

Mobile Location Data Analytics, Privacy, and Security

A Dissertation Presented for the
Doctor of Philosophy
Degree

The University of Tennessee, Knoxville

Yunhe Feng

August 2020

Copyright © by Yunhe Feng, 2020
All Rights Reserved.

Dedication

To my parents, Yanshun Feng and Mingxia Li, and my uncle, Chunlei Li.

Acknowledgments

I would like to thank my advisor, Dr. Qing Cao, who guided me through six years of research at UT. I appreciate the unconditional support and professional guidance he gave to me from the first day when I joined UT. His principle of thinking a research problem from comprehensive and thorough aspects inspired me to think more before taking action. When I got stuck in research, his exceptional patience and encouragement always motivated me to move forward. Dr. Cao also provided me with an open and flexible research environment, enabling me to find and follow my research interests.

I would like to thank my committee members, Dr. Scott Ruoti, Dr. Jinyuan Sun, and Dr. Wenjun Zhou, for their support and time. I appreciate the invaluable guidance and insightful comments they gave to me on my research and studies. Also, I learned a lot in paper writing and time management when collaborating with them. I would like to thank all professors and staff at UT for their inspiring and amazing assistance. I would also like to thank my mentors at Oak Ridge National Laboratory, Dr. Dali Wang, and Dr. OluFemi A. Omitaomu, to offer me great interdisciplinary research opportunities.

I would like to thank my lab mate, Dr. Zheng Lu, for being such a wonderful colleague and a funny friend in life. I would like to thank my lab mate, Dr. Sisi Xiong. Her informative suggestions for my research and practical cooking tips benefit me a lot. I am grateful to meet all group members from Dr. Ruoti's USER Lab and Dr. Zhou's Zhou Lab to exchange research ideas and discuss research questions. I also would like to thank Dr. Xi Luo, Qinglei Cao, and Dong Zhong from Dr. Dongarra's ICL Lab for our friendships.

Last but not least, I would like to thank my family and my friends. Although I cannot list all of their names, I appreciate their accompaniment and assistance during my Ph.D. study. I would not be able to achieve anything without their love and support.

Abstract

Mobile location data are ubiquitous in the digital world. People intentionally and unintentionally generate numerous location data when connecting to cellular networks or sharing posts on social networks. As mobile devices normally choose to communicate with nearby cell towers outdoor, it is reasonable to infer human locations based on cell tower coordinates. Many social networking platforms, such as Twitter, allow users to geo-tag their posts optionally, publishing personal locations to friends or everyone. These location data are particularly useful for understanding mobile usage behaviors and human mobility patterns. Meanwhile, the public expresses great concern about the privacy and security of their location information. Secure sharing of locations and mitigating malicious location queries from bots (especially on mobile devices) become increasingly imperative and necessary.

For mobile location analytics, we first study the cellular traffic data generated during the communication between mobile apps and nearby cell towers. We propose a multi-level mixture of kernel density estimation (mlKDE) model to profile the geospatial distribution of given individual apps, and demonstrate that mlKDE could effectively and robustly characterize the distributions of app usage in the real world. Then we investigate the location information shared publicly by social media users. Several Twitter case studies are conducted to illustrate how social location data inferred human mobility as well as geospatial distribution patterns.

To enhance the security of private locations, we propose homomorphic bloom filters to enable one party to determine, in a private and secure manner, whether or not the trajectory of a second party has an intersection with specific locations of interest. As location-based services are mainly operated on mobile devices, location queries are frequently launched by mobile users. To distinguish humans and bots, we design and implement SenCAPTCHA, a

mobile-first CAPTCHA using orientation sensors. SenCAPTCHA works by showing users an animal image and asking them to tilt their devices to guide a red ball into the center of that animal’s eye. Two usability studies show that SenCAPTCHA is an “enjoyable” CAPTCHA and it is preferred by the majority of participants to other existing CAPTCHA systems.

Table of Contents

1	Introduction	1
1.1	Spatial Mobile APP Usage in Cellular Networks	2
1.2	Publicly Shared Location Data on Social Networks	4
1.3	Online Secure Sharing of Private Locations	5
1.4	Blocking Malicious Bot Attacks on Locations	8
2	Related Work	10
2.1	Spatial Modeling of Mobile App Usages	10
2.2	Geo-tagged Social Media Data Analytics	11
2.3	Secure Sharing of Private Locations	12
2.4	Authentication Solutions to Battle Location Bots	14
3	A Multi-Granularity Perspective for Spatial Profiling of Mobile Apps	17
3.1	Problem Formulation and Data Description	17
3.1.1	Problem Formulation	18
3.1.2	Cell Tower Location	20
3.1.3	Apps Information	20
3.2	Modeling Individual Apps' Spatial Distributions	22
3.2.1	Kernel Density Estimation	22
3.2.2	The Multi-level Mixture of Kernel Density Estimation	24
3.2.3	Algorithm Implementation	25
3.3	Experimental Results	28
3.3.1	Datasets and Experimental Setup	28

3.3.2	What Does mlKDE Visualization Look Like?	28
3.3.3	How to Interpret Mixing Weights?	33
3.3.4	How Much Does mlKDE Perform Better than KDE?	36
3.3.5	Are Mixing Weights Sensitive to Bandwidths?	38
4	Investigating Shared Locations On Social Networks	40
4.1	Twitter Location Types	41
4.1.1	Twitter User Profile Location	41
4.1.2	Tweet Location	41
4.2	Human Mobility During the 2017 Great American Eclipse	42
4.2.1	Background and Tweet Collection	42
4.2.2	Geographic Distribution Overview	42
4.2.3	Interstate and Intrastate Human Movement	44
4.3	Geographic Patterns of Petitions for New Emojis on Twitter	48
4.3.1	Background and Tweet Collection	48
4.3.2	Geographic Patterns	49
4.3.3	Place-Related Emoji Petitions	51
4.4	Geographic Analyses on Shared Dockless Electric Scooter Deployment	51
4.4.1	Background and Tweet Collection	51
4.4.2	Geo-enabled Temporal Distribution	52
4.4.3	E-Scooter Deployment in the United States	53
4.5	A Fine-Grained View of COVID-19 via Geo-tagged Tweets	55
4.5.1	Background and Tweet Collection	55
4.5.2	Temporal and Geographic Distributions	56
4.5.3	Potential Privacy Leakage	62
5	Secure Sharing of Private Locations through Homomorphic Bloom Filters	64
5.1	Background and Framework Overview	65
5.1.1	Assumptions	65
5.1.2	Background	65
5.1.3	Main Ideas on an Ideal Protocol	67

5.2	Optimized Protocols	69
5.2.1	Optimization 1: A Lightweight Homomorphic Bloom Filter	69
5.2.2	Optimization 2: An Improved Homomorphic Bloom Filter	69
5.2.3	Optimization 3: A Bit-wise Homomorphic Bloom Filter	72
5.3	Security Analysis	74
5.3.1	Alice’s Privacy and Actions	75
5.3.2	Bob’s Privacy and Actions	75
5.3.3	The Aggregation Server’s Actions	75
5.4	Evaluation	76
5.4.1	Datasets and Experimental Setup	76
5.4.2	Parameter Settings	76
5.4.3	Computation Overhead	77
5.4.4	Communication Overhead	78
5.4.5	Protocol Accuracy	78
6	SenCAPTCHA: A Mobile-First CAPTCHA Using Orientation Sensors	81
6.1	SenCAPTCHA	81
6.1.1	Resilience to Bots	83
6.1.2	System Overview	84
6.2	Security Analysis	90
6.2.1	Brute-force Attacks	90
6.2.2	Machine Learning Based Attacks	91
6.3	First Usability Study—Methodology	95
6.3.1	Study Setup	96
6.3.2	Data Cleanup	98
6.3.3	Qualitative Analysis	98
6.3.4	Demographics	98
6.3.5	Limitations	99
6.4	First Usability Study—Results	99
6.4.1	System Usability Scale	99

6.4.2	Time to Complete	102
6.4.3	Qualitative Feedback	102
6.5	Second Usability Study—Methodology	103
6.5.1	Study Setup	104
6.5.2	Data Cleanup	105
6.5.3	Qualitative Analysis	106
6.5.4	Demographics	106
6.6	Second Usability Study—Results	106
6.6.1	After-Scenario Questionnaire (ASQ)	106
6.6.2	Time to Complete	109
6.6.3	Favorite System	109
6.6.4	Refreshes, Failures, and Skips	109
6.6.5	Qualitative Feedback	110
6.7	Discussion	111
6.7.1	Lessons Learned	111
6.7.2	SenCAPTCHA in Production	113
6.7.3	Future Research Directions	113
7	Conclusion	118
	Bibliography	120
	Vita	134

List of Tables

3.1	Detailed overview information of the 10 cities.	29
3.2	Mixing weights of WeChat and Momo in 10 cities.	33
4.1	The 20 most frequent Interstate movements	45
4.2	OpenStreetMap key-value results of GSP tagged tweets	63
6.1	Success rates for SIFT-, TIF-, and IKT-based animal keypoint detection	93
6.2	Demographics for first usability study	100
6.3	Quantitative results from the first usability study	101
6.4	Demographics for second usability study	107

List of Figures

3.1	Geographic distribution of cell towers, unique users and total data volumes. .	21
3.2	Top 50 apps by total data volumes.	23
3.3	Grouping apps with a three-level tree-like hierarchy.	23
3.4	mlKDE to combine multi-granularity data.	26
3.5	Actual density distributions of QQ at different granularity levels in city F . . .	30
3.6	Density estimates of QQ in city F using different approaches.	32
3.7	Mixing weights from 6:00 pm to 9:00 pm in city B.	35
3.8	Identifying the Uniqueness of Apps based on Mixing Weights	35
3.9	Average log likelihood over all apps.	37
3.10	Mixing weights of WeChat with different bandwidths in city B.	39
4.1	Path of the eclipse shadow across the world and United States. The narrow tracks are called the path of totality. Small blue dots inside the narrow track in (b) refer to U.S. cities. The highlighted states in (b) represent areas where total solar eclipse was visible.	43
4.2	Worldwide and national spatial distribution of tweets. (a) The U.S. contributes more than 89% tweets. (b) States with a large population contribute more. (c) States along the path of totality tweet more per million people.	43
4.3	Distribution of tweets with exact latitude and longitude coordinates.	45
4.4	The 50 most popular interstate trips during the event.	47
4.5	Interstate movement	47
4.6	Intrastate movement	47

4.7	A sample emoji request	50
4.8	Geographic distributions. (a) English emoji requests are made from 110 countries, and the United States, United Kingdom, and Canada account for more than 80% emoji requests. (b) States with large populations request more emojis than those having small populations. (c) After normalizing requests using the state-wise population, states show similar patterns.	50
4.9	Time distribution from Oct. 6, 2018 to Mar. 14, 2020. The data of Oct. 2018 and Mar. 2020 are not plotted in (a) and (b) due to the data incompleteness.	53
4.10	Geospatial distribution across United States	54
4.11	Distribution of scooter tweets with exact GPS coordinates at the city level	55
4.12	The monthly number of geo-tagged tweets in 50 states and Washington D.C. in the United States.	57
4.13	The daily number of tweets from the top 10 states generating most tweets.	57
4.14	Hourly distribution in three phases: Phase 1 (Jan. 25 - Feb. 24), Phase 2 (Feb. 25 - Mar. 14), Phase 3 (Mar. 15 - May 10).	59
4.15	State-level geospatial distribution of geo-tagged tweets across the United States	59
4.16	Distribution of tweets tagged with exact GPS coordinates at the county level	61
4.17	Home location examples on Google Map	63
5.1	Computational model architecture	66
5.2	Bloom filter encrypts a query with the bit-wise encryption scheme	73
5.3	Computation overhead	78
5.4	Communication overhead	79
5.5	Accuracy vs computation overhead	79
5.6	Accuracy vs communication overhead	79
6.1	The red moving ball is controlled by built-in orientation sensors (X and Y axes) on mobile devices. Users tilt their devices to move the ball to target positions to pass the SenCAPTCHA test.	82
6.2	Image mutations to improve attack resilience	83
6.3	SenCAPTCHA's workflow	85

6.4	Example human-generated trajectories	89
6.5	Values of $\mu - 2.5\sigma$ for random guessing attacks	91
6.6	TIF attacks. ● (eye), ● (nose), ● (ear), ● (mouth)	94
6.7	IKT attacks. ● (eye), ● (nose), ● (mouth)	96
6.8	Quantitative results from the second usability study	108
6.9	How current CAPTCHAs would appear on a smartwatch.	114

Chapter 1

Introduction

In recent years, smart mobile devices have become an integral part of people's daily lives. According to Statista [26], global smartphone users in 2020 reached 3.5 billion, which makes 44.87% of the world's population a smartphone owner. The ubiquitous mobile devices generate huge amounts of diverse data every day. Among them, the mobile location is one of the most important and unique data types because it records sensitive personal information and is easy to be produced in various scenarios. For example, when smartphones connect cellular networks, nearby cell towers can locate mobile users approximately. Many social networking platforms, such as Twitter, allow users to publish geo-tagged contents to the public. These location data play a key role in the understanding app usages, tracking human mobility, and discovering insightful geographic patterns. Meanwhile, privacy and security of location data have raised significant concern, especially after the Facebook-Cambridge Analytica data scandal [122] in 2018. It becomes imperative to tackle the paradox between security and usability of location data. As most location-based services are designed for mobile devices, mobile malicious bots may aggressively attack remote location servers. Therefore, mobile-friendly authentication solutions to distinguish between humans and bots are desired and needed.

