference trajectory for each run based on the accuracy control phones and use it for the comparison. Note that we must construct a reference trajectory *for each run*, since temporal factors (e.g., congestion, transit delays) are likely to be different even for different runs of the same timeline.

Formally, let the set of sensed points for an evaluation run r be P_r. Let the set of corresponding spatial ground truth points be G. Note that the spatial ground truth is not dependent on the run. Let the accuracy control points for android and iOS respectively be ACP_{a_r} and ACP_{i_r}. Let the temporal start and end ground truth for the segment being evaluated be $TGT = \{tgt_s, tgt_e\}$. We can then define the metrics as follows:

$\boldsymbol{\Delta_{sd}}$ Perpendicular distances from the sensed points to the ground truth trajectory. Lower is better.

$$\sqrt{\frac{1}{|P|}\sum_{p \in P_r} d(p, G)^2}$$

$\boldsymbol{\Delta_{std}}$

1. Use the accuracy control and ground truth trajectories to determine a combined spatiotemporal reference trajectory G_r. Reference trajectory calculation is complicated because the accuracy controls have significant error in practice. Note that, unlike spatial ground truth, spatiotemporal ground truth is **run-specific**, due to variations in travel time.
2. Perpendicular distance from the sensed points to the reference trajectory.

$$\sqrt{\frac{1}{|P|}\sum_{p \in P_r} d(p, G_r)^2}$$

7.2.3.3 Classification

Classification metrics are the easiest to work with, since they fit well into classical machine learning paradigms. Since each section has a mode, and we know the ground truth modes, we can simply count the number of correct values to represent the accuracy. However, there are some challenges that are unique to this domain.

1. The list of modes supported by the classifier may be limited. In particular, it may not be easy to distinguish between city and express buses, or between regular bicycles and e-bikes. Therefore, classification algorithms may choose to restrict the set of classes that they support, mapping all bus trips to `BUS` and all bicycling trips to `BICYCLING`.

2. Since we classify sections, the classification accuracy depends on the segmentation accuracy. For example, if a classification algorithm uses the average speed

as a determining factor, but the segmentation combines the walk to the station with the subsequent short train section, the section may be misclassified. We address this by reporting the ratio of the sensed segment that has the correct mode.

Formally, let GTS be the set of ground truth segments for a particular timeline. As with the segmentation metrics, each $gts \in GTS$ can match a sequence $SS_{gts} = \{ss_{gts_1}, ss_{gts_2}, \ldots, ss_{gts_n}\}$ of $ss \in SS$. Note that since we only label modes, we only consider *sections* and not generic *segments* here. Similar metrics can be applied to trip labels (e.g., purpose) if we support them in the future. We can then define the overall, segmentation-dependent accuracy by checking the fraction of time spent in matching modes. Note that this can sometimes be greater than 1, as a spillover from segmentation mismatches (Table 7.3). As close to 1 as possible is better.

$$a_s = \sum_{ss_{gts} \in SS_{gts}, ss_{gts}.label = gts.label} \frac{|ss_{gts}.end\_ts - ss_{gts}.start\_ts|}{|gts.end\_ts - gts.start\_ts|} \forall gts \in GTS$$

Once we have computed these metrics, we can combine them in various ways for comparisons. For example: (i) we can combine the data for a timeline (e.g., `unimodal trip car bike mtv la`) (ii) we can combine the data for a mode (e.g., `CAR`) (iii) we can combine the data for a trip or section (e.g., `freeway driving weekday`)

7.2.4 Power: calibration and evaluation

We start with an understanding of the power considerations for automatic sensing. Our evaluation procedure assumes that the phones are effectively identical and can be directly compared. However, even phones from the same model may have differences in behavior based on minor manufacturing differences or prior usage. We first calibrate the phones to determine whether their behavior is consistent for consistent configurations and identical inputs. We then move to an evaluation phase in which we use the calibrated phones to understand power behavior for a particular OS version under different settings. This section highlights interesting results from both the calibration and evaluation phases.

7.2.4.1 Calibration results

We calibrated the phones by configuring them identically and placing them next to each other on a stationary surface. We then tracked the power drain across time. Since the configurations were identical and the environment was identical, we expected to see that the final power drain for a particular configuration was consistent. We also expected consistent power drain across phones for the same configuration.

Accuracy vs. Frequency: Although there are outliers, repeated experiments allow us to see clear behavior differences. High accuracy, high frequency (HAHF) is

	automotive	confidence	cycling	running	stationary	walking	fmt_time
154	False	medium	False	False	False	**True**	19:01:53-07:00
155	False	high	False	False	False	False	19:02:46-07:00
156	False	medium	False	False	False	**True**	19:02:51-07:00
157	False	high	False	False	False	**True**	19:03:46-07:00
				...			
172	False	medium	False	False	False	**True**	19:17:59-07:00
173	False	high	False	False	False	**True**	19:18:15-07:00
174	False	high	False	False	False	False	19:19:06-07:00
175	False	medium	False	False	False	**True**	19:19:34-07:00
176	False	high	False	False	False	False	19:19:41-07:00
177	False	medium	False	False	False	**True**	19:19:49-07:00
178	False	high	False	False	False	**True**	19:20:04-07:00
179	False	high	False	False	False	False	19:21:16-07:00
180	False	high	False	False	False	**True**	19:21:36-07:00
181	False	high	False	False	False	False	19:22:36-07:00
182	False	high	False	False	True	False	19:27:21-07:00
183	False	high	False	False	False	False	19:28:01-07:00

Table 7.3. **Example of how bad segmentation can lead to classification accuracies** $\gg 1$ using an example fom an iOS `MAHFDC` run. This trip consisted of a `walk_start` section from `18:59:17 -> 19:01:06`, a `suburb_bicycling` section from `19:01:06 -> 19:20:31` and a `walk_end` section from `19:20:31 -> 19:20:57`. However, the sensing API did not detect any cycling (see transitions above), so the only sensed section was `19:01:53 -> 19:27:21, WALKING`. So the ≈ 30 sec long `walk_end` transition matched the entire ≈ 26 min long sensed section, **and** the mode was correct. So the computed accuracy ratio was **5800%!!**

	Transition	Time
0	T_START_TRACKING	2019-09-11T08:10:23-07:00
1	T_INITIALIZE	2019-09-11T08:10:23-07:00
2	T_INIT_COMPLETE	2019-09-11T08:10:24-07:00
3	T_VISIT_ENDED	2019-09-11T08:15:48-07:00
4	T_TRIP_STARTED	2019-09-11T08:15:48-07:00
5	T_TRIP_STARTED	2019-09-11T08:15:48-07:00
6	T_VISIT_STARTED	2019-09-11T08:15:48-07:00
7	T_TRIP_END_DETECTED	2019-09-11T08:15:48-07:00
8	T_VISIT_ENDED	2019-09-11T08:30:33-07:00
9	T_VISIT_STARTED	2019-09-11T10:39:18-07:00
10	T_TRIP_END_DETECTED	2019-09-11T10:39:18-07:00

Table 7.4. **Example of ephemeral trip leading to large iOS trip start delay**. The trip was actually from `08:12:17 -> 10:37:45`. We got a trip start at `08:15:48` but it ended immediately at `08:15:48`. We then got another trip start at `08:30:33` when we actually started collecting data. So even with interpolation to fill the gap at the start, the matching analysed trip was `08:26:23 -> 10:38:02`

the only high power configuration on android, lowering either the accuracy or the frequency leads to essentially the same power drain. On iOS, the frequency is not significant, the only way to reduce the power drain is to lower the accuracy (Figure 7.5, top) to the `100 m` level. This is the level we use for medium accuracy in the rest of this evaluation.

Consistency across phones: While the android phones are largely consistent, there is a small, but persistent difference of $\approx 5\%$ between `iphone-2` and `iphone-3` on iOS. Since these are the intervention phones, we may need to account for this in the power evaluation (Figure 7.5, bottom).

7.2.4.2 Evaluation results

The calibration results imply the following expected behavior for the evaluation:

1. the accuracy control, which has high accuracy, high frequency, always on sensing should result in the greatest power drain
2. lowering the quality (either by lowering the frequency or the accuracy) should result in moderate power drain
3. the power control should have the least power drain.

We do see this behavior on the longest timeline for iOS and the shortest timeline for android (Figure 7.6. However, we also see some unexpected behavior, primarily

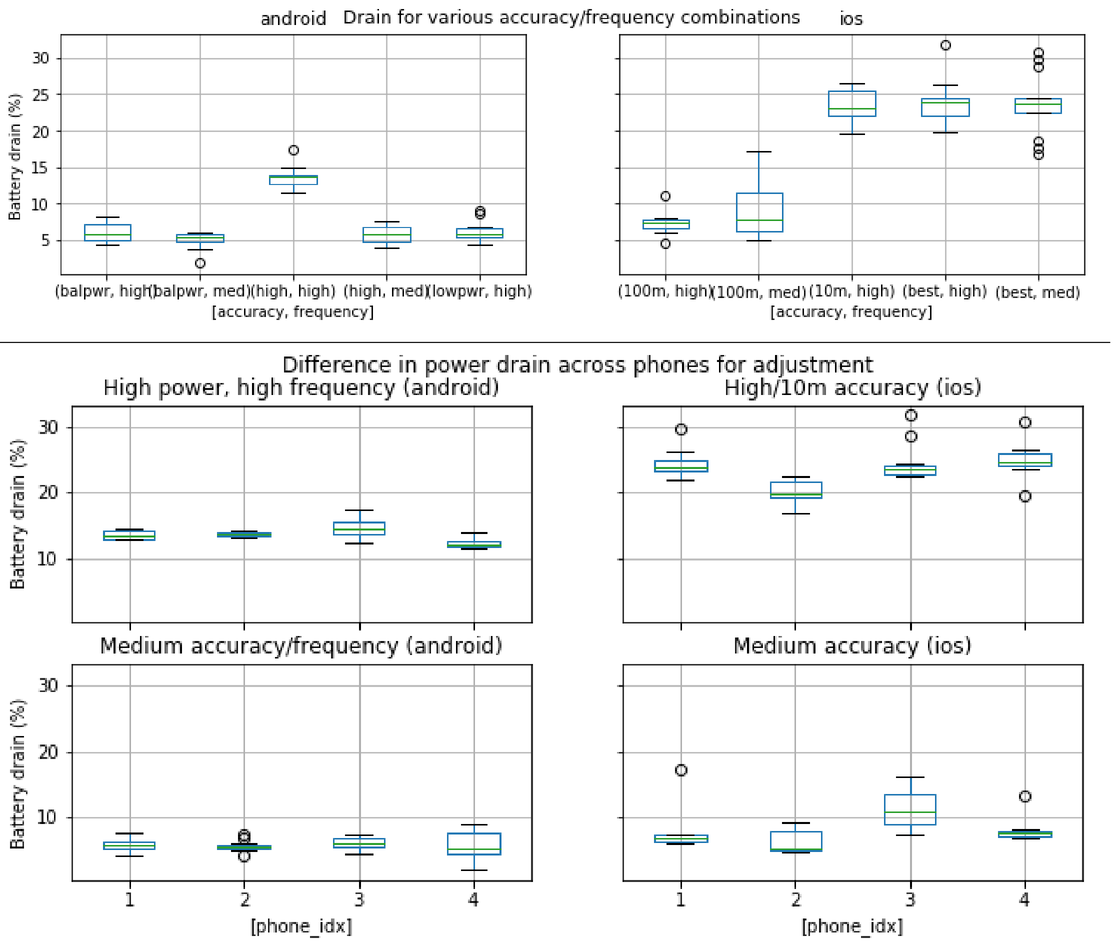


Figure 7.5. **Calibration of stationary phones, configured identically with the specified configuration, with app-based sensing turned always on**. The boxes represent the IQR, the green line represents the median. *Top: variation across configurations:* the x-axis represents the interpolated battery drain after the median duration of the runs. The y axis represents combinations of accuracy and frequency. *Bottom:* variation across phones, pulled out across configurations that show significant differences.

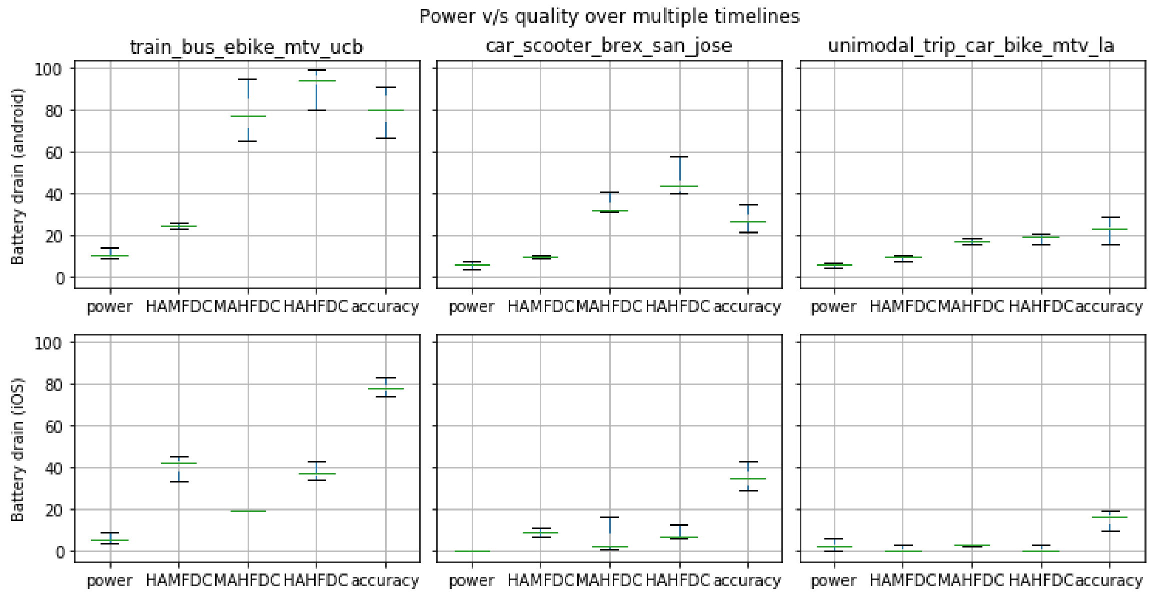


Figure 7.6. **Power drain with different evaluation settings over multiple timelines**. Since we have only three runs each for `MAHFDC` and `HAMFDC`, we show the range directly in the plot instead of boxes and outliers. x axis is the battery drain, y axis has a partial order of sensing quality from L $\rightarrow$ R; `MAHFDC` and `HAMFDC` are both medium quality and not comparable. Timelines are from longest to shortest.

for android high frequency duty cycling. Also, the 5% difference between phones that we found in the calibration phase does not seem to matter significantly. Either it falls within the margin of error (iOS short timelines) or is dwarfed by the differences between configurations (iOS long timeline).

On the shortest timeline for iOS, the differences are small and the error bounds overlap so duty cycled sensing is effectively free. On the medium timeline for iOS, the duty cycled sensing leads to an increased drain of $\approx 10\%$. One of the medium accuracy runs did not duty cycle due to an error in the trip end sensing API, so the `MAHFDC` range is large. Even with that failure, the overall `MAHFDC` is close to `HA*FDC`. Ignoring that failure, medium accuracy sensing is effectively free. On the longest timeline for iOS, medium accuracy sensing ($\approx 15\%$) has roughly half the drain of high accuracy sensing ($\approx 35\%$), which in turn, is half the drain of the accuracy control ($\approx 75\%$).

The android timelines show counterintutive results in which duty cycling at high frequencies performs worse than leaving the sensing on all the time. Our naïve duty cycling (Section 7.2.2.2) is less efficient than the built-in android duty cycling (Figure 6.2) during travel. This result holds for both high and medium accuracies; only lowering the frequency significantly affects the power drain. For the longest timeline, this increased drain cancels out the effect of lowering the quality (`MAHFDC` $\approx$ `accuracy`). Note that this result is different from the stationary calibration result where both lowered accuracy and frequency were roughly equivalent. This is due to the built-in duty cycling on android which automatically throttles sensing when the phone is not moving. The effect of lowering the frequency is significant — for the longest timelines, the additional power drain drops by close to an order of magnitude from $\approx 70\%$ for the `accuracy` control to $\approx 10\%$ for `HAMFDC`.

7.2.5 Trip Segmentation: sensed vs. `master`

This section compares the improvement in the trip segmentation due to the analysis pipeline, based on the segmentation metrics (Section 7.2.3.1). The sensed values represent the raw data from the phone virtual sensors (Section 5.2), and the `master` values represent the analyzed results. We only consider the `master` branch here since the trip segmentation algorithm is identical on both analysis branches.

7.2.5.1 $\Delta count$

Sensed values The sensing regime does not appear to affect the trip count detection, since the ranges for a particular timeline largely overlap, except for `MAHFDC` on android. Timeline length and complexity **does** affect trip counts, with the longer, more complex timelines having the greatest variation.

`master` + `gis` The analysis algorithms significantly improve the trip-level segmentation. For the shorter, simpler timelines, the segmentation is essentially perfect.

For the long and complicated timeline, we detect more segments than reality, but this is due to the dwell time at multi-modal transfers, and the counts are largely consistent across regimes. A more sophisticated algorithm that linked trips starting or ending at transit locations could improve the trip segmentation. However, since train stations frequently incorporate retail establishments, this would also open up the potential for false positives.

7.2.5.2 $\Delta start\_ts, \Delta end\_ts$

Sensed values In general, the phone APIs detect trip starts and ends within 5 mins of the ground truth. There are also numerous outliers corresponding to missed trips caused by poor trip segmentation. The sensing quality makes no difference to the trip start/end detection, except for the `MAHFDC` on android. This setting performs significantly worse, with the median consistently close to the 30 min max. Further investigation shows that this is because we routinely detect the trip end at the start of the next trip. It turns out that with `MAHFDC`, we continuously receive the exact same point at the trip end. Our naïve algorithm strips out duplicate points since they are likely to be poor quality[3], so we don't detect a trip end until the next start.

`master` + `gis` The trip segmentation algorithm for both the master and gis branches is identical, and is a significant improvement over the sensed algorithm. We handle the `MAHFDC` issue correctly, so that the IQR ranges for both trip start and end are < 10 mins. We also note that start trip detection on iOS appears to be worse than android — the IQR range and the median are larger and there are many more outliers. On closer investigation, this is due to ephemeral trips, where the trip started and almost immediately stopped and then restarted again. Since the ephemeral trip was so short, we did not capture any data during that time and cannot reconstruct the trip properly (Table 7.4).

7.2.6 Section Segmentation: sensed vs. `gis`

This section compares the improvement in the section segmentation due to the analysis pipeline, based on the segmentation metrics (Section 7.2.3.1). The sensed values represent the raw data from the phone virtual sensors (Section 5.2), and the `gis` values represent the analyzed results. We only consider the `gis` branch here since the section segmentation algorithm on the `gis` branch was created by modifying the `master` algorithm and is expected to perform better.

[3]issue 45, fixed by commit 4286d60

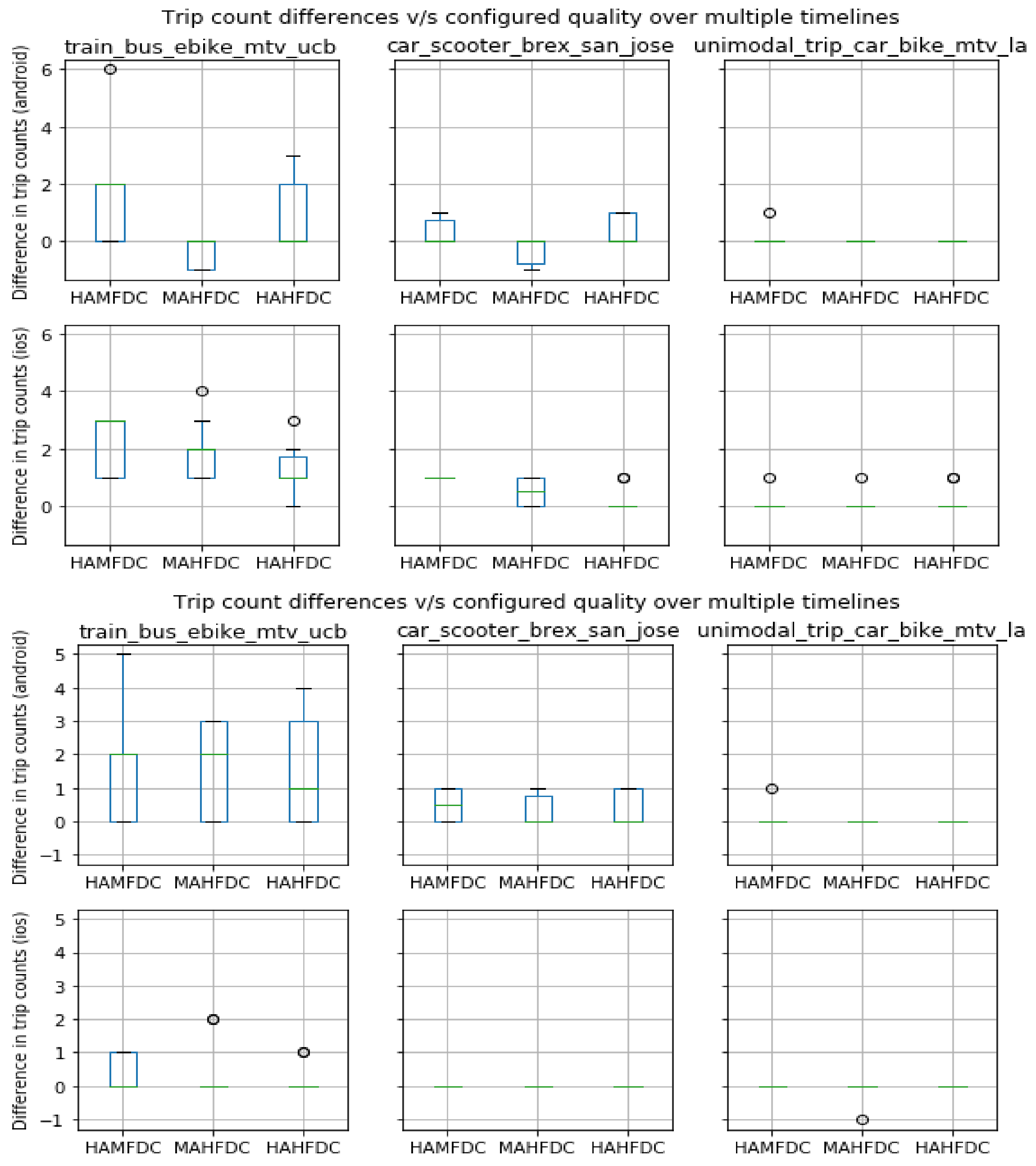


Figure 7.7. **Trip count difference**. Top has the sensed values, bottom has the analysed values. The x axis represents the sensing quality, the y axis represents the number of extra sensed trips matched to each ground truth trip. The ideal value is zero, a positive value indicates too many sensed trips and a negative value indicates too few sensed trips.

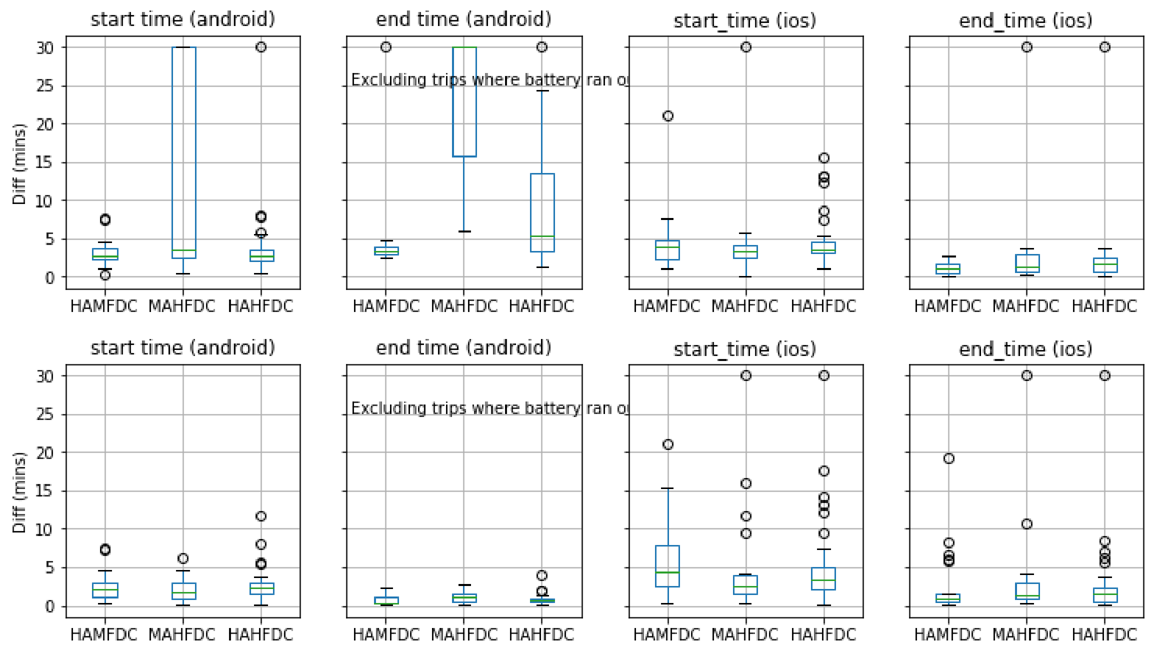


Figure 7.8. **Start/end time difference for trips**. Top has the sensed ranges, bottom has the analysed ranges. x axis is the sensing quality; y axis is the difference in start or end time, in mins. Values are capped at 30 mins and trips where we ran out of battery (e.g., android `HAHFDC` for the longest timeline) are excluded since they don't have a trip end transition.

Section	number with at least one flipflop
suburb_bicycling	4
light_rail_below_above_ground	4
ebike_bikeshare_urban_long	4
city_escooter	3
commuter_rail_with_tunnels	2
subway_underground	2
walk to the bikeshare location	2
city_bus_short	1
freeway_driving_weekday	1
city_bus_rapid_transit	1
walk_to_bus	1
suburb_city_driving_weekend	1
express_bus	1
commuter_rail_aboveground	1

Table 7.5. Frequency distribution of sections that had at least one flipflop ($count_{diff} > 1$). We can see that in general, bicycling/e-scooter trips and transit in mixed traffic have the most flip-flops.

7.2.6.1 $\Delta count$

Sensed values The sensed sections are extremely noisy, with large numbers of outliers. The number of outliers is independent of both the sensing quality and the timeline. This implies that the motion segmentation APIs on the phone do not use location sensing. Summaries of daily activities (e.g., proportion of time spent in each activity) are not affected by small flip-flops, but other aggregate characteristics such as duration of active transportation trips will be strongly affected. Almost every section has only one match. The exceptions are the `HAHFDC` android sections where we ran out of battery and two iOS runs of `walk_urban_university_0`.

`gis` The GIS branch improves the section segmentation algorithms to reduce flip-flopping. There is a dramatic reduction in the number of flip-flops, but there is a modest increase in the number of sections with no match. This is due to extremely poor sensed data (e.g., Figure 7.10) that is interpreted incorrectly by the algorithms. There is still some flip-flopping, notably for bicycling/e-scooter and transit in dense traffic (Table 7.5), but it is limited to a few flip-flops.