1.1 Spatial Mobile APP Usage in Cellular Networks

Mobile apps are reshaping the way humans communicate in everyday life. For example, more and more people are sending and receiving messages through instant messaging apps (e.g., WhatsApp) or social networking apps (e.g., Twitter), rather than through Short Message Services (SMS, also known as “text messages”) for their personal communications. In particular, it was reported in 2014 that the data traffic generated by traditional voice calls accounts for less than 10% of all mobile data traffic [31]. Such changes coincided with the large and rapidly growing number of smartphone users, who more and more frequently use data-intensive services. Recent advances in technology and developments in infrastructure [31] have played major role in enabling those data-intensive services. In order to meet the data traffic demands, mobile network operators are constantly upgrading network capacities. App usage data with geospatial information can inform the actions that mobile network operators take in a number of critical operational decisions. Therefore, the abundance of mobile apps’ data, which account for a large proportion of base stations’ traffic load, is ripe for exploration and analysis.

Existing studies focused on understanding the geospatial data generated by mobile apps from various perspectives. Many studies explored the geospatial information to track human mobility [68, 111], ignoring the data volumes. Collecting data from instrumented smartphones operated by very limited number of volunteers (30 volunteers for [68] and 5 volunteers for [111], in particular), these studies could not provide the geographical distribution of various apps on a large scale. Other studies identified diverse usage behaviors of smartphone apps and showed the geospatial distribution of aggregated app usages in same genres (e.g., social networking, game, email) on a large scale [88, 100, 123], but did not demonstrate the geospatial distribution of individual app usage. Antithetically, we assume the perspective of service providers (i.e., app developers and mobile network operators) to focus on the geospatial characteristics of individual apps, with all users considered in aggregate. The question we attempt to address is that, given records of app usages, can we profile the geospatial distribution of data volumes caused by the usage of each app in a robust manner? This question is essential for solving a number of real-world resource

scheduling and optimization problems, such as bandwidth allocation, load balancing, and network caching.

Despite the exciting possibilities, profiling the distribution of data volumes per app is a challenging problem. First, different apps exhibit a high degree of heterogeneity in functions and characteristics. For example, navigation apps are mainly used on highways or local roads; flashlight apps become active after nightfall; and instant messaging apps are typically used continuously. Second, the profiles have to be dynamically updated over time. The same app has different geospatial distributions at different times. This could be caused by many factors, such as fluctuations in the number of app users, movements of app users, usages of app’s provided functions. Third, the mobile data traffic of an individual app can be quite sparse during short intervals, especially for a large number of less popular apps on the long tail. We can only obtain inaccurate geospatial distribution patterns by applying existing sparse models on an individual app’s data traffic.

To address these challenges, we propose a multi-level mixture model based on kernel densities, where each level is represented by a single kernel density estimation. In particular, we consider the data volumes of mobile apps at three different granularity levels. At the individual app level, we consider the total data volumes generated by all people who are using the app. At the group level, we aggregate data volumes from a group of related apps to represent certain properties of the app (e.g., based on category or purposes). At the population level, we aggregate the data volumes from all apps, which represent the overall activeness of all users. After estimating the spatial density of data volumes at each level, we adaptively train the kernel weights, which not only profile the geospatial distribution of an app more robustly, but also provide interpretable insights about the app and its peers, at multiple levels. Finally, the three kernel estimations are combined together, based on their respective weights, to build the multi-level mixture model.

Our contributions can be summarized as follows. First, we provide a multi-granularity solution of mobile apps’ data, by considering individual level, group level, and population level data volumes, to overcome the sparsity of individual level data and build a multi-level mixture model based on kernel densities with a high fitting accuracy. Second, our model can handle the heterogeneity of different apps as well as the dynamic profiling for the same

app over time. For both different apps and the same app at different time points, our model adaptively assigns weights for the three levels of data. The weights can also be used to analyze the uniqueness of apps. Third, our model is general enough to be applied to other domains with similar formats.

1.2 Publicly Shared Location Data on Social Networks

In the past decade, social networks have emerged and grown into one of the most important communication tools to broadcast oneself, build connections with friends, and express opinions and emotions to the public. Many social networking platforms, e.g., Facebook, Twitter, Instagram, and Snapchat, allow users to optionally geo-tag their posts on social media. Such location information bridges the online and offline worlds by linking virtual contents and activities with real places.

Among all popular social networks, Twitter stands out as an open data source for numerous location-based research studies due to its following features. First, Twitter ranks as one of the most active social media, generating 500 million tweets per day [107]. Second, Twitter users expect what they post, including physical locations, to be publicly available, addressing potential ethical concerns of collecting user data. Third, multiple types of locations, namely user profile locations and tweet locations, are available on Twitter. Profile locations consist of free-form text, and tweet locations can be either general places or GPS coordinates. Finally, Twitter provides APIs for randomly sampling tweets of interest, whereas many other social network platforms prohibit automated content scrapers (e.g., Facebook [32] and Instagram [54]).

Twitter location data inspire and motivate a large number of research works in various fields, such as disaster tracking [57, 4], public health monitoring [28, 121], human mobility analytics [48, 58], and even presidential election predictions [101, 55]. Researchers utilized Twitter as a crowdsourcing platform to harvest tweets across regions of interest. Since geo-tagged tweets recorded where, when, and what the public did, they were crucial for discovering knowledge and revealing insightful patterns.

In this dissertation, we collect and analyze large amounts of Twitter streaming location data to study human mobility, service petitions, transportation, and disease propagations. We demonstrate Twitter locations are social-good but may leak sensitive information, such as home addresses. Our studies examine how big geo-tagged data on social networks facilitate large-scale and complex social computing research in multiple scenarios. Specifically, we chase the 2017 Great American Eclipse on Twitter to look into human mobility during once-in-a-lifetime events. We also present the petition patterns of new emojis on Twitter inside the United States and across the world. Then, we take Twitter as our lens to show the geographic usage patterns of shared electric dockless scooters in English-speaking countries. Finally, we create a large geo-tagged COVID-19 Twitter dataset to track residents’ reactions and point out the potential private information leakage.

Our contributions can be summarized as follows. First, we overcome the challenges of collecting, preprocessing, and analyzing streaming big social media data to advance the research of social physics [5]. In each case study, millions of real-time tweets are harvested for in-depth analytics. Second, heterogeneous location data, including countries, states, counties, and GPS coordinates, are considered to extract underlying patterns. For example, we innovatively infer local time zones, which was disabled by Twitter APIs in May 2018, from state locations to profile temporal distributions at the hour level. The GPS-tagged data enable us to reveal the possible private location leakage, such as home addresses. Third, all of our case studies keep up with recent or current events and provide timely feedback and insights for understanding these events.

1.3 Online Secure Sharing of Private Locations

Location data are becoming increasingly popular on social networks and mobile apps. Smartphone users are sharing location data, ranging from ride-sharing apps to geographically enhanced games, such as the Pokemon Go, among others. In these scenarios, one big concern is the privacy of users, as unexpected leaks of users’ location trajectories will allow potentially malicious attacks to gain advantages in the real world [30]. For example, by knowing when a user leaves and returns home each day, a potential third-party attacker could identify the

time periods during which the user’s home is not occupied. The attacks range from mild inconveniences such as privacy leaks to much more serious attacks such as break-ins.

One significant challenge in keeping location data secure is that we should not trust servers as safe against attacks [64]. Indeed, the significant increase on hacking activities against servers means that if we store non-encrypted data on servers, we will face the risks of data leakage when the servers are compromised. Perhaps paradoxically, when building apps that involve location data, servers typically perform extensive computation on the location traces, making it necessary for the servers to be able to decrypt data as needed. For example, Facebook servers need users’ non-encrypted locations to find nearby friends. Mobile service providers need users’ rough locations to analyze smartphone usage behaviors [77]. Consequently, we ask, is it possible for us to maintain the security of location traces while *allowing* servers to perform location-specific computations, such as calculating intersections?

Given the demands to keep user location data secure, as well as the need for the servers to perform basic operations on the trajectory data, in this dissertation, we develop a secure computation framework on location data where the servers have no knowledge of its original data, i.e., the servers do not keep the private keys to decrypt data. In this way, compromised servers will not cause leaked user data. Our work is motivated and enabled by the recently developed fully homomorphic encryptions, where recent progress demonstrated it is feasible to perform meaningful and predictable computations on encrypted data without decrypting them first [37, 33]. Although existing primitives primarily support simple operations such as bit-wise operators, additions, and multiplications, in this work, we build on these homomorphic techniques, and expand them to support location-specific calculations. Specifically, we focus on one commonly used building-block operation in location based data processing: computing the location intersections.

Formally, this computational operation assumes that two users, Alice and Bob, want to find out if their location datasets (e.g., collections of singleton locations, or trajectories, or areas within specified boundaries) have intersections. The application model works as follows: Alice first publishes her location datasets that are encrypted using her public key, via the cloud aggregation server, so that Bob does not get access to the plain data directly. Instead, Bob can build a query based on his location sets and send it to the aggregation

server. Note that as Bob also cares about his privacy, his query is also encrypted by Alice’s public key. To increase security, before being encrypted, both Alice’s location dataset and Bob’s query are inserted into an advanced data structure called the bloom filter [8], which hashes all location data into binary strings. In this way, even when the cloud aggregation server is compromised and the secret key of Alice is hacked at the same time, the plain location data is still not leaked. Next, the aggregation server performs the *matching* step, where Alice’s encrypted datasets are matched against Bob’s query, and a (still encrypted) computational result is returned to Alice. Alice can decide whether there is an intersection between the incoming query and her datasets by decrypting the result using her own private key. If there is an intersection, Alice is probably able to send additional information to Bob.

We now make a few remarks on this computational model. First, by design, this model ensures that the location datasets are fully secure, as Alice will only publish the hashed datasets after encryption. On the side of Bob, the query location data is also pre-processed using hash functions, and then encrypted using Alice’s public key. Hence, Alice and Bob never need to share the plaintext of their location trajectories directly. Second, this computational model is particularly secure against compromised servers, as the aggregation server does not keep any plaintext of location trajectories. Finally, only queries that return positive results (i.e., there are intersections) will lead to further interactions between Alice and Bob. For queries that do not return intersections, Alice and Bob do not learn about each other’s locations.

The contributions of this paper are as follows. First, we design and propose a fully homomorphic version of the bloom filter. Second, we develop multiple optimized protocols that allow one party to determine, in a private and secure manner, whether or not a second party’s trajectory has an intersection with the first party. Our design is fully flexible, meaning that each user is able to specify what kind of datasets (e.g., trajectories, areas, and isolated points) they would like to make visible, and be queried by other users. Third, we offer working prototypes based on the open-source homomorphic libraries. Additionally, our preliminary evaluation results on a real-world dataset, which is collected in a city area on users’ smartphones, demonstrate the feasibility of the proposed approaches as well as the security of the protocol designs.