7.2.6.2 $\Delta start\_ts, \Delta end\_ts$

Sensed values Since we have a lot of flip-flopping, we are likely to match only a fragment of the real trip and end up with significant differences in both the

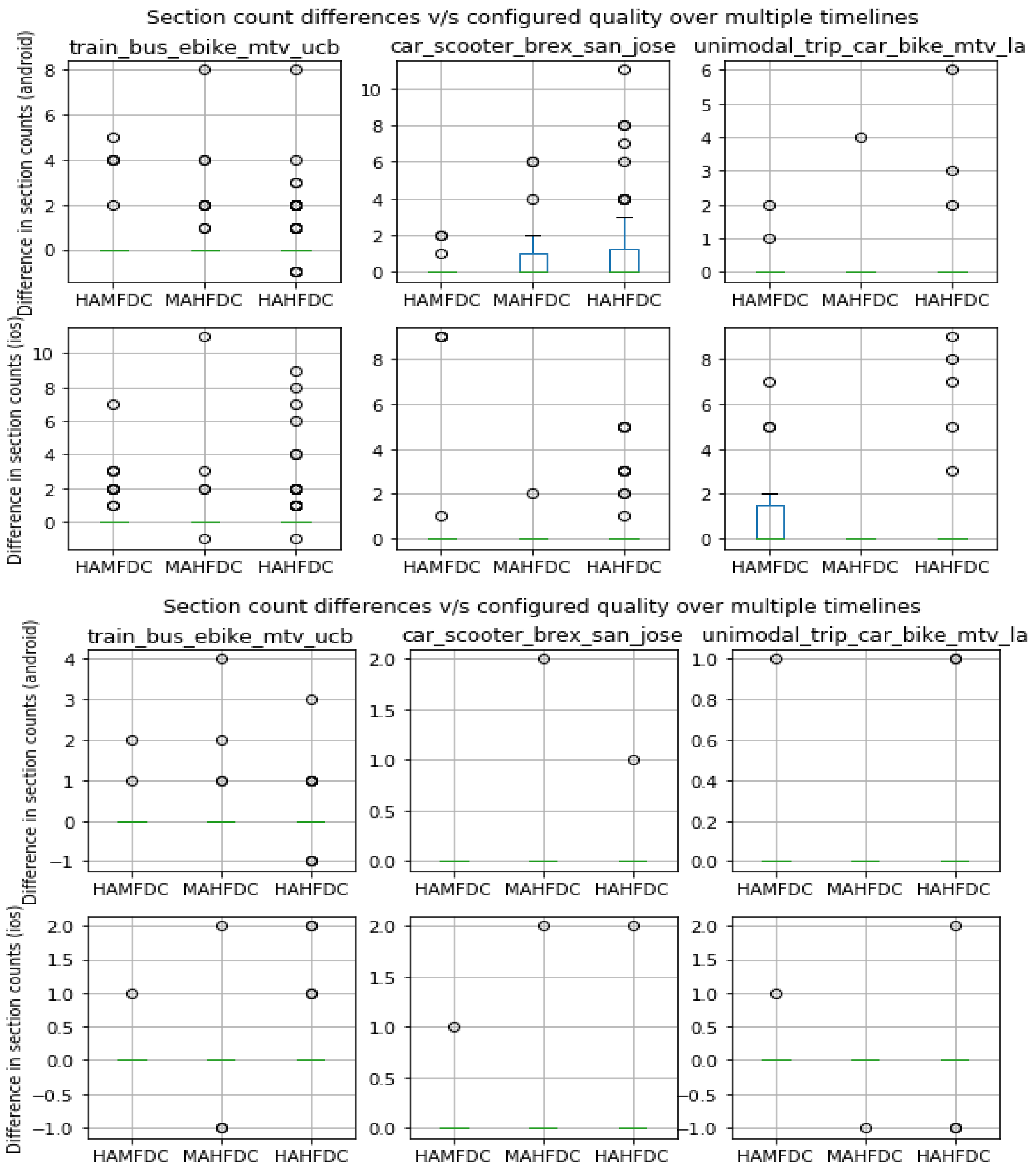
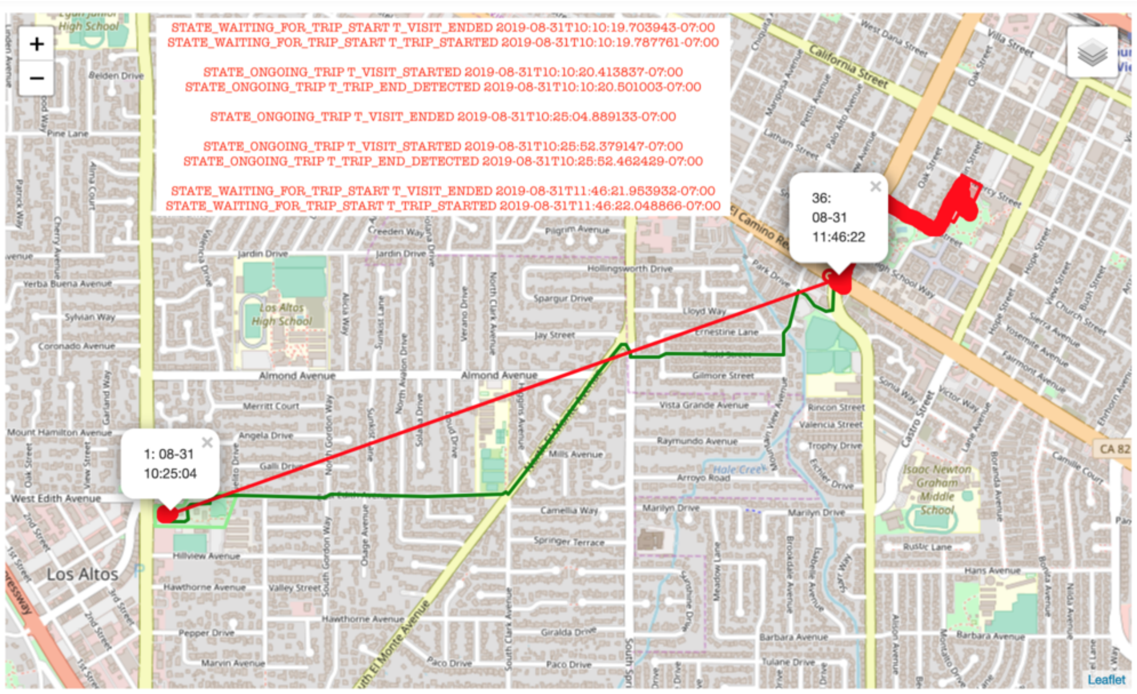


Figure 7.9. **Section count difference** The x axis represents the sensing quality, the y axis represents the number of extra sensed sections matched to each ground truth section. The ideal value is zero, a positive value indicates too many sensed sections and a negative value indicates too few sensed sections.



```
STATE_WAITING_FOR_TRIP_START T_VISIT_ENDED 2019-08-31T10:10:19.703943-07:00
STATE_WAITING_FOR_TRIP_START T_TRIP_STARTED 2019-08-31T10:10:19.787761-07:00

STATE_ONGOING_TRIP T_VISIT_STARTED 2019-08-31T10:10:20.413837-07:00
STATE_ONGOING_TRIP T_TRIP_END_DETECTED 2019-08-31T10:10:20.501003-07:00

STATE_ONGOING_TRIP T_VISIT_ENDED 2019-08-31T10:25:04.889133-07:00

STATE_ONGOING_TRIP T_VISIT_STARTED 2019-08-31T10:25:52.379147-07:00
STATE_ONGOING_TRIP T_TRIP_END_DETECTED 2019-08-31T10:25:52.462429-07:00

STATE_WAITING_FOR_TRIP_START T_VISIT_ENDED 2019-08-31T11:46:21.953932-07:00
STATE_WAITING_FOR_TRIP_START T_TRIP_STARTED 2019-08-31T11:46:22.048866-07:00
```

Figure 7.10. **Example of bad data causing lingering missing sections** The ground truth was two trips, from `10:07:27 -> 10:23:08` and from `11:30:50 -> 11:52:38`. The sensed data included an ephemeral trip from `10:10:19 -> 10:10:20` with no points and from `10:25:04 -> 10:25:52` with some points. Then, we miss most of the return trip before getting a trip start at `11:46:21`. In the post-processing, these are merged into one trip with one section `10:25:07 -> 11:54:17`, so neither of the ground truth sections or trips match.

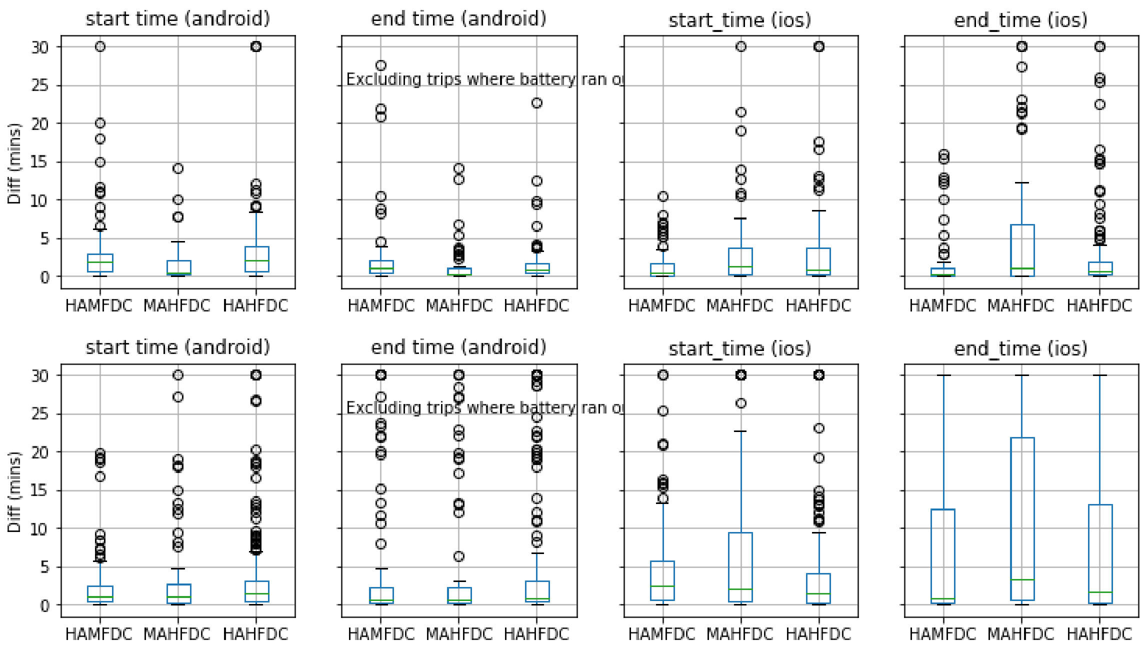


Figure 7.11. **Start/end time difference for sections**. Top has the sensed ranges, bottom has the analysed ranges from the GIS branch. x axis is the sensing quality; y axis is the difference in start or end time, in mins. Values are capped at 30 mins and trips where we ran out of battery (e.g., android `HAHFDC` for the longest timeline) are excluded since they don't have a trip end transition.

start and end times. So while the median delay is low (≈ 1-2 mins), there are significant outliers.

`gis` The flip-flop smoothing that helped with the section counts seems to have made the outliers worse. The median delay is still consistently low but the outliers are worse. It appears that the consolidation of sections sometimes leads to bigger gaps in the start/end detection.

7.2.7 Classification

This section compares the improvement in mode classification of sections due to the analysis pipeline, based on the classification metrics (Section 7.2.3.3). The sensed values represent the raw data from the phone virtual sensors (Section 5.2), and the `gis` values represent the analyzed results. We only consider the `gis` branch here since the classification algorithm on the `gis` branch was created to improve the classification accuracy of the `master` algorithm.

Sensed values Even though there is a lot of flip-flopping, the classification results for sensed data are reasonable. The median for two of the three timelines is around 1, although there are many outliers. Surprisingly, the accuracy of shortest, simplest trajectory is the worst, and needs to be investigated further.

`gis` The flip-flop smoothing that helped with the section counts really helps with the classification as well. There are now almost no outliers in any of the timelines. The best overall timeline is the intermediate one. The most complex one has a fairly low median on android and many outliers on iOS. Note that, the GIS sections support more modes than the sensed ones; in particular, they distinguish between vehicular modes such as `BUS, CAR, TRAIN, SUBWAY` etc.

7.2.8 Trajectory: sensed vs. `master`

The spatial and spatiotemporal accuracy results (Figure 7.13, 7.14) are remarkably similar in the absence of outliers, although the spatiotemporal error is $\approx$ 2x the corresponding spatial error. At high quality, duty cycling doesn't affect the error significantly. However, the lower quality results are surprising. On android, `HAMFDC`, which should use high accuracy, GPS-based sensing is **worse** overall than `MAHFDC`, which should use medium accuracy, Wi-Fi/cell tower sensing. These results are flipped for iOS. Since these are the regimes with the lowest power drain (Section 7.2.4.2), we have a clear power/accuracy trade-off for trajectory sensing.

The spatial results with outliers are similar, with `HAMFDC` on android and `MAHFDC` on iOS having more outliers than the other regimes. For the spatiotemporal metrics, the results on iOS change, with `MAHFDC` having the most outliers. This seems to indicate that the medium accuracy data is (i) more susceptible to temporal errors, like returning prior points along the spatial ground truth (Figure 7.4), and (ii) more sparse, returning no entries if the sensors are unavailable. Recall that for spatiotemporal data, we resample the entries at 1 sec frequency and calculate the distance at matching temporal points.

The smoothing algorithm works well, but is not perfect. The smoothing algorithm does not affect values within the IQR, it is largely an outlier detection and removal algorithm. If there are several outliers in a row, then the smoothed trajectory joins the end points with a straight line, leading to smaller outliers (intuition in Figure 7.15,7.16). Using a map matching algorithm that supports multi-modal travel, although complex, would potentially remove these outliers.

Concretely, after the algorithm is run the max outlier for spatial errors on android goes down from 20 km to 15 km, and there are a lot fewer outliers. On iOS, the max outlier goes from 15 km to 5 km. For spatiotemporal errors, the max outlier goes from 40 km to 15 km on android and from 20 km to 5 km on iOS.

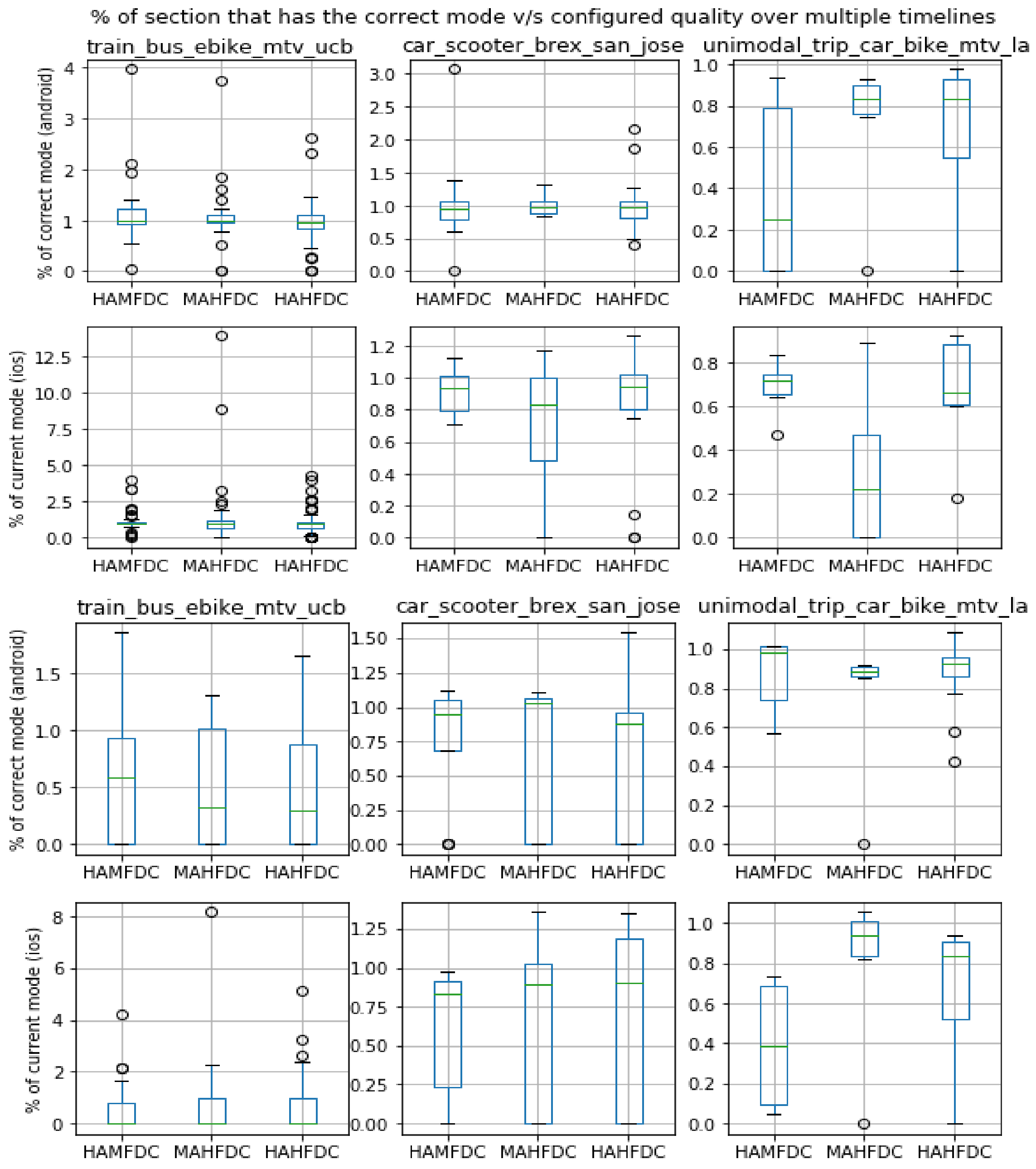


Figure 7.12. **Classification accuracy percentages**. Top has the sensed ranges, bottom has the analyzed ranges from the GIS branch. x axis is the sensing quality; y axis is the percentage of time in the section that is spent in the correct mode. 1 is the ideal value, values of > 1 or 0 typically indicate bad segmentation (Figure 7.3). Only `TRAVEL` sections are considered, and sections where we ran out of battery (e.g., android `commuter_rail_with_tunnels_0` and `inner_suburb_downtown_walk_0`) are excluded.

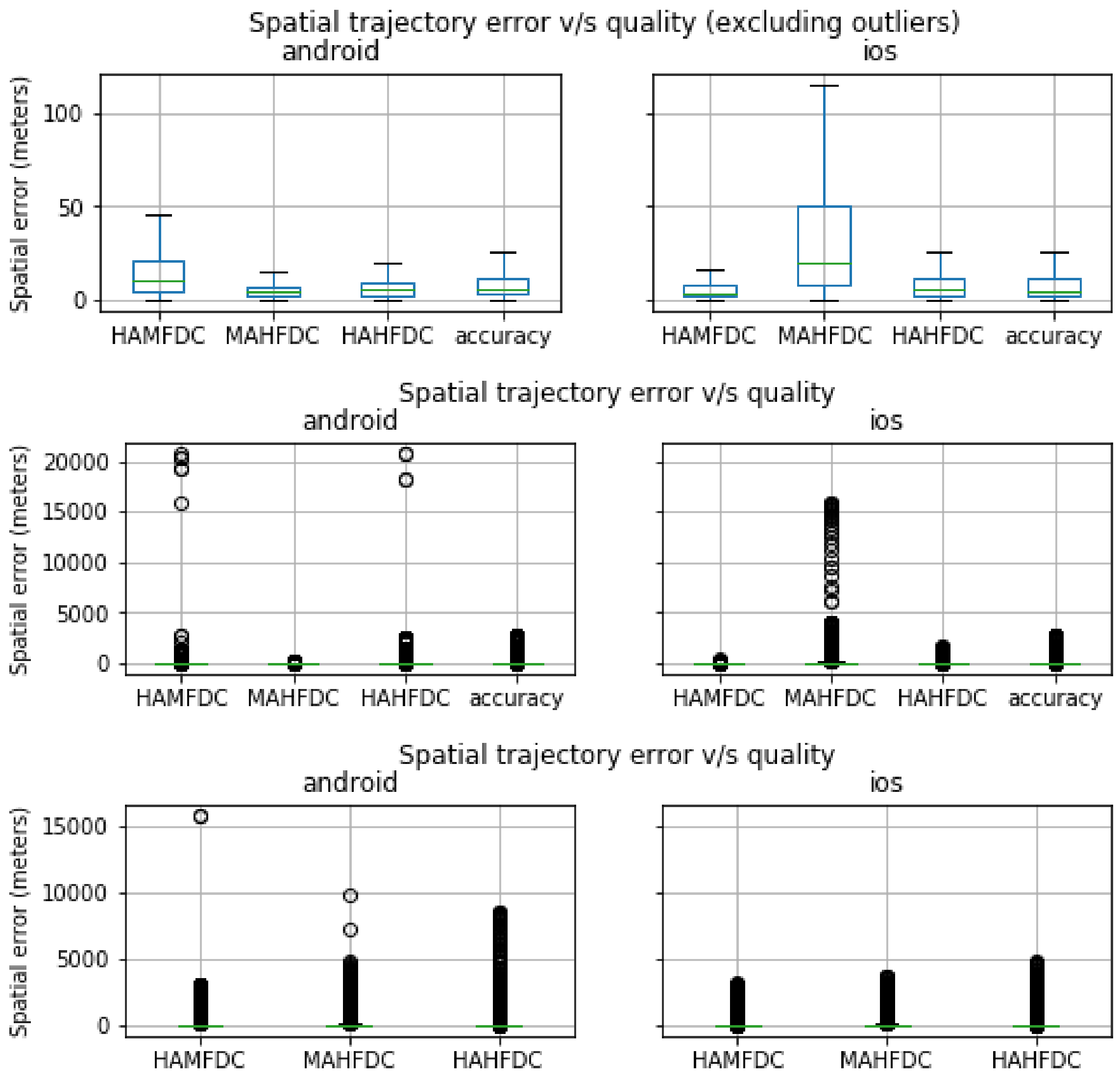

Spatial trajectory error v/s quality (excluding outliers)
android
ios
Spatial error (meters)
0
50
100
HAMFDC
MAHFDC
HAHFDC
accuracy
Spatial trajectory error v/s quality
android
ios
Spatial error (meters)
0
5000
10000
15000
20000
HAMFDC
MAHFDC
HAHFDC
accuracy
Spatial trajectory error v/s quality
android
ios
Spatial error (meters)
0
5000
10000
15000
HAMFDC
MAHFDC
HAHFDC

Figure 7.13.

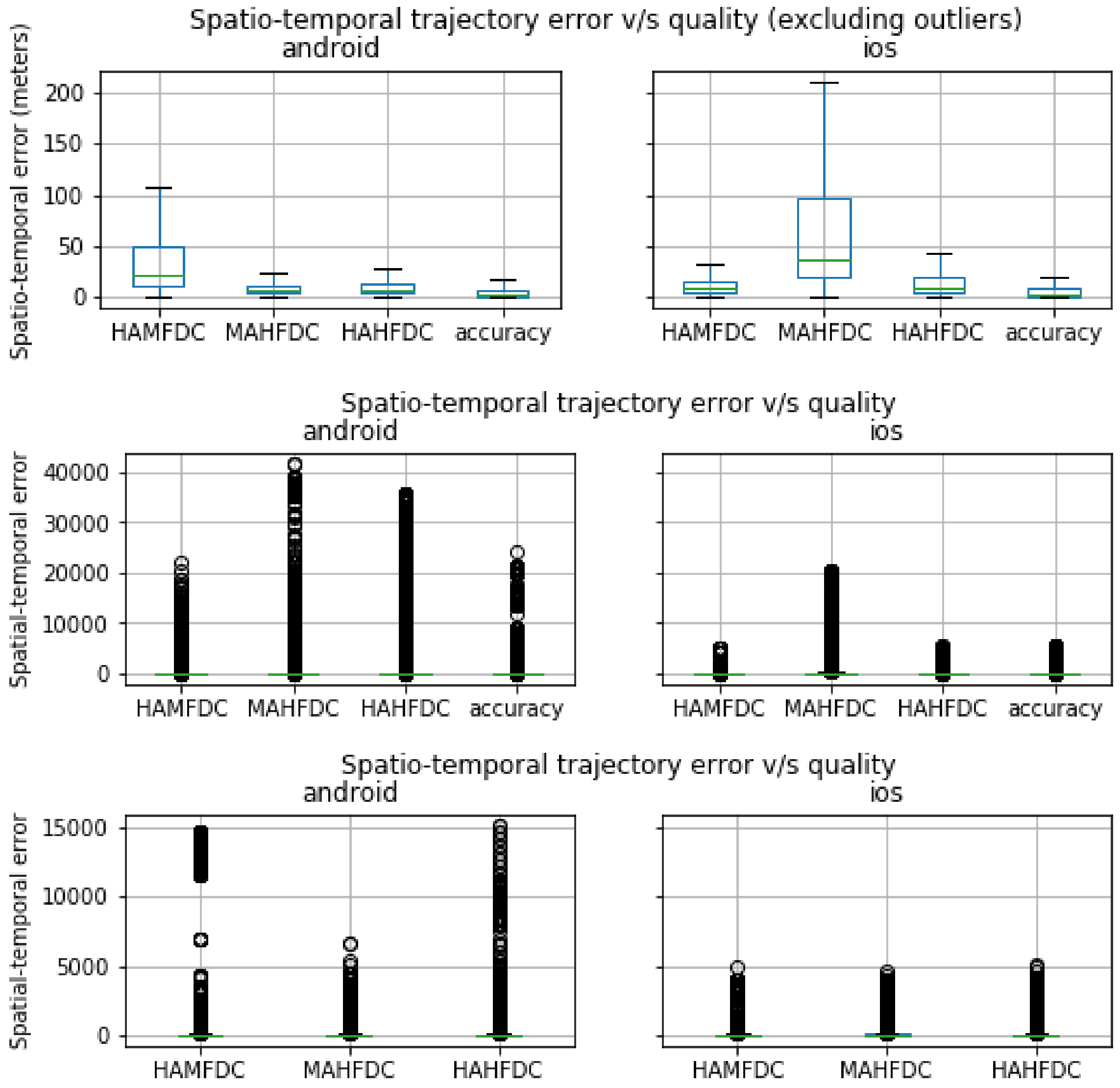
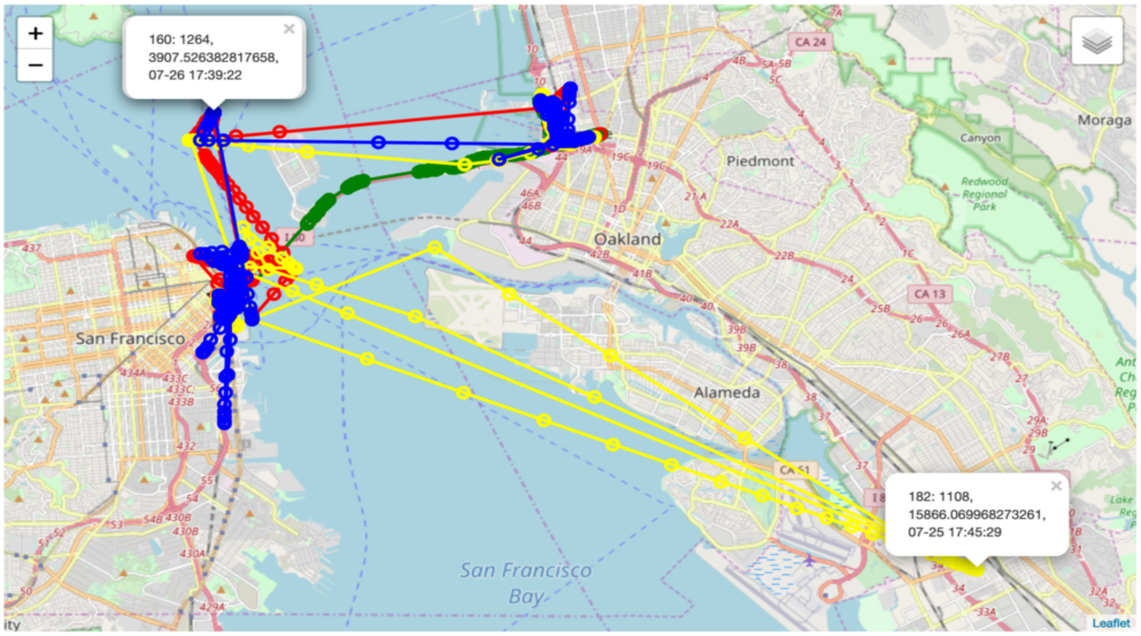

Spatio-temporal trajectory error v/s quality (excluding outliers)
android
ios
Spatio-temporal error (meters)
HAMFDC
MAHFDC
HAHFDC
accuracy
Spatio-temporal trajectory error v/s quality
android
ios
Spatial-temporal error
HAMFDC
MAHFDC
HAHFDC
accuracy
Spatio-temporal trajectory error v/s quality
android
ios
Spatial-temporal error
HAMFDC
MAHFDC
HAHFDC

Figure 7.14.