1.4 Blocking Malicious Bot Attacks on Locations

Bots (autonomous Internet programs and scripts) are an important piece of the Internet. For example, Googlebot crawls the Internet to gather data necessary to power Google’s search engine. However, over 55% of bot traffic is malicious, with 94.2% of websites experiencing at least one bot attack every 90-days [53]. These malicious bots are responsible for a range of undesirable actions, such as comment spamming, password guessing attacks, and click fraud. Therefore, differentiating between human- and bot-initiated actions is a critical task on today’s Internet.

To fight against malicious bots, von Ahn et al. [78] proposed the Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) system in 2000. AltaVista [73] implemented one of the first practical CAPTCHA schemes and used it to block fraudulent account registrations. Today, there are many different CAPTCHA systems, with most systems being able to be categorized into text-based, image-based, audio-based, video-based, and puzzle-based CAPTCHAs. While recent updates to Google’s ReCAPTCHA system (i.e., NoCAPTCHA) can reduce the frequency with which users need to solve a CAPTCHA, they do not remove the need for CAPTCHAs altogether. As such, research is still needed to increase the effectiveness and usability of CAPTCHA systems.

Most existing CAPTCHAs are designed for desktop users, failing to offer user-friendly mechanisms to mobile users who frequently generate and query personal information including location data. It is problematic to run these CAPTCHAs on mobile devices for a couple of reasons. First, many CAPTCHAs are not designed to work well when screen size or resolution is limited. While mobile phone screen sizes and resolutions continue to increase in size, this is not true of wearable technology (e.g., smartwatches). For example, the 44mm Apple Watch has a screen size of just 9.77 cm^2 and a resolution of only 368 by 448 pixels. Second, CAPTCHAs designed for desktops fail to take advantage of the sensors available on modern mobile devices. This is unfortunate, as leveraging the phone’s sensors could help compensate for the limited screen size and resolution.

In this dissertation, we present SenCAPTCHA, an orientation sensor-based CAPTCHA scheme that addresses both these limitations. SenCAPTCHA works by displaying a picture

of an animal and asking users to tilt their phones to guide a red ball into the center of that animal’s eye. Using the trajectory of the ball, SenCAPTCHA can distinguish between human- and bot-generated solutions. SenCAPTCHA requires very little space to display and complete the CAPTCHA, allowing it to work well on mobile and wearable devices with small screens (e.g., smartwatches). Additionally, using the orientation sensor avoids the need for users to type text or select images, tasks which can be difficult on smaller screens (e.g., awkward keyboards, fat finger problems).

The contributions of this work are summarized as follows. The design of SenCAPTCHA, a mobile-first CAPTCHA that uses a mobile device’s orientation sensor to compensate for limited screen space. We demonstrate that by rotating and tiling the animal image shown to users by SenCAPTCHA, we can prevent machine learning algorithms from locating the animal’s eye (i.e., identifying the solution to the puzzle). By examining the trajectory used to solve SenCAPTCHA, we can also differentiate between bots that are searching for the animal’s eye and users who know where it is, helping prevent bot-based attacks against SenCAPTCHA. We conduct a usability study of SenCAPTCHA with 270 participants. The results of this study identify how various features—how the image is modified, the starting point for the red ball, and how close the ball needs get to the eye—affect completion time and perceived usability. We conduct a usability study with 202 participants comparing SenCAPTCHA to four other representative CAPTCHA schemes demonstrating that SenCAPTCHA outperforms these systems regarding perceived usability and completion time. Most participants indicated that SenCAPTCHA was their favorite CAPTCHA of those tested.

Chapter 2

Related Work

In this chapter, we summarize recent literature on the location-inspired studies in the context of cellular networks and social networks, and solutions for secure sharing locations and combating location bots.

2.1 Spatial Modeling of Mobile App Usages

Many spatial models [34, 72, 100, 123, 128, 129], which can profile the spatial distribution of app usages, have been explored in the existing literature. Fu et al. [34] explored the geographic dependencies of estate values from various granularities and exploited these dependencies in a probabilistic ranking model. Zheng et al. [128] developed a predictive model to forecast the air quality monitored by weather stations. The proposed data-driven model considers local temporal factors, global spatial factors, meteorological data and sudden changes. Both Fu et al. [34] and Zheng et al. [128] also used large-scale information to aid in fine granularity prediction. Studies like [100, 123] characterized geospatial dynamics of aggregated app usages, but they did not demonstrate the geospatial distribution of individual app usage. Zhou et al. [129] proposed a predictive method using spatial-temporal kernel density estimation (stKDE) to deal with complex spatio-temporal dynamics and sparsity at high resolutions in large-scale datasets. A spatio-temporal weight function was adopted based on informativeness to the current predictive task to incorporate temporal and spatial patterns in ambulance demand.

Lichman et al. [72] developed accurate individual-level models of spatial location based on kernel density estimates (KDE). Spatial distribution modeling at the individual level is a challenging problem due to the sparsity of data and heterogeneity of mobility patterns among individuals. Although direct application at the individual level may cause a significant over-fitting problem, such problems can be eliminated with a hierarchical extension of the traditional KDE. The hierarchical KDE has inspired us to develop our model incorporating the data traffic at the scale of the group level and the population level to deal with the sparsity problem for a single app. However, the characteristics of our data and our approach are different with [72] in the following aspects. First, the data samples of [72] were randomly distributed in a large region, while our data samples were specific cell tower locations with dynamic data volumes. Second, Lichman et al. [72] assigned each individual component the same weight, while the individual components in our approach are assigned dynamically varying weights based on their features at different given time.

2.2 Geo-tagged Social Media Data Analytics

There is rich literature investigating how locations shared on social networks facilitate diverse research studies. We mainly focus those works from four categories, namely, human mobility, online campaigns, transport studies, and disease propagations. For human mobility, Hawelka et al. [48] explored international travel patterns using almost one billion geo-located tweets and concluded that Twitter was a useful proxy for understanding and quantifying global mobility patterns. Similarly, Jurdak et al. [58] demonstrated geo-tagged tweets could capture rich features of human mobility including the diversity of movements within and between cities. For social network based campaigns, Harlow et al. [47] reported that social networks played an important role in contemporary activism in both the United States and Latin America. Borge-Holthoefer et al. [10] revealed the structural and dynamical patterns emerging from the activity of Twitter around the May 15th movement in Spain.

Many existing works leverage geo-tagged social media data to study transport related research questions. Grant-Muller et al. [43] summarized how to use social media sources, including location names, to enhance transport data collection. Maghrebi et al. [79] extracted

travel modes from Twitter data posted in Melbourne metropolitan areas. There also exists a considerable body of literature examining how location-based social media can help understand and monitor disease propagations. For example, Tran et al. [114] utilized a large-scale analysis of geo-tagged social media messages to understand the public reactions towards Ebola. Qazi et al. [90] released the GeoCoV19 dataset, which contained hundreds of millions of multilingual geo-tagged tweets, to combat the COVID-19 pandemic.

Inspired by these existing works, we further explore the locations posted on social networks in the context of recent and emerging events, including 2017 Great American Eclipse, online emoji campaigns, emerging shared dockless e-scooters, and COVID-19. In each case study, we collect millions of streaming tweets and conduct timely and in-depth big data analytics. Besides, diverse locations extracted from user profiles and what people post on social media have been studied.

2.3 Secure Sharing of Private Locations

Previous work on keeping location secure has investigated multiple directions. The first direction, which is called k-anonymous obfuscation [18], tries to hide the true locations of users by obfuscating them to the granularity of larger cells. Such methods, though privacy-aware, make it harder to develop applications that require precise user locations. Another method is through statistical methods [99], which add random noise to the samples of individual users, but keep the global statistical parameters to be more or less reliable. Such methods are only suitable for large-scale statistical needs, but are not useful where one user’s data needs to be exploited for application needs, e.g., ride sharing and navigation.

Our proposed approaches fall under the third direction that performs encryption and decryption methods on the location data in order to achieve location security and privacy even when servers may be compromised. One recently developed encryption scheme suitable for this purpose is the homomorphic encryption, which aims to support complex processing on the encrypted data without decrypting them first, yet still yielding results that, once decrypted, are meaningful and correct. The homomorphic encryption has widely inspired applications [75, 69] of cloud computing.

Existing work in this direction has attempted to apply homomorphic computing techniques to process location data. For example, the work in [46] utilized partially homomorphic encryption to develop a privacy-preserving location proximity protocol called the InnerCircle. A more recent work [7] adopted homomorphic encryption to determine an optimal meeting location for a group of users. Note that all of these works are implemented based on partially homomorphic encryption methods that only support limited types of operations, i.e., either addition or multiplication, but not both.

A common task needed in location data processing is related to checking intersections between private sets. To this end, previous work has taken advantage of a data structure called the bloom filter, which supports constant time checkings on set memberships. For example, the work in [27] proposed the garbled bloom filters, based on which the oblivious Bloom intersection is performed to check the private set intersections. Another work [127] used the bloom filters to represent the location tags to conduct private proximity tests. In [60], the bloom filter was combined with partially homomorphic (Goldwasser–Micali) encryption to design an outsourced private set intersection protocol. Because partially homomorphic encryption only supports either the multiplication operation or addition operation, the work [60] had to rely on additional methods, such as the Sander Young Yung technique [97], to mimic the second operation but with a failure probability.

In contrast to these existing efforts, our work is directly motivated by the recent progress to develop fully homomorphic encryption schemes, such as the well-known scheme developed by Gentry in 2009 [37]. Compared with partially homomorphic encryption, this scheme is far more powerful because it supports both multiplication and addition operations simultaneously and allows for arbitrary computations on the encrypted data principally [2]. Inspired by Gentry’s work [37], several practically feasible fully homomorphic encryption schemes have been developed [33, 117, 12]. Therefore, we build our protocol on top of existing fully homomorphic computing libraries, but we note that future developments of better paradigms will lead to lower computing cost and overhead, as well as better security in our system.

2.4 Authentication Solutions to Battle Location Bots

The CAPTCHA is one of the most widely used authentication mechanisms to defend against malicious bot attacks. Many different categories of CAPTCHA schemes have been proposed in the literature. In this section, we discuss several of these approaches.

Google’s NoCAPTCHA: Recently, Google updated its reCAPTCHA v2/v3 (i.e., NoCAPTCHA) [51] to identify bots by examining information regarding the user’s network connection, browser settings, cookies, and previous browsing behavior. In many cases, NoCAPTCHA can completely avoid user interaction or only ask the user to click a checkbox that says, “I’m not a robot.” Still, there are many instances in which NoCAPTCHA is unable to automatically determine whether a user is human. This can be because of technical limitations (e.g., cookies are deleted, an incognito web browser session is used, or JavaScript is disabled [104]) or because NoCAPTCHA’s algorithm lacks sufficient evidence to confirm whether the user is human. In these cases, NoCAPTCHA falls back to showing users a traditional CAPTCHA. As such, NoCAPTCHA does not remove the need for other CAPTCHAs, but rather only reduces how often CAPTCHAs must be shown.

Text-Based CAPTCHAs: Text-based CAPTCHAs take advantage of the inherent difficulty of Optical Character Recognition (OCR). The first practical text-based CAPTCHA was invented by Altavista [73] in 1997. Later, von Ahn et al. [120] proposed reCAPTCHA, a system that asked users to transcribe text scanned from old books. There is also research that seeks to make it more difficult to perform OCR. Baird et al. [3] proposed the ScatterType CAPTCHA, which resisted the segmentation of characters from a word. Hsieh et al. [50] adopted the scale-invariant feature transform (SIFT) algorithm to make characters indiscernible for bots but distinguishable for humans. In recent years, text-based CAPTCHAs have been shown to be vulnerable to deep learning-based approaches [41, 17, 105]. The future of text-based CAPTCHA seems uncertain [103], with companies like Google gradually phasing out text-based CAPTCHAs.

Audio-Based CAPTCHAs: To make CAPTCHAs available for individuals with visual impairments, a range of research explored audio-based CAPTCHAs [49, 62, 19, 35, 66], which rely on the difficulty of transcribing the text, particularly when there is significant

background noise [22]. However, audio-based CAPTCHAs are limited in that they require different datasets for each spoken language and also that in many languages have multiple letters that are difficult to distinguish aurally [108].