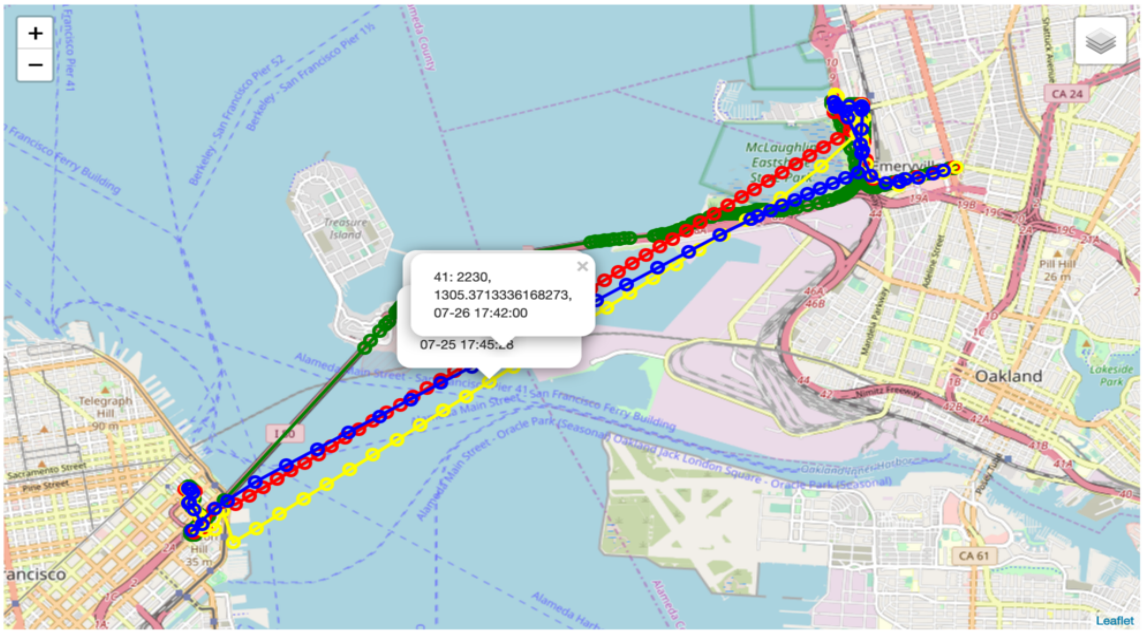
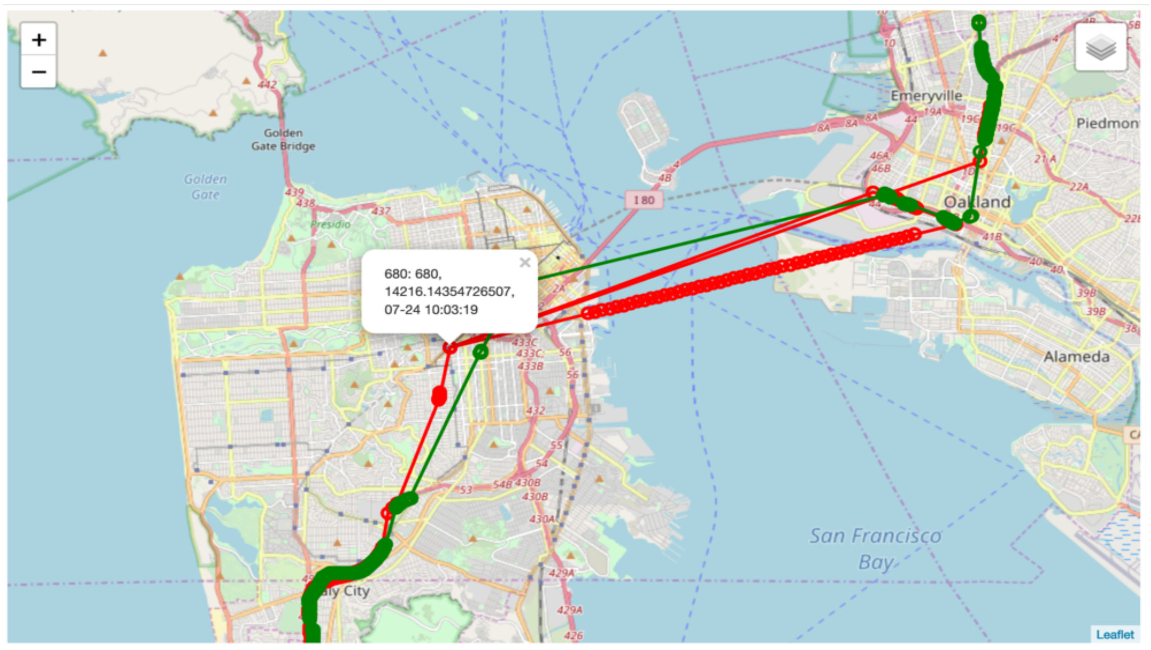


Figure 7.15. **Example of smoothing in the case of high spatial error**. The green line is the ground truth and the three colored lines are the sensed values for three runs of the `express_bus` section on iOS with the `MAHFDC` configuration. The bus route is entirely above ground but on a bridge. The popups indicate the point with the greatest error for each run. The sensed data (top) shows significant outliers, including one in Fremont. The smoothing (bottom) removes the outliers, but creates a straight line that is still 1km away from the ground truth at its furthest point.
.



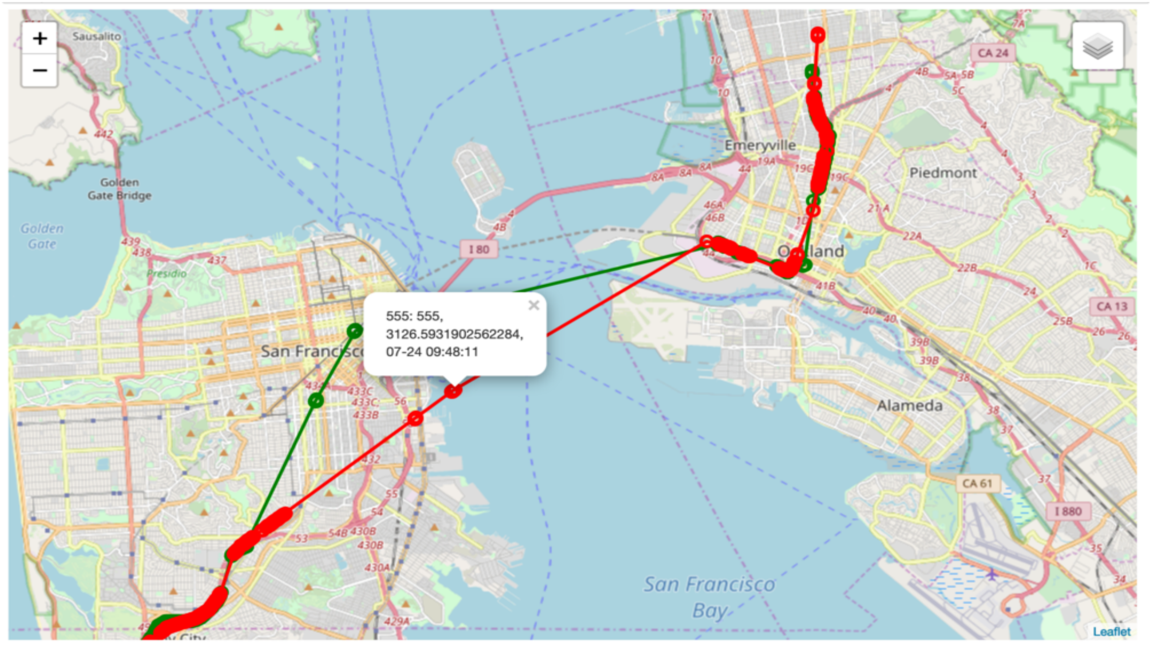


Figure 7.16. **Example of smoothing in the case of high spatial-temporal error**. The green line is the ground truth and the red line is the sensed value for the first run of the `subway_underground` section on android with the `HAMFDC` configuration. The subway route goes underground in the middle of San Francisco and comes aboveground after crossing the bay. The popup indicates the point with the greatest error. The sensed data (top) shows significant cross-bay (14km) zigzags to a point along the spatial ground truth. The smoothing (bottom) removes the zigzags, but creates a straight line that is still 3km away from the ground truth.

7.3 Conclusion

Through a rigorous, quantitative evaluation of the **platform architecture**, we show that the e-mission platform is: (i) *extensible*, since all customization was to non-core modules; all module extensions could be performed without rewriting other modules, and (ii) *useful*, since the time taken to create a custom "app" for a new project was < 3 months of part-time undergraduate time.

Through a rigorous, quantitative evaluation of the **inferred accuracy**, we show that: (i) the biggest determinant of *power drain* on android is frequency and on iOS is accuracy. (ii) the analysis algorithms can work with *multiple sensing settings*, (iii) trips are typically detected within 5 minutes, sections within 2 minutes, and (iv) the typical trajectory accuracy is within 50 m even for the lower power sensing options for travel with significant underground sections.

After having shown that the platform works well and can be customized to meet the needs of variety of human mobility systems, we conclude with some thoughts on future bi-directional improvements, propose a methodological innovation, and outline some tips for others tackling extensible platforms building in the future.

Chapter 8

Conclusion and Future work

This thesis outlines an extensible platform that can serve as the foundation for multiple Computational Mobility (CM) projects. The platform is an active open source project, with deployment interest from every inhabited continent other than Africa, and multiple contributions from external groups.

The platform has a well-defined **modular** architecture that has been validated by 3 different real-world use cases. The implemented systems used an average of 64% of the features of the platform, with approximately 3-4 months of part-time CS undergraduate time for each new case. Every use case contributed at least one extension, primarily client-related, back to the platform.

The platform uses virtual sensors that allow systems to choose to operate at varying levels of **accuracy**. The accuracy is evaluated through a novel procedure that then captures the power vs. accuracy vs. analysis trade-offs for various settings. The settings include duty cycling to enable medium grained sensing at low power, and phone based motion activities for low power mode classification. The thesis defines a set of three timelines of varying lengths in the San Francisco Bay Area and uses the procedure to collect a public dataset for them. Using this dataset, the evaluation shows that power drain on iOS is sensitive to accuracy and on android is sensitive to frequency. It also shows that collection settings primarily affect the trajectory, and analysis improves the trajectory and segmentation count metrics the most.

The platform has a well-defined analysis pipeline operating with a data model that enables **reproducibility**. The input data is read-only and generates transformed copies as it passes through the pipeline. This ensures that all steps without external integrations generate the same results when re-run on the same inputs. It implements modified versions of the standard trip diary algorithms that work with virtual sensor data.

The rest of this chapter is structured as follows. It starts with discussing CS $\leftrightarrow$ domain improvements in both directions: (i) *CS* $\rightarrow$ *domain* enhancements related to benchmarking, plugin-based architecture and reinforcement learning (Section 8.1) and (ii) *CS* $\leftarrow$ *domain* transfer in areas such as privacy, trustworthiness, incentivization and decision-making (Section 8.2). It then introduces *Agile Urban Planning*, a methodological innovation in mobility enabled by these improvements (Section 8.3). Finally, it then takes a philosophical turn, discussing the potential for data misuse and some related technical solutions (Section 8.4), before ending with speculation on extension to more general use (Section 8.5).

8.1 Future application of CS $\rightarrow$ domain

The current e-mission platform is a bottom-up instantiation which focuses on satisfying the requirements for CM while adhering to CS philosophies. It meets the basic requirements well enough to have wide adoption and an active community with multiple external contributions. Examples of recent contributions include native app integration by UW (USA) [1], internationalization support from FabMob (France)[2], country-specific carbon footprint calculations from the Open Source Lab (Germany)[3], ODK XForms support using EnketoCore in collaboration with UNSW (Australia)[4].

There are other, more complex enhancements derived from CS concepts that can improve the functionality of the platform and enhance its novelty over related closed source products.

Benchmarking Current e-mission analysis accuracy can be poor for complex trips with underground components. Better accuracies typically require better sensing, either at higher quality or with more sensors, which can consume more battery. A standard specification-based benchmark similar to TPC could evaluate the *system* and capture various points in the power/accuracy trade-off. A high quality, privacy preserving, cross-platform, ground-truthed dataset similar to ImageNet could be generated through the benchmark and evaluate *algorithms*. In both cases, CS approaches to evaluation can help accelerate the development of the field.

Pluginize architecture As various modules of the platform compete on performance, they form a family of implementations. Although the platform architecture has well-defined modules, alternate implementations are implemented within the same repository and code changes are required to switch between them. Supporting plugins will allow greater independence between modules, and potential security improvements, similar to the shift from a monolithic kernel to a microkernel in classic Operating Systems.

[1] `https://github.com/e-mission/e-mission-docs/issues/410`
[2] `https://github.com/e-mission/e-mission-phone/pull/584`
[3] `https://github.com/e-mission/e-mission-phone/pull/597`
[4] `https://github.com/e-mission/e-mission-phone/pull/581`

Reinforcement learning Better analysis algorithms may be able to improve the accuracy of *sensor-separable* inference, but CM would like to be able to distinguish between a rich set of modes. Some of these modes, such as single occupant vehicle vs. carpool vs. ride-hailing, are indistinguishable based on location data alone. Introducing a standard reinforcement learning framework can help in expanding the inferences supported without excessively increasing user burden.

8.2 CS use-inspired research $\leftarrow$ domain usage

For CM to be truly interdisciplinary, there must be a bi-directional exchange of ideas between CS and the domain. Since the CS $\leftarrow$ research areas are inspired by the usage in the domain, the CS $\rightarrow$ domain changes had to be implemented first.

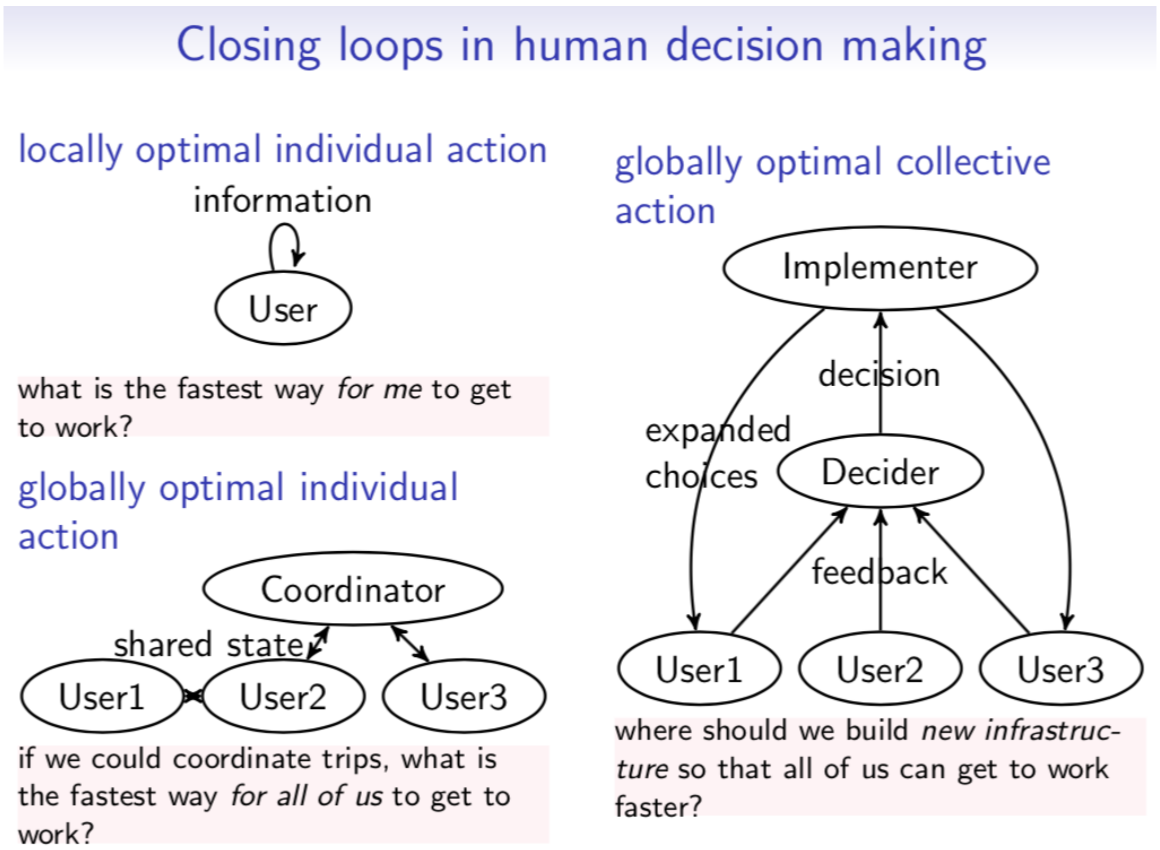
This thesis outlines the application of Computer *Science* philosophy to the domain, by building a reusable, extensible, rigorously evaluated platform with reproducible analysis. CM can now build on this foundation and the challenges encountered in real world deployments to areas where we can broaden our understanding of CS problems. We have already started this work, with the creation of a differential private framework that supports fine-grained aggregate queries([Sul19]).

It is ironic, but perhaps inevitable with interdisciplinary research, that the conventional CS research topics will be tackled in work performed after this thesis.

privacy Infrastructure sensors typically focus on data gathering at a particular point. The closest pre-sensor analog is a human with a clipboard standing next to the street and counting cars, bikes or humans. Trip diaries focus on data gathering about an individual. The closest per-sensor analog is a human *tail* who follows you around and sees everything that you do. The second is clearly creepier than the first, and raises significant privacy concerns. Similar issues arise in almost all background sensing, e.g., always-on, voice controlled smart phone controllers or always-on surveillance cameras for monitoring household workers (*nannycams*).

trustworthiness While no sensor is error-free, sensors can be calibrated and their errors accounted for in future analysis. However, collecting and analyzing large quantities of *qualitative* data computationally raises new challenges. How do we know that respondents are truthful? And even if they are, how do we know that perceptual data from one traveler is comparable to data from another traveler? Similar issues arise in almost all online platforms with user-generated content such as social media posts or product reviews.

incentivization Installing infrastructure sensors only requires convincing the local authorities, who can see clear benefits to using the resulting data. Collecting


Closing loops in human decision making
locally optimal individual action
information
User
globally optimal individual action
Coordinator
shared state
User1
User2
User3
globally optimal collective action
Implementer
decision
expanded choices
Decider
feedback
User1
User2
User3

what is the fastest way *for me* to get to work?

if we could coordinate trips, what is the fastest way *for all of us* to get to work?

where should we build *new infrastructure* so that all of us can get to work faster?

Figure 8.1. Example challenge: Indirect, collective decision-making

data from individual travelers requires convincing multiple travelers to contribute data, each of whom may see little direct benefit from the result. How can we convince travelers to install our mobility sensors and contribute their data? Can we use this personalized data to modify behavior instead of just instrumenting it? Similar issues rise in the field of persuasive technology, which a sub-field of Human Computer Interaction, and includes work on healthy eating habits, and the use of behavioral techniques (e.g., nudges) for social good.

decision-making Current software-supported decision-making typically happens at the individual level. There is some research interest in coordinated decision-making, but decision-making about infrastructure is inherently more challenging since it needs to balance competing interests (Figure 8.1). CS researchers have called for an Intelligent Infrastructure (II) revolution, similar to the current AI revolution [Jor18].

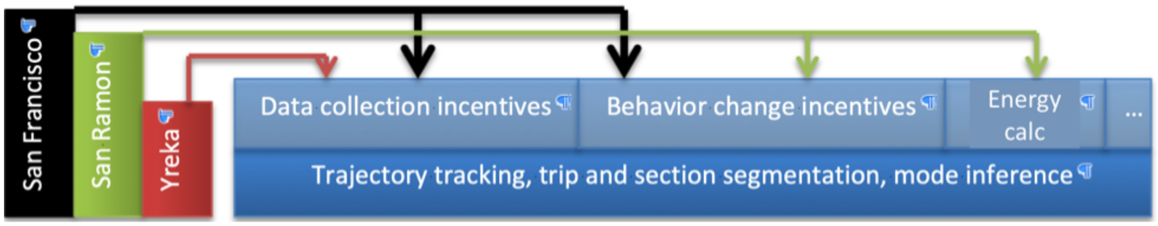
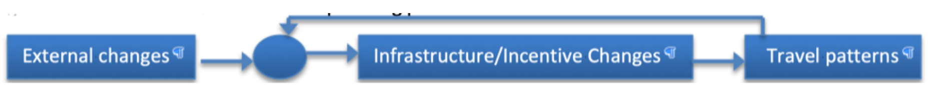

San Francisco
San Ramon
Yreka
Data collection incentives
Behavior change incentives
Energy calc
...
Trajectory tracking, trip and section segmentation, mode inference

Figure 8.2. Solutions tailored for large cities (San Francisco), auto-dependent suburbs (San Ramon), and rural small towns (Yreka) can be assembled from pre-tested incentives built on top of open, customizable components for tracking and analysis

8.3 Broader Impact: Agile Urban Planning



Agile urban planning is the idea that we can prototype changes in urban environments in order to quickly determine which are most promising. Local governments that adopt agile urban planning practices can introduce a control feedback loop that helps them meet their sustainability goals. They can apply local travel patterns to existing models to develop some combination of infrastructure and incentive changes, and use the resulting shifts in travel patterns to propose new changes until their overall goals are reached. Once the system is in equilibrium, it can also sense disruptive external changes (e.g., advent of TNCs) and track ongoing compliance with goals (e.g., congestion levels).

This idea also indicates the need for the translational work in this thesis, and its focus on standardization, reuse and replicability. While industry partnerships are key to large-scale adoption, policy-making based solely on proprietary solutions is problematic. There are well-known biases in commercial AI systems [YGW13], examples of suppression and falsification of pollution data[5], and ethical concerns with requiring citizens to provide data to third parties that monetize their data in order to participate fully in civic life[6]. A better approach is for scientists to focus on building an extensible, robust, open foundation. Industry partners can build innovative solutions tailored to local conditions on top of this foundation (Figure 8.2).

8.4 Potential risks of tracking location data

Traditional Computer Science theses have typically not explored the ethical implications of their research. However, this interdisciplinary thesis needs to speak to

[5]https://www.nytimes.com/2015/09/23/business/international/volkswagen-diesel-car-scandal.html

[6]https://www.nytimes.com/2019/09/24/opinion/public-broadcasting-facebook.html

both domains, and that entails a high-level discussion of the potentially negative impacts as well.

The primary ethical concern with this work is the ability of the stored location information to be misused. It could be misused (i) *openly*, by repressive regimes for mass surveillance of their citizens, and (ii) *covertly*, by data collectors for discrimination or monteization, or (iii) *illegally*, by hackers who gain access to the stored data without consent.

This section discusses the safeguards against each scenario built in to the system, and the expectations that it places on builders, deployers and end-users.

8.4.1 Open surveillance by repressive regimes

Technologically advanced repressive regimes can force their population to install tracking software on their phones and use it for mass surveillance. However, any regime with such goals would not need this platform. There are multiple commercial one-off systems with similar functionality that they can choose from. Given the ease of use of the location virtual sensors (Section 5.2), any regime with the technical capability to store and analyze the large-scale data generated could also easily write a custom app to collect the data, just like the many one-off systems generated by transportation researchers ([PSCM18, JMA$^+$13, SAM$^+$15]). Most of the work in this thesis deals with lowering barriers to adoption by addressing issues like power drain, reuse, extensibility, inference accuracy and reproducibility. None of these are particularly relevant to a repressive regime that can force citizens to participate in data collection, manually delete outlier points and label the purpose of all their trips.

8.4.2 Covert misuse by data collectors

With a classic centralized architecture like this one (Section 3.3), the system maintainer has absolute control of the data from a technical perspective. The only restrictions on usage and sharing of the data are legal.

Commercial entities that collect data as part of providing services have to abide by the terms and conditions that their users agree to. Commercial entities that contract with institutions typically share the data with them and are expected to abide by contractual limits on its usage. However, the lack of technical barriers, and the abstruse language in boilerplate contracts makes enforcement difficult or impossible.

With an open platform, the institution can collect the data directly on servers that it controls. This can make unauthorized data sharing (e.g., for monetization), less likely. However, the potential for discrimination persists. For example, if an employer tracks employee commutes as part of a Transportation Demand Management(TDM) program, they might use their arrival and departure times as a virtual time card. Such concerns could be partially alleviated by having the local government in the city run the server and share TDM-specific data with employers. But that would give rise to new concerns about potential infringements of civil liberties. In the best traditions of interdisciplinary work, this technological change allows urban planners to explore locally specific questions around institutional trust.

The long-term solution, however, should be a decentralized architecture that allows users own their own data. The restrictions on usage and sharing of the data should be technical, which will make enforcement dramatically easier. This is an open research problem in privacy, although we have started addressing it [Sul19]. Note that platform deployers don't have to support the decentralized architecture, but users don't need to patronize such deployers.

8.4.3 Illegal access by hackers

Democratizing data collection by allowing institutions to participate directly requires them to be familiar with basic sysadmin operating procedure. This may be more challenging than it originally seems.

Experiences with deploying e-mission in conjunction with our partners indicate that even domain experts familiar with computers routinely underestimate the complexity of ops work. Straightforward tasks such as configuring SSL can take months to complete. UI-based access directly to the database is common, proper tunneling over ssh is less so. And nobody other than the reference implementation appears to encrypt their logs.

This is not likely to be fixed by user control of data. Even if the platform successfully supports a decentralized architecture with user control, the OS and hardware that the data is stored on are subject to hacker attacks. It is almost impossible for end-users to maintain cloud data storage systems. End-to-end encryption could allow maintenance to be performed by experts without compromising privacy, but hackers could still deploy phishing attacks.

However, although there is no perfect technical solution, there are many avenues to reduce risk that can be incorporated into the platform. Better engineering and deployment support that incorporates best practices by default, similar to SSL certificates from Let's Encrypt, can make it more likely that deployments will be configured properly. Enclaves and end-to-end encryption can protect data stored in the cloud. And better user education can reduce the chance of phishing attacks.

8.5 A Beginning, Not an End

This thesis has focused squarely on the interplay between computation and mobility. Even the future work has dwelt on how to influence mobility using computational concepts and how to use problems from mobility to inspire computational research. We end this journey by highlighting the new beginnings — in other fields, other domains, and for other such platform builders that this work enables.

Applying other fields to mobility An extensible platform provides a base for researchers from other fields to test theories without significant tooling effort. While mobility researchers are the most likely to use this platform, researchers from other domains can also use it to apply their expertise to the mobility domain. For example: (i) economists can experiment with incentives for participation, (ii) communication researchers can experiment with techniques to explain the societal goals, and (iii) political scientists can study the reactions of various stakeholders to the provided data while making infrastructure decisions.

Using the platform in other domains At its most general, the platform collects a combination of sensed location and surveyed data, linked to one individual. This is most useful for assessing transportation patterns, but it can also be used in health to understand the fitness profiles of individuals. It is particularly useful for capturing physical activity that may not be perceived as exercise, such as walking to run errands.

Lessons for other platform builders The sensed data at the heart of this platform is location. Future platform builders may need to focus on other privacy-sensitive sensors such as video or audio. Some lessons from e-mission that they may want to incorporate are:

(i) *provide user functionality*: even if initial deployers focus only on data collection, supporting the ability to provide user functionality diversifies the deployer community significantly,

(ii) *lower barriers for end-users*: don't require the use of two apps, don't require reading installation instructions, don't require in-personal installs,

(iii) *support mass customization*: every deployer will have their own opinion about features, notably the UI, and the best way to short-circuit endless arguments is to allow them to customize according to their ideas, and

(iv) *make the analysis reproducible*: this will allow graceful recovery from analysis algorithm mistakes when (not if) they occur.

We hope that this work inspires other researchers to tackle interdisciplinary work, collaborate in solving real-world problems, and save the world together while we still can.

Bibliography

[BCH$^+$11] Richard A. Becker, Ramón Cáceres, Karrie Hanson, Ji Meng Loh, Simon Urbanek, Alexander Varshavsky, and Chris Volinsky. A tale of one city: Using cellular network data for urban planning. *Pervasive Computing, IEEE*, 10(4):18–26, 2011.

[BCN13] Michel Bierlaire, Jingmin Chen, and Jeffrey Newman. A probabilistic map matching method for smartphone GPS data. *Transportation Research Part C: Emerging Technologies*, 26:78–98, January 2013.

[BI04] Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. In *Pervasive Computing*, pages 1–17. Springer, 2004.

[BK11] Ulrich Bareth and Axel Kupper. Energy-Efficient Position Tracking in Proactive Location-Based Services for Smartphone Environments. pages 516–521. IEEE, July 2011.

[BPS$^+$14] Efthimios Bothos, Sebastian Prost, Johann Schrammel, Kathrin Röderer, and Gregoris Mentzas. Watch your Emissions: Persuasive Strategies and Choice Architecture for Sustainable Decisions in Urban Mobility. *PsychNology Journal*, 12(3):107–126, 2014.

[BSD$^+$13] Waylon Brunette, Mitchell Sundt, Nicola Dell, Rohit Chaudhri, Nathan Breit, and Gaetano Borriello. Open data kit 2.0: Expanding and refining information services for developing regions. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 10. ACM, 2013.

[Cas15] Joe Castiglione. *Activity-Based Travel Demand Models: A Primer*. Transportation Research Board, Washington, DC, 2015.

[CCC$^+$13] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, and Dario Maio. MSF: An Efficient Mobile Phone Sensing Framework. *International Journal of Distributed Sensor Networks*, 9(3):538937, March 2013.

[CCC$^+$14a] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, Raffaele Ianniello, and Rebecca Montanari. Crowdsensing in Urban Areas for City-Scale Mass Gathering Management: Geofencing and Activity Recognition. *IEEE Sensors Journal*, 14(12):4185–4195, December 2014.

[CCC$^+$14b] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, and Rebecca Montanari. Activity recognition for Smart City scenarios: Google Play Services vs. MoST facilities. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, June 2014.

[CCCF14] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, and Luca Foschini. The participact mobile crowd sensing living lab: The testbed for smart cities. *IEEE Communications Magazine*, 52(10):78–85, October 2014.

[CCH10] Catherine T. Lawson, Cynthia Chen, and Hongmian Gong. Advanced Applications of Person-based GPS in an Urban Environment. Technical Report 49111-14-21, University at Albany, Albany, New York, February 2010.