Video-Based CAPTCHAs: In 2009, Kluever et al. [61] proposed the first video-based CAPTCHA, where users watch a video and select the correct labels associated with the video. The first widely deployed video-based CAPTCHA was released by NuCaptcha [52] in 2010. Different from Kluever et al.’s work [61], NuCaptcha provides a video of codewords moving across a dynamic scene. Later research [124, 16] defeated the security of NuCaptcha by adopting moving-image object recognition (MIOR) techniques. Still, compared to static text- and image-based CAPTCHAs, video-based CAPTCHAs are believed to provide a higher level of security [108]. On the other hand, loading large files may adversely affect user experiences, especially for mobile users where bandwidth is limited.

Image-Based CAPTCHAs: Bongo, developed by Mikhail M. Bongard [9], is one of the earliest visual pattern recognition problems used as CAPTCHAs [119]. Chew et al. [21] were among the first to study labeled photograph-based CAPTCHAs. Since then, many image recognition-based works have been proposed, such as IMAGINATION [25] and Asirra [29]. Gossweiler et al. [42] proposed CAPTCHAs by requiring users to identify an image’s upright orientation. Matthews et al. [80] proposed a scene tagging test, which relies on the ability of users to recognize the relationships between irregularly shaped objects embedded in a background image. However, many studies [108, 40] have identified disadvantages with image-based CAPTCHAs.

Puzzle-Based CAPTCHAs: Gao et al. [36] proposed the jigsaw puzzle-based CAPTCHA by dividing an image into misplaced pieces. Kaur et al. [59] designed a boolean algebra puzzle-based CAPTCHA using circuit diagrams consisting of basic logic gates like OR, AND, NOR, among others. Conti et al. [23] implemented the CAPTCHAStar system based on interactive shape discoveries. These CAPTCHAs share in common that they mostly rely on conventional puzzle designs and require additional efforts to pass these puzzles compared to the image-based or text-based ones.

CAPTCHAs Designed for Mobile Devices: Researchers have also proposed some CAPTCHA systems for mobile users to protect their sensitive data. The Drawing

CAPTCHA [102], one of the first CAPTCHAs designed for mobile devices, required users to draw appropriate lines on a screen to solve the CAPTCHA challenge. Lin et al. [74] proposed CAPTCHA Zoo by projecting 3D natural animals into 2D models with a natural background. In addition, several motion sensor based CAPTCHAs have been developed, such as accCAPTCHA [71], GISCHA [126], and CAPPCHA [45]. However, these systems did not demonstrate the involved hard problems could resist computer vision based attacks. Our proposed SenCAPTCHA also leverages built-in motion sensors on mobile devices, and designs a flexible game-like image-based puzzle for users to solve. More importantly, we conduct machine learning and deep learning attack experiments to demonstrate its security.

Chapter 3

A Multi-Granularity Perspective for Spatial Profiling of Mobile Apps

In this chapter, we leverage cell tower locations to investigate the geospatial distribution of individual mobile apps on cellular networks. For apps with sparse data, it is challenging to estimate their geospatial distribution robustly. Instead of only focusing on the app itself data, we take a multi-granularity view on the data generated by all apps and apps in corresponding groups. We propose a multi-level mixture of kernel density estimation (mlKDE) model, which assigns optimal weights for each granularity level, to smooth the spatial profiling. Note that this chapter is based on my published paper:

Y. Feng, Z. Lu, W. Zhou, Q. Cao and X. Li. A Multi-Granularity Perspective for Spatial Profiling of Mobile Apps. Information Sciences 430, (2018): 127-141.

3.1 Problem Formulation and Data Description

In this section, we first formulate the research problem and discuss a general scenario where our model could be applied. Then we describe the spatial feature of cell towers in our dataset. Finally, we provide a descriptive study of the app categories used in our model.

3.1.1 Problem Formulation

In this study, we focus on the geospatial distribution of data volumes due to mobile app usages. As the cell towers provide services to nearby users, it is possible to measure various activities through these towers, such as data volumes downloaded by individual apps, and estimate the overall distribution of such measures over the region. Formally, given records that can be written as

$$\langle \text{TowerID}, \text{TimeStamp}, \text{AppID}, \text{Volumes} \rangle, \quad (3.1)$$

We are interested in finding the density distribution of download volumes attributable to each given app over the region. The objective is to identify estimates that are smooth, efficient, robust, and interpretable.

It is straight-forward to slice time into preferred level of granularity, such as per minute, or per hour. When slicing time into refined, short intervals, the pass-through data volumes will be so low that density estimates at the app level will become unstable. It is hard to summarize the distribution smoothly when the data change abruptly. One possible source of information we could leverage is the hierarchy of app categories. Information at the population level (i.e., all apps) may indicate the overall activeness of users at that time of day. Information at the group level (i.e., all apps in the corresponding category) may indicate the users' interest.

Investigating geospatial distribution of apps' data is important for several reasons. First, understanding geospatial distribution of apps' data can provide valuable insights on the need to increase network capacity based on data traffic demands. Monitoring and analyzing apps' geospatial distribution can assist mobile network operators in improving their customers' user experiences, such as upgrading network hardware and software, and deploying new equipment like portable base stations. Second, the geospatial distribution of apps' cellular data at a large scale provides valuable information about users' smartphone usage behaviors collectively, which can be utilized by advertisers [113], location-based service providers [70], social scientists [1] and even tourists [84]. Cellular data based descriptive statistics like which apps are most popular in specific regions, what best app choices will match users' preferences

at different locations, will offer fundamental information for future investigations. Third, modeling geospatial distribution of an individual app is valuable for the further development, promotions, and maintenance of the app itself. At the design stage, analyzing and comparing the geospatial distributions of the competitive apps can help developers discern the relation between these apps' geospatial distributions and their specific functions. When promoting the app, our model allows users to obtain the geospatial distribution of the app, which can be used for ad placement to maximize promotional effects. During the operations and maintenance phase, data traffic demands for different regions can be calculated aggregately, and app local-based service caches can be added where app services are heavily requested to reduce the response time.

Note that our proposed methodology may be expanded to other general scenarios. The given observations can be written as

$$\langle \text{SensorID}, \text{TimeStamp}, \text{EventID}, \text{EventValue} \rangle. \quad (3.2)$$

Our goal is to model the geospatial distribution of the measurements for each event, collected by a number of fixed sensors whose locations are known. The location is usually denoted as a longitude-latitude pair in geographical coordinates, and the **TimeStamp** is the date and time when an event occurs. The **EventValue** is a quantity that represents the measurement of interest for the event.

Since the goal is to profile the geospatial distribution, it is necessary to ascertain the density of the event measurement for any given longitude-latitude pair. Since the sensors are deployed with varying densities over the region of interest, it is critical to craft a concise description of such densities to support efficient understandings and decision-making.

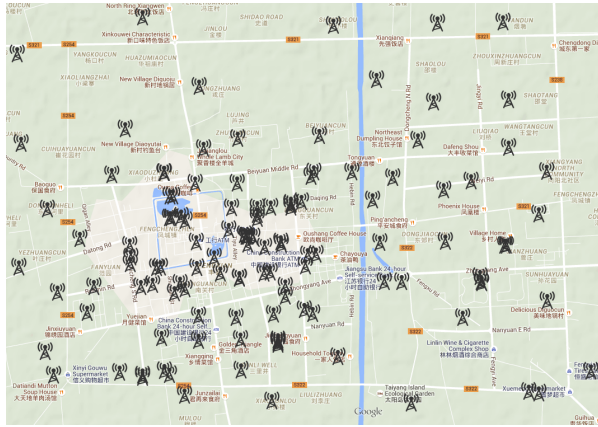
A possible example scenario of utilizing our model is analyzing the sales data from multiple stores or sales outlets where the sales of each product may be aggregated at the brand level or product type level. Like Equation 3.1, the data may be written as $\langle \text{StoreID}, \text{TimeStamp}, \text{UPC}, \text{Quantity} \rangle$. Our methodology can be utilized to profile the sales volumes of any given product, and compare that information to products from the same category, and the overall sales across a region with many stores.

3.1.2 Cell Tower Location

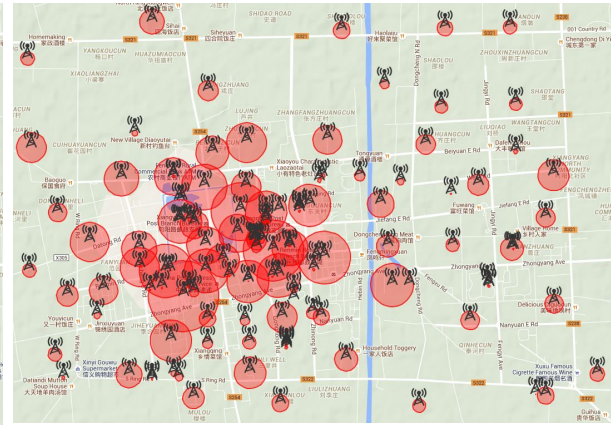
Figure 3.1(a) displays the distribution of cell towers all over a city to ensure service coverage. The distribution of cell towers is dense in the downtown area, while relatively sparse on the outskirts of the city. At each tower, various measurements can be made that probe into the distributional density of the quantity of interest. In Figure 3.1, we visualized measurements of each tower using red circles. The radiuses of circles represent the quantity of interest, such as the number of unique users in Figure 3.1(b), and the total data volumes in Figure 3.1(c) and Figure 3.1(d). We can see that the downtown area is densely populated, where each cell tower serves a large number of users, but not all towers possessing large total data volumes are located in the downtown area. We can also observe that the data volumes are not proportional at each tower to the number of unique users served. In particular, towers serving a small number of users might also process a large amount of total data volumes. Figure 3.1(c) and Figure 3.1(d) demonstrate that data volumes vary greatly over time, even though cell towers remain fixed.

3.1.3 Apps Information

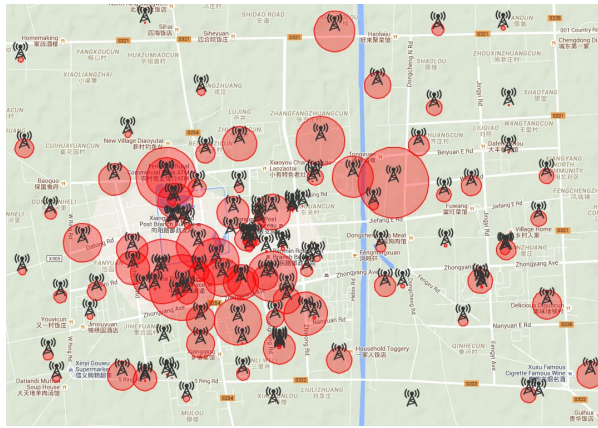
Millions of mobile apps are available on the app market [106], and each generates a large amount of download and upload data traffic through nearby cell towers. We expect that different apps have different data requirements in term of data rates, availability, etc, so examining the apps and their categories enhances modeling the data requirement for each app. For example, based on our data, the 50 most popular apps are shown in Figure 3.2, which indicates a long-tail distribution. In this figure, we can see that social software apps, such as instant messaging, social networking, web surfing, browsing and daily life services tools, are very popular. Moreover, navigation, map, and traffic apps are also frequently used. Note that since our data is collected from the 2G network (the dominating technology in the areas we study), if a user switches to Wi-Fi, we cannot observe the data. As a result, more data intensive apps (like video streaming) are under-represented, since users often switch to Wi-Fi for extended use of these apps in order to save data costs. Meanwhile, navigation



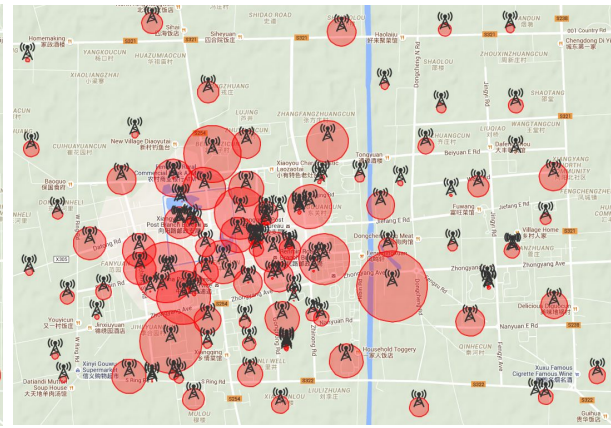
(a) Tower Locations



(b) Number of Unique Users (6:00 pm - 7:00 pm)



(c) Total Data Volumes (6:00 pm - 7:00 pm)



(d) Total Data Volumes (7:00 pm - 8:00 pm)

Figure 3.1: Geographic distribution of cell towers, unique users and total data volumes.

apps are often used in areas without good Wi-Fi coverage, and thus users have to rely on the 2G network regardless of data demands.