[CFF$^+$13] Caitlin D. Cottrill, Francisco Camara Pereira, Fang Zhao, Ines Ferreira Dias, Hock Beng Lim, Moshe Ben-Akiva, and P. Christopher Zegras. The Future Mobility Survey: Experiences in developing a smartphone-based travel survey in Singapore. *Transportation Research Record: Journal of the Transportation Research Board*, (2354):59–67, 2013.

[CK16] Judd Cramer and Alan B. Krueger. Disruptive Change in the Taxi Business: The Case of Uber. *American Economic Review*, 106(5):177–182, May 2016.

[CLB$^+$18] Claudia Carpineti, Vincenzo Lomonaco, Luca Bedogni, Marco Di Felice, and Luciano Bononi. Custom Dual Transportation Mode Detection by Smartphone Devices Exploiting Sensor Diversity. *arXiv:1810.05596 [cs, stat]*, October 2018.

[CLL$^+$11] David Chu, Nicholas D. Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. Balancing energy, latency and accuracy for mobile sensor data classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 54–67. ACM, 2011.

[CMG$^+$17] Mathias Ciliberto, Francisco Javier Ordoñez Morales, Hristijan Gjoreski, Daniel Roggen, Sami Mekki, and Stefan Valentin. High reliability Android application for multidevice multimodal mobile data acquisition and annotation. In *Proceedings of the 15th ACM Conference on Embedded*

Network Sensor Systems - SenSys '17, pages 1–2, Delft, Netherlands, 2017. ACM Press.

[Com] Taxi & Limousine Commission. Trip Record Data New York City.

[Com12] *Computing Research for Sustainability.* The National Academies Press, Washington, DC, 2012.

[CSW16] Andre Carrel, Raja Sengupta, and Joan L. Walker. The San Francisco Travel Quality Study: Tracking trials and tribulations of a transit taker. *Transportation*, pages 1–37, 2016.

[dHVB13] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. Unique in the Crowd: The privacy bounds of human mobility. *Scientific Reports*, 3, March 2013.

[DKT$^+$] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. BOSS: Building Operating System Services. page 15.

[ESP06] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, May 2006.

[FDK11] Denzil Ferreira, Anind K. Dey, and Vassilis Kostakos. Understanding human-smartphone concerns: A study of battery life. In *Pervasive Computing*, pages 19–33. Springer, 2011.

[FDR$^+$08] Aimee Flannery, Richard G. Dowling, Nagui M. Rouphail, Theodore Anton Petritsch, Bruce W. Landis, James A. Bonneson, Paul Ryus, David B. Reinke, Mark Vandehey, Transportation Research Board, National Cooperative Highway Research Program, and Transportation Research Board. *Multimodal Level of Service Analysis for Urban Streets.* National Academies Press, Washington, D.C., September 2008.

[FF13] FHWA and Federal Highway Administration. Highway Performance Monitoring System, March 2013.

[FH17] Emma G. Fitzsimmons and Winnie Hu. The Downside of Ride-Hailing: More New York City Gridlock. *The New York Times*, March 2017.

[FLF$^+$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, August 2016.

[FLHG17] Leah Flake, Michelle Lee, Kevin Hathaway, and Elizabeth Greene. Use of Smartphone Panels for Viable and Cost-Effective GPS Data Collection for Small and Medium Planning Agencies. *Transportation Research Record: Journal of the Transportation Research Board*, 2643:160–165, January 2017.

[FPV$^+$13] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva. Visual Exploration of Big Spatio-Temporal Urban Data: A Study of New York City Taxi Trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, December 2013.

[FT13] Tao Feng and Harry J.P. Timmermans. Transportation mode recognition using GPS and accelerometer data. *Transportation Research Part C: Emerging Technologies*, 37:118–130, December 2013.

[FZP15] Fei Yang, Zhenxing Yao, and Peter Jin. Multi-mode Trip Information Recognition Based on Wavelet Transform Modulus Algorithm by Using GPS and Acceleration Data. In *TRB 94th Annual Meeting Compendium of Papers*, Washington, DC, January 2015.

[GCM$^+$17] Hristijan Gjoreski, Mathias Ciliberto, Francisco Javier Ordoñez Morales, Daniel Roggen, Sami Mekki, and Stefan Valentin. A Versatile Annotated Dataset for Multimodal Locomotion Analytics with Mobile Devices. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems - SenSys '17*, pages 1–2, Delft, Netherlands, 2017. ACM Press.

[GFF$^+$19] Carla Gomes, Xiaoli Fern, Daniel Fink, Douglas Fisher, Alexander Flecker, Daniel Freund, Angela Fuller, John Gregoire, John Hopcroft, Steve Kelling, Zico Kolter, Thomas Dietterich, Warren Powell, Nicole Sintov, John Selker, Bart Selman, Daniel Sheldon, David Shmoys, Milind Tambe, Weng-Keen Wong, Christopher Wood, Xiaojian Wu, Christopher Barrett, Yexiang Xue, Amulya Yadav, Abdul-Aziz Yakubu, Mary Lou Zeeman, Jon Conrad, Bistra Dilkina, Stefano Ermon, Fei Fang, Andrew Farnsworth, and Alan Fern. Computational sustainability: Computing for a better world and a sustainable future. *Communications of the ACM*, 62(9):56–65, August 2019.

[GFHG16] Elizabeth Greene, Leah Flake, Kevin Hathaway, and Michael Geilich. A Seven-day smartphone-based GPS household travel survey in Indiana. In *9th Annual Meeting of the Transportation Research Board*, Washington, D.C, January 2016. Transportation Research Board.

[GR16] Gregory Powell and Rebecca Yoon. 2016-31059.pdf. Notice of Proposed Rulemaking (NPRM) NHTSA–2016–0126, National Highway Traffic

Safety Administration (NHTSA), Department of Transportation (DOT), 2016.

[GWB$^+$10] P.A. Gonzalez, J.S. Weinstein, S.J. Barbeau, M.A. Labrador, P.L. Winters, N.L. Georggi, and R. Perez. Automating mode detection for travel behaviour analysis by using global positioning systems-enabled mobile phones and neural networks. *IET Intelligent Transport Systems*, 4(1):37, 2010.

[Har] Chris Harding. From Smartphone Apps to In-Person Data Collection: Modern and Cost-Effective Multimodal Travel Data Collection for Evidence-Based Planning. page 554.

[HCCL13] Xiping Hu, Terry H. S. Chu, Henry C. B. Chan, and Victor C. M. Leung. Vita: A Crowdsensing-Oriented Mobile Cyber-Physical System. *IEEE Transactions on Emerging Topics in Computing*, 1(1):148–165, June 2013.

[HHE13] Sungsoon Hwang, Timothy Hanke, and Christian Evans. Automated Extraction of Community Mobility Measures from GPS Stream Data Using Temporal DBSCAN. In *Computational Science and Its Applications–ICCSA 2013*, pages 86–98. Springer, 2013.

[HLA$^+$10] Carl Hartung, Adam Lerer, Yaw Anokwa, Clint Tseng, Waylon Brunette, and Gaetano Borriello. Open data kit: Tools to build information services for developing regions. In *Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development - ICTD '10*, pages 1–12, London, United Kingdom, 2010. ACM Press.

[HLD$^+$13] Xiping Hu, Victor C.M. Leung, Weichang Du, Boon-Chong Seet, and Panos Nasiopoulos. A Service-Oriented Mobile Social Networking Platform for Disaster Situations. In *2013 46th Hawaii International Conference on System Sciences*, pages 136–145, Wailea, HI, USA, January 2013. IEEE.

[HLZ$^+$16] Hongik University, Jae Seung Lee, P. Christopher Zegras, Fang Zhao, Daehee Kim, and Junhee Kang. Testing the Reliability of a Smartphone-Based Travel Survey: An Experiment in Seoul. *The Journal of The Korea Institute of Intelligent Transport Systems*, 15(2):50–62, April 2016.

[HNT13] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM Press, 2013.

[HRK$^+$10] John Hicks, Nithya Ramanathan, Donnie Kim, Mohamad Monibi, Joshua Selsky, Mark Hansen, and Deborah Estrin. AndWellness: An open mobile system for activity and experience sampling. In *Wireless Health 2010 on - WH '10*, page 34, San Diego, California, 2010. ACM Press.

[HS] N B Hounsell and B P Shrestha. AVL based Bus Priority at Traffic Signals: A Review and Case Study of Architectures. *European Journal of Transport and Infrastructure Research*, page 17.

[HSC11] Jeffrey Hood, Elizabeth Sall, and Billy Charlton. A GPS-based bicycle route choice model for San Francisco, California. *Transportation Letters: The International Journal of Transportation Research*, 3(1):63–75, January 2011.

[HSE$^+$13] Cheng-Kang Hsieh, Dallas Swendeman, Deborah Estrin, Nithya Ramanathan, Hongsuda Tangmunarunkit, Faisal Alquaddoomi, John Jenkins, Jinha Kang, Cameron Ketcham, Brent Longstaff, Joshua Selsky, and Betta Dawson. Lifestreams: A modular sense-making toolset for identifying important patterns from everyday life. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys '13*, pages 1–13, Roma, Italy, 2013. ACM Press.

[HSHM] Chris Harding, Siva Srikukenthiran, Khandker Nurul Habib, and Eric J Miller. On the User Experience and Performance of Smartphone Apps as Personalized Travel Survey Instruments: Results from an Experiment in Toronto. page 2.

[HWH$^+$10] Juan C. Herrera, Daniel B. Work, Ryan Herring, Xuegang (Jeff) Ban, Quinn Jacobson, and Alexandre M. Bayen. Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment. *Transportation Research Part C: Emerging Technologies*, 18(4):568–583, August 2010.

[HYL$^+$14] Thomas Holleczek, Liang Yu, Joseph Kang Lee, Oliver Senn, Carlo Ratti, and Patrick Jaillet. Detecting weak public transport connections from cellphone and public transport data. pages 1–8. ACM Press, 2014.

[HYT$^+$10] J. Hu, F. Yan, J. Tian, P. Wang, and K. Cao. Developing PC-Based Automobile Diagnostic System Based on OBD System. In *2010 Asia-Pacific Power and Energy Engineering Conference*, pages 1–5, March 2010.

[IHG12] Darrel C. Ince, Leslie Hatton, and John Graham-Cumming. The case for open computer programs. *Nature*, 482(7386):485–488, February 2012.

[JAC+15] Jerald Jariyasunant, Maya Abou-Zeid, Andre Carrel, Venkatesan Ekambaram, David Gaker, Raja Sengupta, and Joan L. Walker. Quantified Traveler: Travel Feedback Meets the Cloud to Change Behavior. *Journal of Intelligent Transportation Systems*, 19(2):109–124, April 2015.

[JBR+16] Kasthuri Jayarajah, Rajesh Krishna Balan, Meera Radhakrishnan, Archan Misra, and Youngki Lee. LiveLabs: Building In-Situ Mobile Sensing & Behavioural Experimentation TestBeds. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '16*, pages 1–15, Singapore, Singapore, 2016. ACM Press.

[JMA+13] Jerald Jariyasunant, Maya Abou-Zeid, Andre Carrel, Venkatesan Ekambaram, David Gaker, and Raja Sengupta. Quantified Traveler: Travel Feedback Meets the Cloud to Change Behavior. Technical Report UCTC-FR-2013-06, University of California Transportation Center, September 2013.

[JMM08] James A. Bonneson, Michael P. Pratt, and Mark A. Vandehey. Predicting the performance of automobile traffic on urban streets. Technical Report 3-79, Transportation Research Board, January 2008.

[JNS+13] Antti Jylhä, Petteri Nurmi, Miika Sirén, Samuli Hemminki, and Giulio Jacucci. MatkaHupi: A persuasive mobile application for sustainable mobility. pages 227–230. ACM Press, 2013.

[Jor18] Michael Jordan. Artificial Intelligence — The Revolution Hasn't Happened Yet, April 2018.

[KAB+12] Tobias Kuhnimhof, Jimmy Armoogum, Ralph Buehler, Joyce Dargay, Jon Martin Denstadli, and Toshiyuki Yamamoto. Men Shape a Downward Trend in Car Use among Young Adults—Evidence from Six Industrialized Countries. *Transport Reviews*, 32(6):761–779, November 2012.

[KB08] N.K. Kanhere and S.T. Birchfield. Real-Time Incremental Segmentation and Tracking of Vehicles at Low Camera Angles Using Stable Features. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):148–160, March 2008.

[KJP15] Alexandr Krylovskiy, Marco Jahn, and Edoardo Patti. Designing a Smart City Internet of Things Platform with Microservice Architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30, Rome, Italy, August 2015. IEEE.

[KLGT09] Mikkel Baun Kjaergaard, Jakob Langdal, Torben Godsk, and Thomas Toftkjaer. Entracked: Energy-efficient robust position tracking for mobile devices. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, pages 221–234. ACM, 2009.

[KMP13] Sunyoung Kim, Jennifer Mankoff, and Eric Paulos. Sensr: Evaluating a flexible framework for authoring mobile data-collection tools for citizen science. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, pages 1453–1462. ACM, 2013.

[LCSC09] Jyong Lin, Shih-Chang Chen, Yu-Tsen Shih, and Shi-Huang Chen. A Study on Remote On-Line Diagnostic System for Vehicles by Integrating the Technology of OBD, GPS, and 3G. 3(8):7, 2009.

[LGA$^+$12] Juha K. Laurila, Daniel Gatica-Perez, Imad Aad, Olivier Bornet, Trinh-Minh-Tri Do, Olivier Dousse, Julien Eberle, Markus Miettinen, et al. The mobile data challenge: Big data for mobile computing research. In *Pervasive Computing*, 2012.

[LLYK16] W. L. H. Lim, J. T. W. Lum, I. J. W. Yeo, and S. L. Keoh. A crowd-assisted real-time public transport information service: No more endless wait. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6, March 2016.

[LML$^+$10] Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[LV16] Oana Lorintiu and Andrea Vassilev. Transportation mode recognition based on smartphone embedded sensors for carbon footprint estimation. pages 1976–1981. IEEE, November 2016.

[LYL$^+$10] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 71–84, 2010.

[LZ99] Wei-Hua Lin and Jian Zeng. Experimental study of real-time bus arrival time prediction with GPS data. *Transportation Research Record: Journal of the Transportation Research Board*, (1666):101–109, 1999.

[MCCM15] J. E. Meseguer, C. T. Calafate, J. C. Cano, and P. Manzoni. Assessing the impact of driving behavior on instantaneous fuel consumption.

In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 443–448, January 2015.

[MCF15] Marcelo Martins, Justin Cappos, and Rodrigo Fonseca. Selectively taming background android apps to improve battery lifetime. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, pages 563–575. USENIX Association, 2015.