As suggested in Figure 3.2, some apps have high data volumes due to their popularity, but there are many others that were less popular with sporadic observations. For apps with a small amount of data, it is very challenging to estimate their distributions in a robust way. Thus, we are motivated to pursue a high data stability by aggregating related apps. This aggregation may happen among apps that share similar properties or functionalities, and be done at a suitable granularity level. In our dataset, we grouped apps into a three-level tree-like hierarchy, wherein each app only belongs to a single top category, as illustrated in Figure 3.3(a). However, our framework is general enough to easily extend to multiple categories, as illustrated in Figure 3.3(b).

3.2 Modeling Individual Apps' Spatial Distributions

Based on cell towers' latitude and longitude, we modeled the data traffic distribution with probabilistic density functions over this two-dimensional space. To ensure a more flexible model that will adapt to the topology of this two-dimensional space, we shall use mixture models. It is well understood that Gaussian-based kernel density models are usually more flexible and accurate than the Gaussian model and the Gaussian mixture model. In [72], the authors argued that neither the Gaussian model nor the Gaussian mixture model can be used to accurately characterize sparse geospatial distributed data. Therefore, based on these findings, we use the kernel density model rather than the Gaussian model or the Gaussian mixture model.

3.2.1 Kernel Density Estimation

During the time window t , the data volumes downloaded by one app from tower i is denoted as $vol_i^{(t)}$. In this subsection, since we focus on spatial models, we ignore the time period identifier t on the superscript. The location of tower i is denoted as $(lat_i, lon_i)^T$ and our observations are collected as $(lat_i, lon_i, vol_i)^T$, where i is a subscript that goes over each tower ($i=1,2,\dots, n$) and the superscript T means transpose. Given all data points

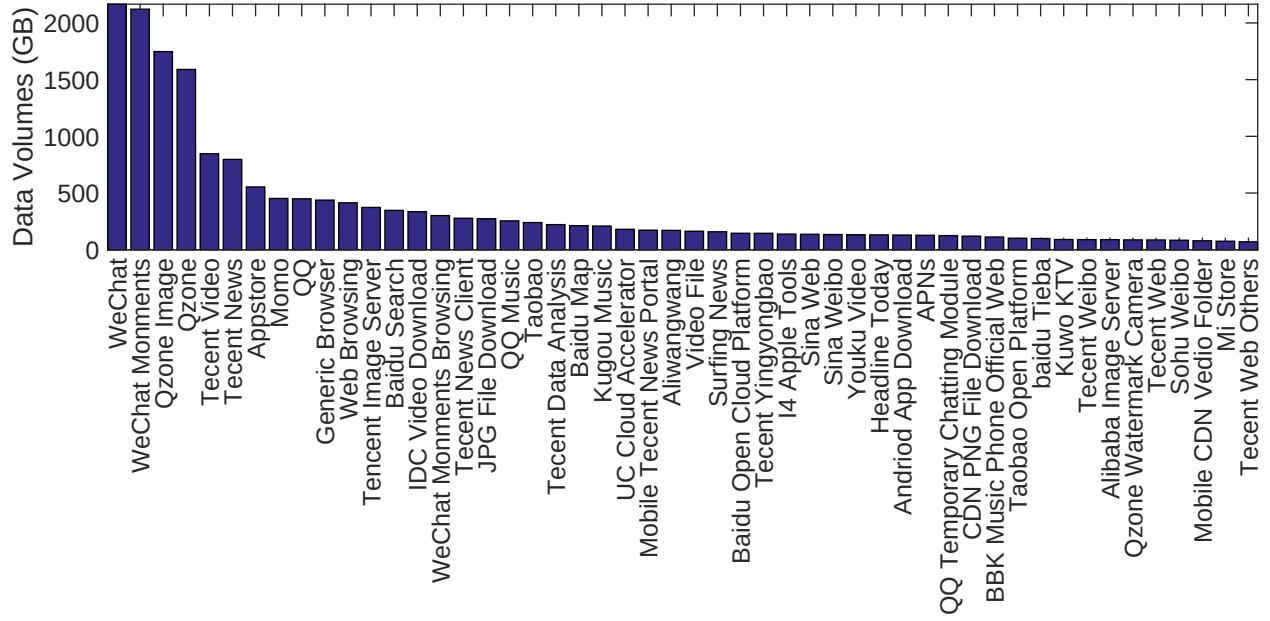
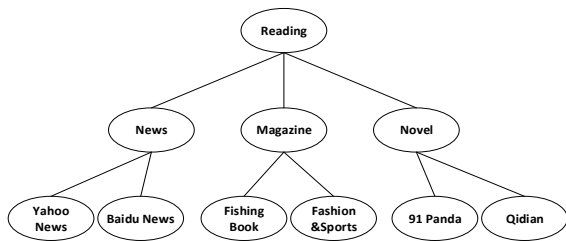
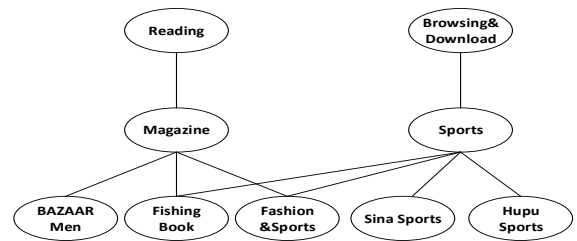


Figure 3.2: Top 50 apps by total data volumes.



(a) Single-Category



(b) Multi-Categories

Figure 3.3: Grouping apps with a three-level tree-like hierarchy.

$P = \{\vec{p}_1, \vec{p}_2, \dots, \vec{p}_n\}$, where $\vec{p}_i = (lat_i, lon_i, vol_i)^T$ is the observation from tower i , and a bandwidth h , then the Gaussian kernel density estimation (KDE) can be written as

$$f_{KDE}(\vec{x} | P, h) = \frac{1}{2n\pi h} \sum_{i=1}^n vol_i \exp \left\{ -\frac{1}{2} (\vec{x} - \vec{x}_i)^T \Sigma_h^{-1} (\vec{x} - \vec{x}_i) \right\}, \quad (3.3)$$

where $\Sigma_h = \begin{pmatrix} h & 0 \\ 0 & h \end{pmatrix}$ and $\vec{x}_i = (lat_i, lon_i)^T$.

Lichman et al. [72] proposed to adaptively learn the bandwidth $H = \{h_1, h_2, \dots, h_n\}$ for different event points (GPS-located Twitter tweets and Gowalla checkins). Given such an event point is denoted as $\vec{x}_i = (lat_i, lon_i)^T$, the Gaussian kernel density function can be written as

$$g_{KDE}(\vec{x} | P, H) = \frac{1}{2n\pi} \sum_{i=1}^n \frac{1}{h_i} \exp \left\{ -\frac{1}{2} (\vec{x} - \vec{x}_i)^T \Sigma_{h_i}^{-1} (\vec{x} - \vec{x}_i) \right\}, \quad (3.4)$$

where $\Sigma_{h_i} = \begin{pmatrix} h_i & 0 \\ 0 & h_i \end{pmatrix}$ and $P = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n\}$. In our case, an adaptive bandwidth for our events is not necessary. Unlike the events in [72], our events (i.e. bytes of data downloaded through cell towers) occurred exclusively at the locations of cell towers. Therefore, every byte of the data is associated with a nearby cell tower at a known and fixed position. Based on our observations, the spatial density differences of towers at downtown areas and on outskirts are not distinct enough to assign an individual bandwidth for each tower.

3.2.2 The Multi-level Mixture of Kernel Density Estimation

With non-negative mixing weights $\pi = \{\pi_1, \pi_2, \dots, \pi_G\}$ such that $\sum_{g=1}^G \pi_g = 1$, the multi-level mixture of kernel density estimation (mlKDE) can be written as

$$f_{mlKDE}(\vec{x} | V, H, \pi) = \sum_{g=1}^G \pi_g f_{KDE}(\vec{x} | P_g, h_g), \quad (3.5)$$

where G is the number of components, $V = \{P_1, P_2, \dots, P_G\}$, P_g stands for all data points from component g , and $H = \{h_1, h_2, \dots, h_G\}$, h_g is the bandwidth of component g . In our case, since all data points of different components are collected from the same towers, the

bandwidths of all the components should be the same. So, Equation 3.5 can be written as

$$f_{mlKDE}(\vec{x}|V, h, \pi) = \sum_{g=1}^G \pi_g f_{KDE}(\vec{x}|P_g, h). \quad (3.6)$$

3.2.3 Algorithm Implementation

To overcome the sparsity of data at the individual level, the mlKDE with the Gaussian kernels was used to describe the data traffic density distribution of an individual app. The basic idea of the mlKDE is first applying Gaussian KDE on multi-granularity data, then combining all these KDEs with corresponding and reasonable weights, and finally determining the target density distribution. Figure 3.4 illustrates how the mlKDE is used to estimate the data traffic density distribution of app i at time window of t in City B's downtown area. Three Gaussian KDE components, population component, group component and individual component, are combined together with respective weights to build the mlKDE. Each component is a Gaussian KDE over the 2-D (latitude, longitude) geographical space, expressed as Equation 3.3. The population component aggregates total data volumes of all apps at time window t ; the group component aggregates total volumes of all the apps in the same group to which app i belongs at time window t ; the individual component only captures the data volumes of app i at the time window of $t - 1$.

Next, we must train the mixing weights π_g in Equation 3.6 which are crucial to the accuracy of mlKDE. In our case, three components were considered, i.e. $G = 3$, so $\pi = \{\pi_1, \pi_2, \pi_3\}$ and $\sum_{g=1}^3 \pi_g = 1$. We took P_t (the population data at time window t), G_t (the group data at time window t) and I_{t-1} (individual data at time window $t - 1$) as the training set, and I_t (the individual data at time window t) as the validation set. For each observation $(lat_i, lon_i, vol_i)^T$ in the validation set I_t , we calculated its probability density under each component using the training set. Then, we optimized the combinations of $\pi = \{\pi_1, \pi_2, \pi_3\}$ by maximizing the joint likelihood for all observations in I_t . Because the observation $(lat_i, lon_i, vol_i)^T$ was equivalent to sampling $(lat_i, lon_i)^T$ for vol_i times from tower i , it could be regarded as a weighted observation $(lat_i, lon_i)^T$ with a weight of vol_i .

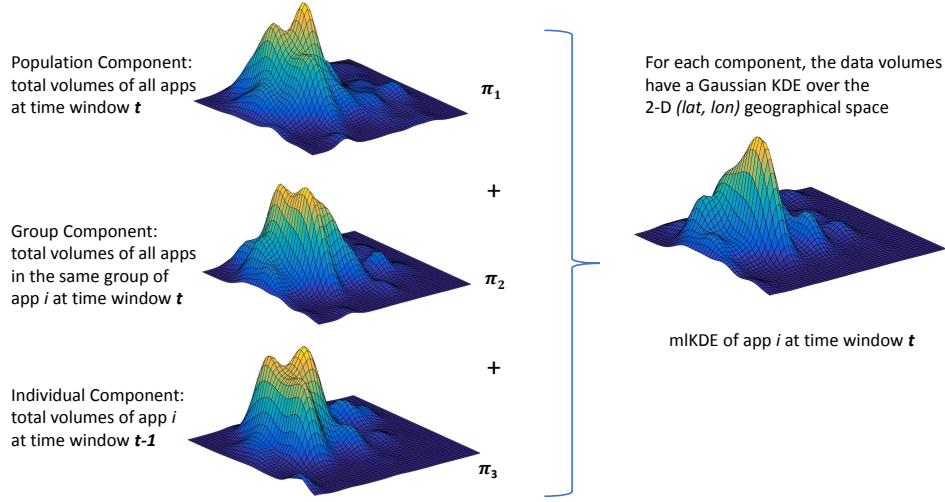


Figure 3.4: mlKDE to combine multi-granularity data.

Based on the weighted observations, we used an EM algorithm to train mixing weights for mlKDE, as shown in Algorithm 33.

Lines 1-6 initialize the mixing weights $\pi_1 = \pi_2 = \pi_3 = 1/3$ and use Equation 3.3 to calculate the likelihood L_{ij} of the weighted observation $(lat_i, lon_i)^T$ of Tower i for component j . Lines 7-33 train the mixing weights of mlKDE until they converge. Lines 8-23 is the expectation (E) step, which first computes Pr'_{ij} the probability that the weighted observation $(lat_i, lon_i)^T$ of tower i comes from the component j , then normalizes Pr'_{ij} as Pr''_{ij} , and finally calculates Pr_{ij} by considering the weight vol_i of each weighted observation $(lat_i, lon_i)^T$. Lines 24-32 is the maximization (M) step. Since the KDE is nonparametric given that the bandwidth h is fixed, no parameter needs to be updated for mlKDE except for the mixing weights:

$$\pi_j = \frac{\sum_{i=1}^n vol_i \cdot Pr_{ij}}{\sum_{i=1}^n vol_i} \quad (3.7)$$

where n is the total number of cell towers.

In this algorithm, the likelihood L_{ij} of the weighted observation $(lat_i, lon_i)^T$ of Tower i for component j is computed only once, greatly decreasing the computing time.

Algorithm 1: The EM Algorithm for the Training Mixing Weights of mlKDE

input : P_t : population data at time window t ; G_t : group data at time window t ; I_{t-1} : individual data at time window $t-1$; I_t : individual data at time window t ; T_i : weighted observation $(lat_i, lon_i)^T$ of tower i ; Let $V = \{P_t, G_t, I_{t-1}\}$, i.e., $V_1 = P_t, V_2 = G_t, V_3 = I_{t-1}$

output : $\pi = \{\pi_1, \pi_2, \pi_3\}$: non-negative mixing weights, i.e. weights of population component, group component and individual component, subject to $\pi_1 + \pi_2 + \pi_3 = 1$

```
1 foreach component  $j$  do
2    $\pi_j \leftarrow 1/3$ ;
3   foreach Tower  $i$  do
4      $L_{ij} \leftarrow f_{KDE}(T_i \mid V_j, h)$ ;
5   end
6 end
7 while  $\pi$  not converge do
8   // E step
9   foreach Tower  $i$  do
10     $sum\_pro \leftarrow 0$ ;
11    foreach component  $j$  do
12       $Pr'_{ij} \leftarrow L_{ij} * \pi_j$ ;
13       $sum\_pro \leftarrow sum\_pro + Pr'_{ij}$ ;
14    end
15    foreach component  $j$  do
16       $Pr''_{ij} \leftarrow Pr'_{ij} / sum\_pro$ ;
17    end
18    foreach Tower  $i$  do
19       $vol_i \leftarrow$  volumes of Tower  $i$  from  $I_t$ ;
20      foreach component  $j$  do
21         $Pr_{ij} \leftarrow Pr''_{ij} * vol_i$ ;
22      end
23    end
24    // M step
25    foreach component  $j$  do
26       $sum\_pro \leftarrow 0$ ;
27       $sum\_vol \leftarrow 0$ ;
28      foreach Tower  $i$  do
29         $sum\_pro \leftarrow sum\_pro + Pr_{ij}$ ;
30         $sum\_vol \leftarrow sum\_vol + vol_i$ ;
31      end
32       $\pi_j \leftarrow sum\_pro / sum\_vol$ ;
33    end
34 end
```

3.3 Experimental Results

In this section, we report experimental results based on real-world datasets. We provide an overview of the dataset, present the experimental setup, and discuss the insights and performance evaluation results.

3.3.1 Datasets and Experimental Setup

Our datasets consisted of more than 22 million records generated by mobile apps via 2G cell towers in 10 mid-sized cities in the same province of China. Each record included the tower ID, the app ID, data volumes, and the starting and ending time stamps. Generally, mobile network operators rely on the Deep Packet Inspection (DPI) technique, which can identify, filter and analyze data packets, to acquire the app ID from mobile data traffic [6, 24]. All data were collected from 6:00 pm to 9:00 pm on a Sunday evening in September 2014. The total download data volumes of all apps was over 220 GB and the number of unique users is 401,272. Table 3.1 shows summary statistics of the 10 cities.

All experiments were performed on a PC with a 3.30GHz Core i5 processor and 8GB memory, running Windows 7 Enterprise. The KDE and mlKDE were implemented in MATLAB R2014b. For mlKDE, we have tried various bandwidth values (0.01, 0.1 and 1) and time window sizes (1 minute, 3 minutes and 5 minutes). Due to space limitation, we set the bandwidth default as 1 and the time window size default as 1 minute in the rest of the experiments unless otherwise specified.

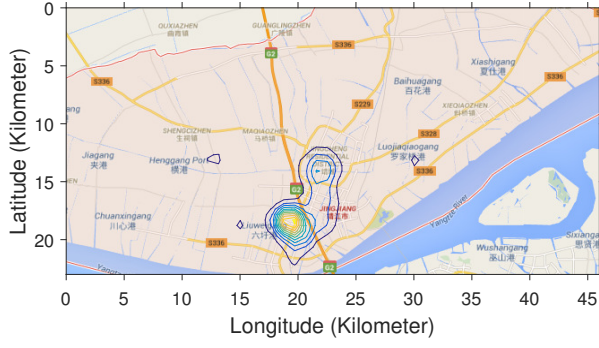
3.3.2 What Does mlKDE Visualization Look Like?

QQ is one of the ubiquitous instant messaging tools in China, which was developed by Tencent and has over 1 billion registered users worldwide. According to Tencent’s Q3 report in 2015 [110], the number of monthly active users of QQ on smart devices was 639 million. In this section, we use QQ as an example to demonstrate the effectiveness of our approach in profiling the spatial distribution of QQ’s data volumes in a timely fashion.

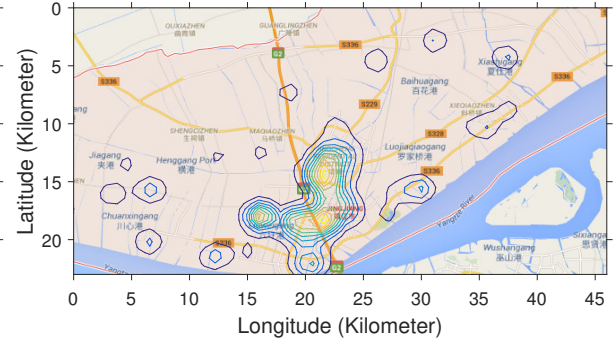
Example density distributions of QQ and the corresponding group and population are provided in Figure 3.5, which were estimated using Gaussian kernel density functions.

Table 3.1: Detailed overview information of the 10 cities.

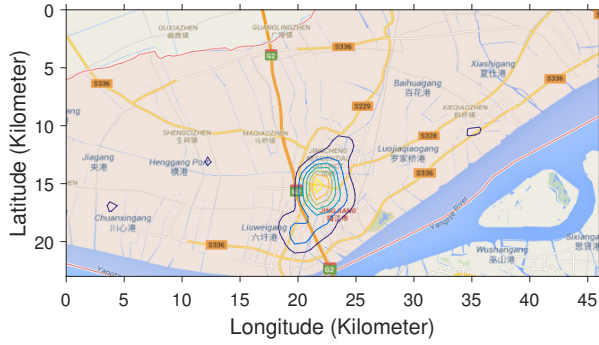
City	Area (km ²)	Population	# Towers	# Users	# Records	Volumes (GB)
A	1,915	1,200k	118	27,837	1,291,537	11.78
B	1,446	1,166k	122	31,816	2,040,977	20.71
C	1,439	1,108k	178	39,242	2,087,500	19.01
D	928	793k	264	39,849	2,238,362	23.35
E	1,160	804k	262	41,809	2,144,647	21.82
F	656	666k	498	100,844	5,937,397	61.73
G	1,349	1,286k	127	30,227	1,894,636	19.41
H	2,855	966k	175	34,352	1,733,018	16.74
I	1,461	614k	62	17,232	768,526	7.71
J	2,395	1,572k	206	38,064	2,285,537	23.36



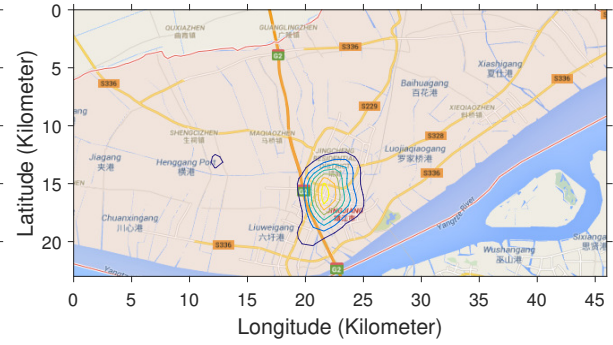
(a) Individual Level Density ($t = 49$)



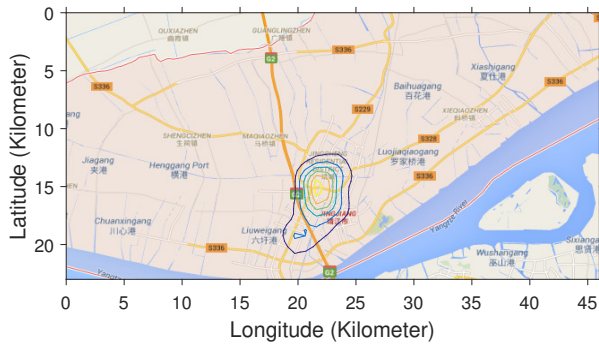
(b) Individual Level Density ($t = 50$)



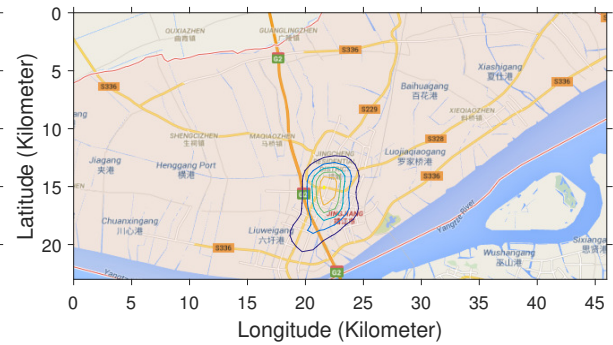
(c) Group Level Density ($t = 49$)



(d) Group Level Density ($t = 50$)



(e) Population Level Density ($t = 49$)

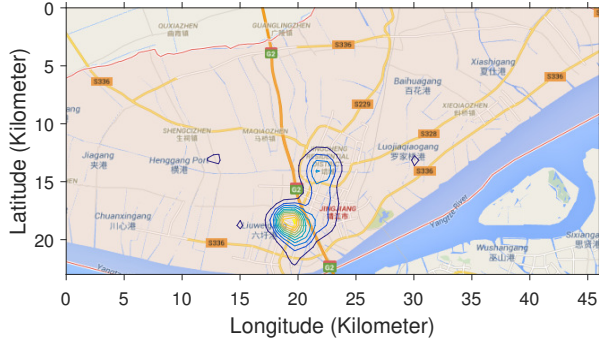


(f) Population Level Density ($t = 50$)

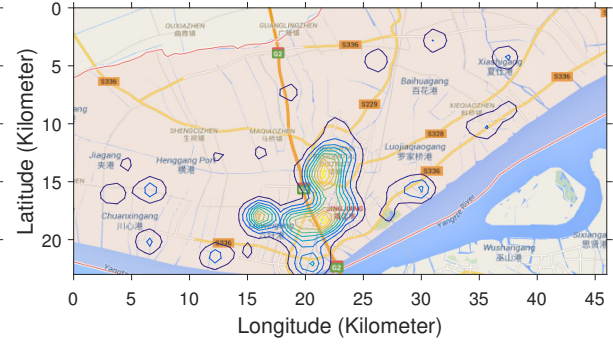
Figure 3.5: Actual density distributions of QQ at different granularity levels in city F .

The individual level density distribution at the 49th and 50th time window is shown in Figure 3.5(a) and Figure 3.5(b), respectively. Despite QQ being a very popular app that has large data volumes, it is evident that at the individual app level, the density distribution of consecutive time windows differ greatly. This indicates that if we focus on data at the individual level, the noise and variations will be large, which may make it hard to recognize the geospatial distribution patterns. In contrast, when observing at the group and population level, the densities for the same time windows exhibit much greater stability (see Figure 3.5(c) to Figure 3.5(f)).