[McN] Michael G. McNally. The Four-Step Model. In *Handbook of Transport Modelling*, pages 35–53.

[MGS10] E McCallister, T Grance, and K A Scarfone. Guide to protecting the confidentiality of Personally Identifiable Information (PII). Technical Report NIST SP 800-122, National Institute of Standards and Technology, Gaithersburg, MD, 2010.

[MM16] Caleb Ronald Munigety and Tom V. Mathew. Towards Behavioral Modeling of Drivers in Mixed Traffic Conditions. *Transportation in Developing Economies*, 2(1), April 2016.

[MMN+17] R. Malekian, N. R. Moloisane, L. Nair, B. T. Maharaj, and U. A. K. Chude-Okonkwo. Design and Implementation of a Wireless OBD II Fleet Management System. *IEEE Sensors Journal*, 17(4):1154–1164, February 2017.

[MR] Michael G. McNally and Craig R. Rindt. The Activity-Based Approach. In *Handbook of Transport Modelling*, pages 55–73.

[Nat12] Suman Nath. ACE: Exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pages 29–42. ACM, 2012.

[NK08] Nadine Schüssler and Kay Axhausen. Processing GPS raw data without additional information. *Transportation Research Record: Journal of the Transportation Research Board*, 2008.

[OnB] On-Board Diagnostic II (OBD II) Systems Fact Sheet — California Air Resources Board.

[OVW+11] Marcelo Oliveira, Peter Vovsha, Jean Wolf, Yehoshua Birotker, Danny Givon, and Julie Paasche. Global Positioning System-Assisted Prompted Recall Household Travel Survey to Support Development of Advanced Travel Model in Jerusalem, Israel. *Transportation Research Record: Journal of the Transportation Research Board*, 2246:16–23, December 2011.

[OW49] O.K. Normann and W.P. Walker. Highway capacity manual: Practical applications of research. Technical report, U.S. Department of Commerce, Bureau of Public Roads, Washington, 1949. Open Library ID: OL6085220M.

[PF16] Zachary Patterson and Kyle Fitzsimmon. DATAMOBILE: A SMARTPHONE TRAVEL SURVEY EXPERIMENT. *Transportation Research Record: Journal of the Transportation Research Board*, 2594(1):35–43, January 2016.

[PFJM19] Zachary Patterson, Kyle Fitzsimmons, Stewart Jackson, and Takeshi Mukai. Itinerum: The open smartphone travel survey platform. *SoftwareX*, 10:100230, July 2019.

[PGS17] Adrian C. Prelipcean, Gyözö Gidófalvi, and Yusak O. Susilo. Transportation mode detection – an in-depth review of applicability and reliability. *Transport Reviews*, 37(4):442–464, July 2017.

[PKG10] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 299–314. ACM, 2010.

[PLMM15] Veljko Pejovic, Neal Lathia, Cecilia Mascolo, and Mirco Musolesi. Mobile-Based Experience Sampling for Behaviour Research. *arXiv preprint arXiv:1508.03725*, 2015.

[Plu05] A. Plummer. The Chicago Area Transportation Study. *Creating the first plan (1955–1962). A narrative. Rapport pour le Chicago Area Transportation Study, Chicago*, 35, 2005.

[Poz13] Alex Poznanski. *Analysing Demographic and Geographic Characteristics of "Cycle Atlanta" Smartphone Application Users.* Master's, Georgia Institute of Technology, May 2013.

[PSCM18] Francesco Piras, Eleonora Sottile, Daniele Calli, and Italo Meloni. Automatic data collection for detecting travel behavior: The IPET platform. *Procedia Computer Science*, 134:421–426, 2018.

[RMB$^+$10] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks*, 6(2):1–27, February 2010.

[Rob70] Robert L. Anderson. Electromagnetic loop vehicle detectors. *IEEE Transactions on Vehicular Technology*, 19(1):23–30, 1970.

[RQZ07] Ahmad Rahmati, Angela Qian, and Lin Zhong. Understanding human-battery interaction on mobile phones. In *Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 265–272. ACM, 2007.

[RSD$^+$10] Sasank Reddy, Katie Shilton, Gleb Denisov, Christian Cenizal, Deborah Estrin, and Mani Srivastava. Biketastic: Sensing and mapping for better biking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1817–1820. ACM, 2010.

[SAM$^+$15] Hamid Safi, Behrang Assemi, Mahmoud Mesbah, Luis Ferreira, and Mark Hickman. Design and Implementation of a Smartphone-Based Travel Survey. *Transportation Research Record: Journal of the Transportation Research Board*, 2526:99–107, January 2015.

[SAMF16a] Hamid Safi, Behrang Assemi, Mahmoud Mesbah, and Luis Ferreira. A trip-detection method for smartphone-assisted travel data collection. In *Transportation Research Board (TRB) 95th Annual Meeting*, 2016.

[SAMF16b] Hamid Safi, Behrang Assemi, Mahmoud Mesbah, and Luis Ferreira. A trip-detection method for smartphone-assisted travel data collection. In *Transportation Research Board (TRB) 95th Annual Meeting*, 2016.

[SBM$^+$18] K. Shankari, Mohamed Amine Bouzaghrane, Samuel M. Maurer, Paul Waddell, David E. Culler, and Randy H. Katz. E-mission: An Open-Source, Smartphone Platform for Collecting Human Travel Data. *Transportation Research Record: Journal of the Transportation Research Board*, 2672(42):1–12, December 2018.

[Sch17] Bruce Schaller. Unsustainable? The Growth of App-Based Ride Services and Traffic, Travel and the Future of New York City. Technical report, New York, NY, February 2017.

[Seg96] Jakub Segen. A camera-based system for tracking people in real time. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference On*, volume 3, pages 63–67. IEEE, 1996.

[SFC$^+$] K Shankari, Jonathan Furst, David E Culler, Yawen Wang, Philippe Bonnet, and Randy H Katz. Zephyr: Simple, Ready-to-use Software-based Power Evaluation for Background Sensing Smartphone Applications. page 21.

[SJF05] Peter R. Stopher, Qingjian Jiang, and Camden FitzGerald. Processing GPS data from travel surveys. *2nd international colloqium on the behavioural foundations of integrated land-use and transportation models: frameworks, models and applications, Toronto*, 2005.

[SMP09] Guruprasad Somasundaram, Vassilios Morellas, and Nikolaos Papanikolopoulos. Counting pedestrians and bicycles in traffic scenes. In *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference On*, pages 1–6. IEEE, 2009.

[SMS13] Akihiko Sakata, Yukimasa Matsumoto, and Hidekazu Suzuki. Development of Bus Location System with Smartphone and Effect of Providing Regional Information added on Bus Information. page 12, 2013.

[SP12] Vijay Srinivasan and Thomas Phan. An accurate two-tier classifier for efficient duty-cycling of smartphone activity recognition systems. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, page 11, 2012.

[SPM+15] Johann Schrammel, Sebastian Prost, Elke Mattheiss, Efthimios Bothos, and Manfred Tscheligi. Using Individual and Collaborative Challenges in Behavior Change Support Systems: Findings from a Two-Month Field Trial of a Trip Planner Application. In Thomas MacTavish and Santosh Basapur, editors, *Persuasive Technology*, volume 9072, pages 160–171. Springer International Publishing, Cham, 2015.

[SS13] Brandon Schoettle and Michael Sivak. The reasons for the recent decline in young driver licensing in the US. 2013.

[SSLA15] Peter R. Stopher, Li Shen, Wen Liu, and Asif Ahmed. The Challenge of Obtaining Ground Truth for GPS Processing. *Transportation Research Procedia*, 11:206–217, 2015.

[Sul19] John Sullivan. *An Approach to Privacy Preserving for Spatio-Temporal Data.* Master's, EECS Department, University of California, Berkeley, May 2019.

[SWLN14] Rahul C. Shah, Chieh-yih Wan, Hong Lu, and Lama Nachman. Classifying the mode of transportation on mobile phones using GIS information. pages 225–229. ACM Press, 2014.

[SYCK15] Kalyanaraman Shankari, Mogeng Yin, David Culler, and Randy H Katz. E-Mission: Automated transportation emission calculation using smartphones. In *Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 268–271, March 2015.

[TKK+15] H. Tangmunarunkit, J. Kang, Z. Khalapyan, J. Ooms, N. Ramanathan, D. Estrin, C. K. Hsieh, B. Longstaff, S. Nolen, J. Jenkins, C. Ketcham, J. Selsky, F. Alquaddoomi, and D. George. Ohmage: A General and Extensible End-to-End Participatory Sensing Platform. *ACM Transactions on Intelligent Systems and Technology*, 6(3):1–21, April 2015.

[TMRR12] Sha Tao, Vasileios Manolopoulos, Saul Rodriguez, and Ana Rusu. Real-Time Urban Traffic State Estimation with A-GPS Mobile Phones as Probes. *Journal of Transportation Technologies*, 02(01):22–31, 2012.

[TS72] Sheung Yuen Amy Tsui and Amer S Shalaby. Enhanced System for Link and Mode Identification for Personal Travel Surveys Based on Global Positioning Systems. *Transportation Research Record*, page 8, 1972.

[UE] Climate Change Division US EPA. U.S. Greenhouse Gas Inventory Report. The national greenhouse gas inventory is developed each year to track trends in U.S. emissions and removals. Find emissions by source, economic sector and greenhouse gas.

[Vic17] Victoria Transport Policy Institute. Online TDM Encyclopedia - Multi-Modal Level-Of-Service. *TDM Encyclopedia*, April 2017.

[VS15] Akshay Vij and K. Shankari. When is big data big enough? Implications of using GPS-based surveys for travel demand analysis. *Transportation Research Part C: Emerging Technologies*, 56:446–462, July 2015.

[WBO$^+$14] Jean Wolf, William Bachman, Marcelo Simas Oliveira, Joshua Auld, Abolfazl (Kouros) Mohammadian, Peter Vovsha, National Cooperative Highway Research Program, Transportation Research Board, and National Academies of Sciences, Engineering, and Medicine. *Applying GPS Data to Understand Travel Behavior, Volume I: Background, Methods, and Tests*, volume 1. Transportation Research Board, Washington, D.C., June 2014.

[WGB01] Jean Wolf, Randall Guensler, and William Bachman. Elimination of the Travel Diary: Experiment to Derive Trip Purpose from Global Positioning System Travel Data. *Transportation Research Record: Journal of the Transportation Research Board*, 1768(1):125–134, January 2001.

[WHB$^+$12] Pu Wang, Timothy Hunter, Alexandre M. Bayen, Katja Schechtner, and Marta C. González. Understanding Road Usage Patterns in Urban Areas. *Scientific Reports*, 2, December 2012.

[Wol14] Jean Wolf. *Applying GPS Data to Understand Travel Behavior*. Number 775 in Nchrp National Cooperative Highway Research Program Report. Transportation Research Board of the National Academies, Washington, DC, 2014.

[WSWG] Stephan Winter, Monika Sester, Ouri Wolfson, and Glenn Geers. Towards a Computational Transportation Science. page 10.

[YCP$^+$13] Zhixian Yan, Dipanjan Chakraborty, Christine Parent, Stefano Spaccapietra, and Karl Aberer. Semantic trajectories: Mobility data computation and annotation. *ACM Transactions on Intelligent Systems and Technology*, 4(3):1, June 2013.

[YGW13] David J. Yates, Girish J. Jeff Gulati, and Joseph W. Weiss. Understanding the Impact of Policy, Regulation and Governance on Mobile Broadband Diffusion. In *2013 46th Hawaii International Conference on System Sciences*, pages 2852–2861, Wailea, HI, USA, January 2013. IEEE.

[YSC$^+$12] Zhixian Yan, Vigneshwaran Subbaraju, Dipanjan Chakraborty, Archan Misra, and Karl Aberer. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In *Wearable Computers (ISWC), 2012 16th International Symposium On*, pages 17–24, 2012.

[YYW$^+$14] 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, August 2014.

[Zan09] Paul A Zandbergen. Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning. *Transactions in GIS*, 13:5–25, June 2009.

[ZB] Hui Zang and Jean Bolot. Anonymization of Location Data Does Not Work: A Large-Scale Measurement Study. page 12.

[ZB11] Paul A. Zandbergen and Sean J. Barbeau. Positional Accuracy of Assisted GPS Data from High-Sensitivity GPS-enabled Mobile Phones. *Journal of Navigation*, 64(3):381–399, July 2011.

[ZCCM11] J. Zaldivar, C. T. Calafate, J. C. Cano, and P. Manzoni. Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones. In *2011 IEEE 36th Conference on Local Computer Networks*, pages 813–819, October 2011.

[ZCL$^+$10] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding transportation modes based on GPS data for web applications. *ACM Transactions on the Web*, 4(1):1–36, January 2010.

[ZGP$^+$15] Fang Zhao, Ajinkya Ghorpade, Francisco Câmara Pereira, Christopher Zegras, and Moshe Ben-Akiva. Stop Detection in Smartphone-based Travel Surveys. *Transportation Research Procedia*, 11:218–226, 2015.

[ZHUK13a] Xianyuan Zhan, Samiul Hasan, Satish V. Ukkusuri, and Camille Kamga. Urban link travel time estimation using large-scale taxi data with partial information. *Transportation Research Part C: Emerging Technologies*, 33:37–49, August 2013.

[ZHUK13b] Xianyuan Zhan, Samiul Hasan, Satish V. Ukkusuri, and Camille Kamga. Urban link travel time estimation using large-scale taxi data with partial information. *Transportation Research Part C: Emerging Technologies*, 33:37–49, August 2013.

[ZSC13] Mohamed Zaki, Tarek Sayed, and Andrew Cheung. Computer Vision Techniques for the Automated Collection of Cyclist Data. *Transportation Research Record: Journal of the Transportation Research Board*, 2387:10–19, December 2013.

[ZTG^{+}11] John Zimmerman, Anthony Tomasic, Charles Garrod, Daisy Yoo, Chaya Hiruncharoenvate, Rafae Aziz, Nikhil Ravi Thiruvengadam, Yun Huang, and Aaron Steinfeld. Field trial of Tiramisu: Crowd-sourcing bus arrival times to spur co-design. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11*, page 1677, Vancouver, BC, Canada, 2011. ACM Press.

[ZWHI15] Mingyang Zhong, Jiahui Wen, Peizhao Hu, and Jadwiga Indulska. Advancing Android activity recognition service with Markov smoother. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference On*, pages 38–43. IEEE, 2015.

[ZXM10] Yu Zheng, Xing Xie, and Wei-Ying Ma. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.