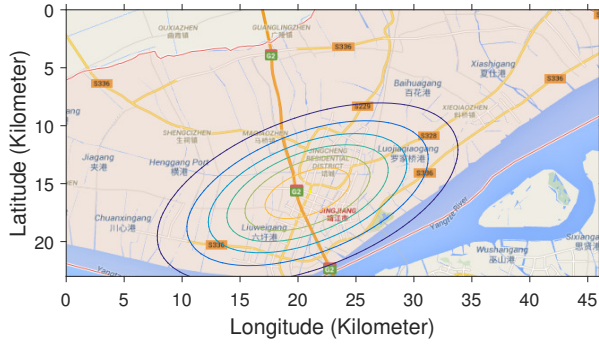
Based on data presented in Figure 3.5, Figure 3.6 demonstrates the individual level spatial density estimates using different approaches. Figure 3.6(a) and Figure 3.6(b) show the actual density distributions at the 49th and the 50th time window. The two figures depict density distributions flexibly, but they change dramatically from the 49th to the 50th time window. Figure 3.6(c) and Figure 3.6(d) describe the 2D Gaussian density distribution of individual level data at 49th and 50th time window respectively. Both of Figure 3.6(c) and Figure 3.6(d) have only one dense center, which implies they are too coarse and not flexible enough to describe the actual density distributions, even though the shapes of density distributions remain relatively stable as time changes. Compared with our proposed mlKDE, both the above 2D Gaussian and KDE do not take into account the group data and population data, Figure 3.6(e) and Figure 3.6(f) illustrate the density distributions using mlKDE. The mlKDE density distribution reveals two dense centers located at similar positions where the dense center found in KDE was located, which means they were able to depict density distributions in a flexible way. Additionally, the mlKDE density distributions remain relatively stable with advancing time, which estimates and profiles the geospatial distributions of individual apps in a robust manner. Furthermore, it demonstrates an easily recognizable pattern, regardless of whether the actual individual level distribution is too sparse or sporadic. This pattern provides valuable knowledge that mobile operators and app developers can employ to optimize bandwidth allocation, load balancing, and network caching.



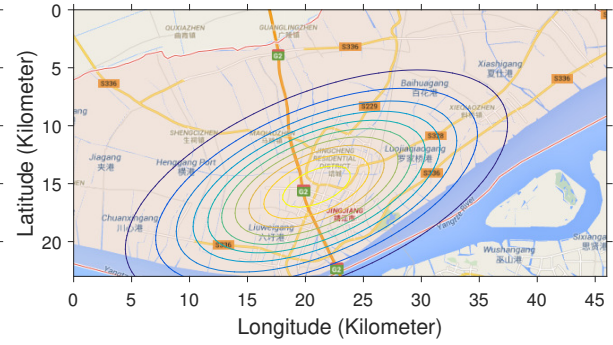
(a) Actual ($t = 49$)



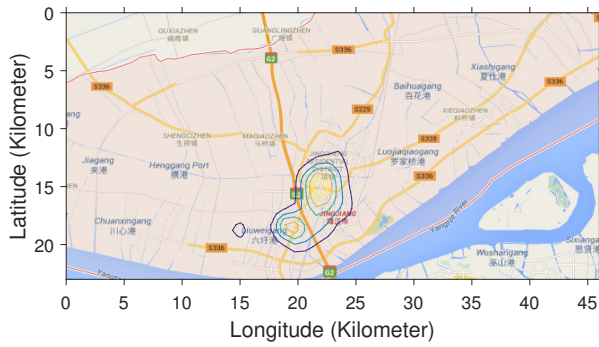
(b) Actual ($t = 50$)



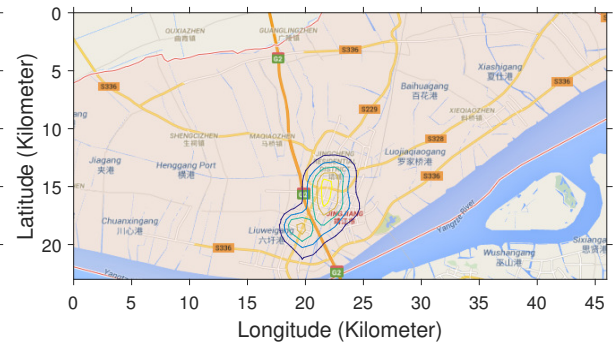
(c) Gaussian ($t = 49$)



(d) Gaussian ($t = 50$)



(e) mKDE ($t = 49$)



(f) mKDE ($t = 50$)

Figure 3.6: Density estimates of QQ in city F using different approaches.

3.3.3 How to Interpret Mixing Weights?

In this subsection, we examine the mixing weights trained by our model, focusing on the interpretations and implications. First, we show that the same app’s mixing weights are consistent across different cities. Second, we present the relative stability of the mixing weights of apps over time. Third, we provide examples of applying mixing weights to test the uniqueness of distinct apps.

Are Mixing Weights of Same Apps Similar in Different Cities?

We trained the mixing weights of myriad apps in 10 cities, and discovered that the mixing weight patterns of the same app in all these cities were very similar. For example, the average mixing weights of two apps, **WeChat** and **Momo**, are listed in Table 3.2. For **WeChat**, both the population and individual weights were quite low (both below 0.1) whereas the group weights were high (above 0.8). For **Momo**, the group weights were low (below 0.1) whereas the population and individual weights were higher (both between 0.3 and 0.7). The overall distributional pattern of mixing weights of the same app varied only slightly across the different cities, indicating that our approach did find results that are repeatable in different geographical areas.

Table 3.2: Mixing weights of **WeChat** and **Momo** in 10 cities.

City	WeChat			Momo		
	Population	Group	Individual	Population	Group	Individual
A	0.008	0.941	0.051	0.433	0.026	0.541
B	0.004	0.922	0.074	0.373	0.024	0.603
C	0.005	0.951	0.044	0.417	0.018	0.565
D	0.005	0.942	0.053	0.375	0.020	0.604
E	0.001	0.979	0.020	0.417	0.019	0.564
F	0.001	0.970	0.029	0.362	0.014	0.624
G	0.010	0.919	0.071	0.378	0.031	0.591
H	0.005	0.939	0.056	0.377	0.013	0.610
I	0.025	0.890	0.085	0.363	0.032	0.605
J	0.007	0.925	0.068	0.378	0.014	0.608
Avg.	0.007	0.938	0.055	0.387	0.021	0.592
S.D.	0.007	0.026	0.020	0.025	0.007	0.026

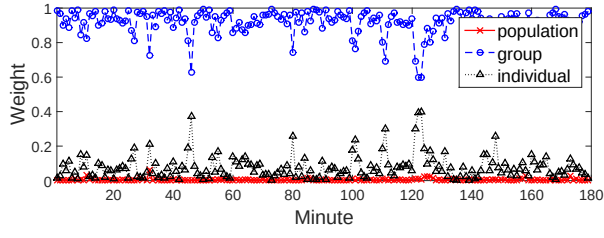
Are Mixing Weights Stable Over Time?

Apps in the same group have certain common characteristics, but can also possess their own independent features. We picked two instant messaging apps, **WeChat** and **Momo**, to demonstrate and analyze their different mixing weights. Table 3.2 shows the difference in average mixing weights between these two apps.

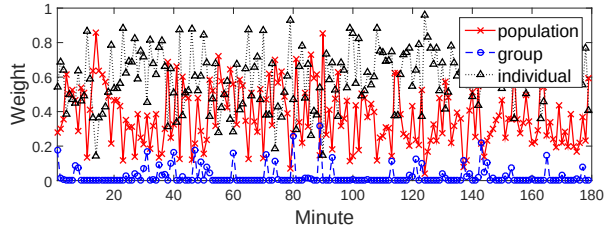
The dynamic mixing weights of **WeChat** and **Momo** from 6:00 pm to 9:00 pm in city B are shown in Figure 3.7. We can easily see differences in their mixing weights. **WeChat** has a heavy group weight dominating the weights of both population and individual, which perhaps can be explained by the fact that **WeChat** is the most popular instant messaging app and its data volumes account for more than 64.6% of the total instant messaging data volumes in our data. Due to its market dominance, the **WeChat** usage behaviors can represent all instant messaging apps to a certain degree. However, for **Momo**, the individual weight is largest followed by weights of population and group components. Unlike **WeChat** and most other instant messaging apps connecting familiar people, **Momo** is uniquely aimed at matching people for chats and dates. It is **Momo**'s uniqueness that produced the close to zero weights of the group component.

How Do Mixing Weights Identify Uniqueness of Apps?

Generally, there are strong discriminations between the weights of different apps. What if app A was used to estimate the data volume density distribution at each granularity level (using KDE) and train the weights among these levels using app B's data (i.e., by calculating the likelihood of observing app B's data given app A's density distributions)? Intuitively, the weight of the individual component would normally be very low, because it was designed to capture the unique app-level characteristics. However, the weights of the group and population components depend on the similarity of the apps' groups. If app A and app B belong to different groups, the weight of the group component would be low (due to low group similarity) and the weight of the population component would be high. An example is shown in Figure 3.8(a), where the **Appstore** data were used to estimate the distributions at each level and the **WeChat** data were used to train the weights. On the

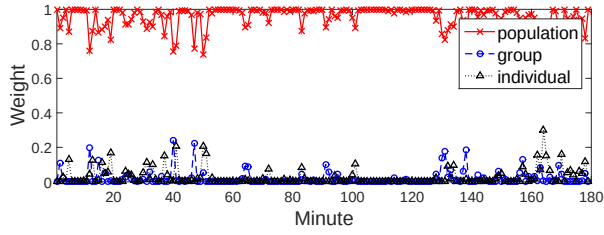


(a) WeChat

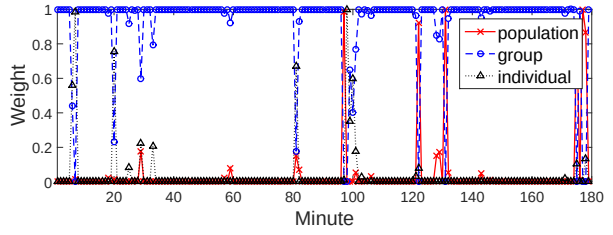


(b) Momo

Figure 3.7: Mixing weights from 6:00 pm to 9:00 pm in city B.



(a) Two Distinct Apps in Different Groups



(b) Two Distinct Apps in the Same Group

Figure 3.8: Identifying the Uniqueness of Apps based on Mixing Weights

contrary, if app A and app B belong to the same group, the weight of the group component would be high and the weight of the population component would be low. An example is shown in Figure 3.8(b), where the `QQMusic` data were used to estimate the distributions at each level and the `KugouMusic` data were used to train the weights. This finding can attest to the veracity of our mlKDE approach.

Using the mlKDE method, we can analyze the uniqueness of apps, which is important information when developers make policies when designing and developing apps. In general, one app will be more similar with apps in the same group than with apps classified in different groups. However, in some cases, the temporal and spatial density distributions of some apps will be much different from other apps in the same group. We call these apps “singular” apps. One example of a singular app is `SkyMap`, cataloged in the group of “maps.” `SkyMap` is unique in that its users typically access the app at night when they are located outdoors. Other map apps, such as `GoogleMap` and `Navfree`, are heavily used both during the daytime and the nighttime.

3.3.4 How Much Does mlKDE Perform Better than KDE?

We compared the performances of the KDE and the mlKDE quantitatively. Intuitively, mlKDE is more robust and flexible than KDE, because mlKDE leverages aggregate information from group and population data, which enables it to better handle apps with sparse individual data. We evaluated the performances of the KDE and the mlKDE in the 10 cities with different parameters. Figure 3.9 shows the average log likelihood of KDE and the mlKDE. We found that average log likelihood over all apps of mlKDE outperforms KDE in nearly all cities with different bandwidths. With the decrease of bandwidths from 1 to 0.01, the performances of mlKDE are slightly enhanced while that of KDE are weakened dramatically. This demonstrated the robustness of our approach and unstableness of KDE in terms of bandwidth.

For mlKDE, a smaller bandwidth leads to a higher accuracy, because a small enough bandwidth will treat each cell tower located at fixed positions as distinct peak, which is consistent with the characteristics of our data. For KDE, we used the data of previous time window to estimate the distribution, and the data of current time window to calculate

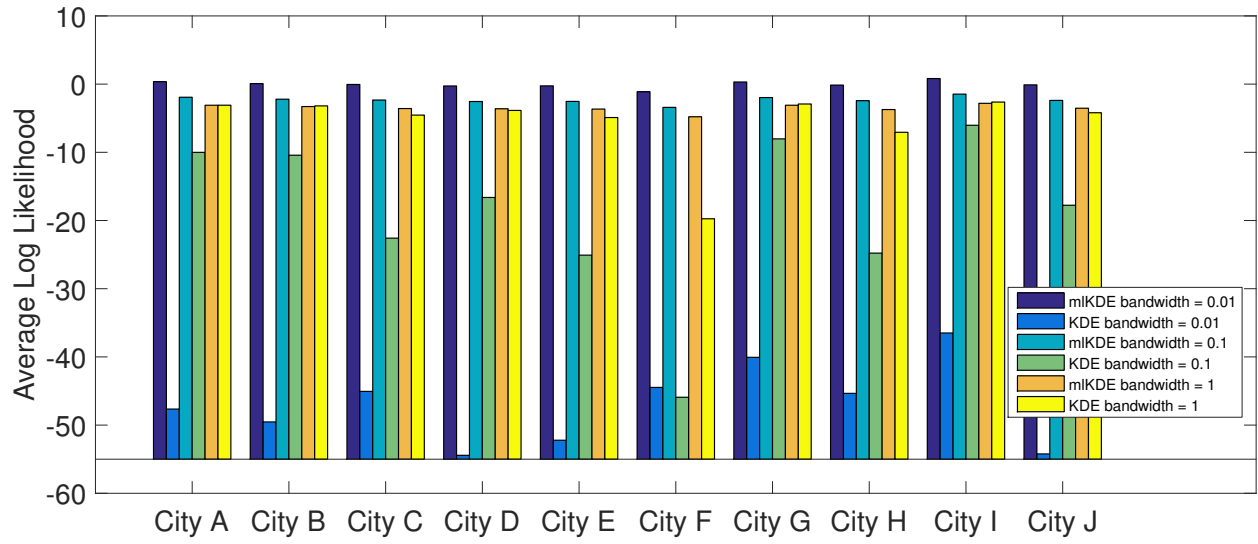


Figure 3.9: Average log likelihood over all apps.

average log likelihood. Since the data density distributions of an app may vary greatly between two continuous time windows, a smaller bandwidth depicting the previous time window data with a higher accuracy would not guarantee a good description of current time window data (see Figure 3.6(a) and Figure 3.6(b)). Therefore, the average log likelihood would be very low when the bandwidth is set as a small value. As the bandwidth becomes larger, the KDE describes the data of previous time window more coarsely, but it is a better fit for the data of current time window, so the average log likelihood rises.

The experimental results presented in this subsection are based on the time granularity of 1 minute (i.e. the time window length is 1 minute). Consistent results were found for different time granularity of 3 minutes, 5 minutes and 10 minutes.

3.3.5 Are Mixing Weights Sensitive to Bandwidths?

The mixing weights of different components greatly impact the performance evaluations of mlKDE. In our experiments, we tested different values of bandwidths and found that the smaller the bandwidth, the better the performance. However, when the bandwidth becomes small enough, the mixing weights of mlKDE are minimally sensitive to different bandwidths. Figure 3.10 shows the weights of individual, group and population component of WeChat in City B, using bandwidths 0.01, 0.1 and 1 respectively. Data with bandwidths of 0.01 and 0.1 have almost identical weights for all components. The weights of data with a bandwidth of 1 is slightly different with the other two values at several time points, but all three values have the same trend. We also tested other apps and found the same result. We can conclude that mixing weights is not sensitive to bandwidths when bandwidths are adequately small.

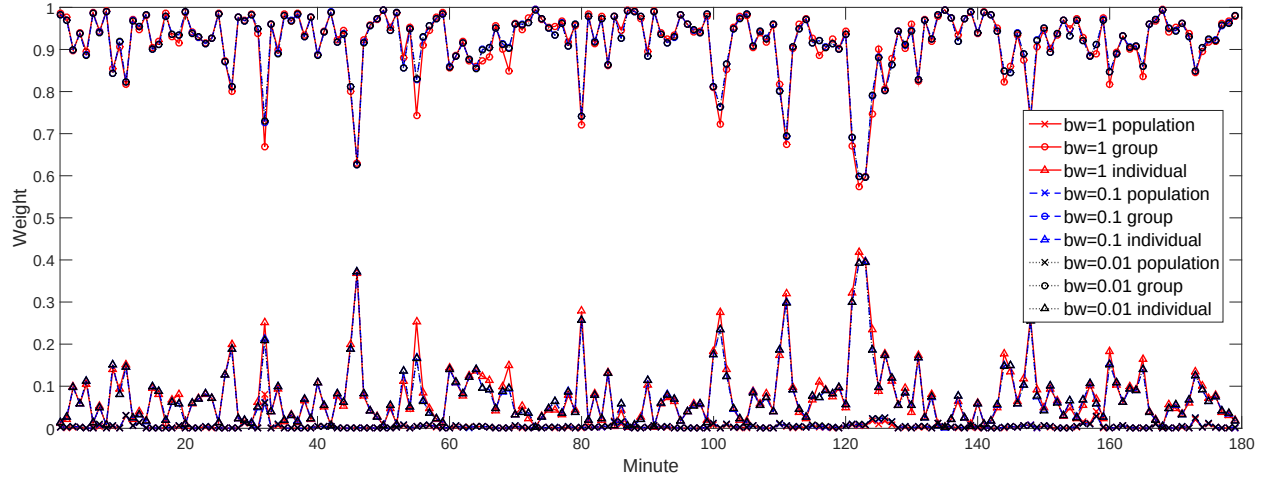


Figure 3.10: Mixing weights of WeChat with different bandwidths in city B.

Chapter 4

Investigating Shared Locations On Social Networks

In this chapter, we take Twitter as a lens to investigate the publicly shared locations on social networks. We first introduce Twitter location types and the methodologies to infer locations given free-text descriptions. Then we use four case studies to demonstrate how Twitter location data facilitate the research of human mobility, online petitions, transportation, and disease propagations. Note that this chapter is partially based on my published papers:

Y. Feng, Z. Lu, Z. Zheng, P. Sun, W. Zhou, R. Huang, and Q. Cao. Chasing Total Solar Eclipses on Twitter: Big Social Data Analytics in Once-in-a-lifetime Events. IEEE Global Communications Conference (GLOBECOM), 2019

Y. Feng, W. Zhou, Z. Lu, Z. Wang, and Q. Cao. The World Wants Mangoes and Kangaroos: A Study of New Emoji Requests Based on Thirty Million Tweets. The Web Conference (formerly known as WWW), 2019.

Y. Feng, Z. Lu, W. Zhou, Z. Wang, and Q. Cao. New Emoji Requests from Twitter Users: When, Where, Why, and What We Can Do About Them. ACM Transactions on Social Computing (TSC) 3(2), 2020.

4.1 Twitter Location Types

4.1.1 Twitter User Profile Location

The profile location refers to the residential address where Twitter users specified in their public account profiles. Since Twitter users are allowed to use free-form text to describe profile locations, it is challenging to extract precise geographical information from such messy, flexible and diverse profile locations using rule-based approaches. Instead, we resort to Nominatim [87], a search engine for OpenStreetMap data, to infer the most likely locations by submitting queries via Nominatim online APIs. Places at different levels of detail (e.g., city, county, and street) are fetched through Nominatim depending on the address precision in the user profile. For example, the exact latitude/longitude point coordinate of a coffee house titled “Precision Pours” at Louisville, Colorado can be determined by querying the detailed location of “1030 E South Boulder Rd”. Note that some account locations, such as “17th & Fountain”, “Anywhere, Earth”, and “From old houses, to old towns”, are too casual to be geo-referenced.

4.1.2 Tweet Location

When posting tweets, Twitter users are allowed to geo-tag their tweets, publishing their real-time locations. Tweet locations can be either a box polygon of coordinates defining general areas like cities and neighborhoods, or an exact GPS latitude and longitude coordinate. Both types of tweet locations contain high-level geography information, such as the associated country and state, making it possible to explore the overview of national and statewide geographic patterns. The geo-tagged states can also infer local time zones that had been removed from public tweets since May 2018, enabling the investigation of local hourly tweeting behaviors.

As tweet GPS locations identify exact coordinates, they contain more detailed location information such as the associated county. Similar to parsing user profile locations, we leverage Nominatim [87] to retrieve the corresponding detailed places, e.g., the county and

street, for GPS data. Then GPS locations are aggregated together based on the extracted places to discover fine-grained geographic patterns.

4.2 Human Mobility During the 2017 Great American Eclipse

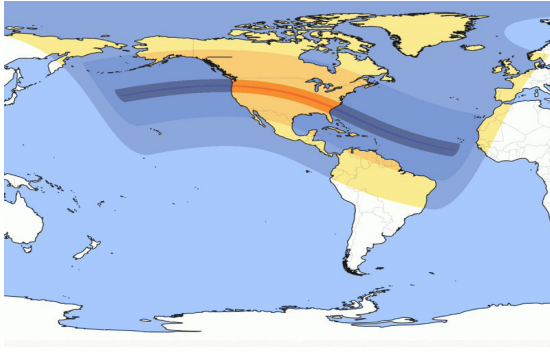
4.2.1 Background and Tweet Collection

A natural phenomenon of *The 2017 Great American Eclipse* happened on Aug 21, 2017 (see Figure 4.1) offered a perfect chance to study how the extremely rare event influenced the people’s life in large scale. This coast-to-coast total solar eclipse crossed the continental United States (U.S.) for the first time since 1918, and the next one with a similar path will occur in 2024. This rare and spectacular event attracted wide attention from the entire continent. According to The Washington Post [44], nine in ten adults in the U.S. watched this total eclipse. In addition, the eclipse related topics went viral on social networks before, during, and after this event.

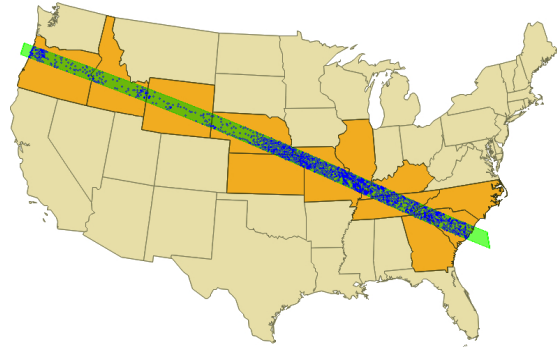
To chase this total solar eclipse on social networks, we used Twitter’s Streaming APIs, which enable developers to filter and collect real-time tweets, to crawl the English tweets containing the word “eclipse” from the entire duration of the event (from Aug. 20, 2017 to Aug. 24, 2017). The collected tweets were formatted in JavaScript Object Notation (JSON) files with named attributes and associated values [116]. After removing bots, we put together 5.97 million tweets including original tweets, replies, retweets, and quoted tweets, and 1.17 million unique tweets after deleting retweets.

4.2.2 Geographic Distribution Overview

Although this solar eclipse’s path of totality was mainly across the U.S., it attracted worldwide attention. Figure 4.2(a) shows that the tweets mentioning the English word “eclipse” come from 168 countries, even including non-English speaking countries like China and Japan. It is not surprising that a majority (89.5%) of collected tweets come from the

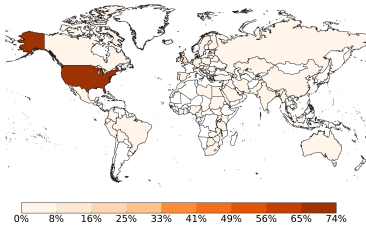


(a) Across the world [112]

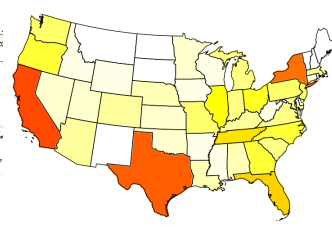


(b) Across the U.S. [56]

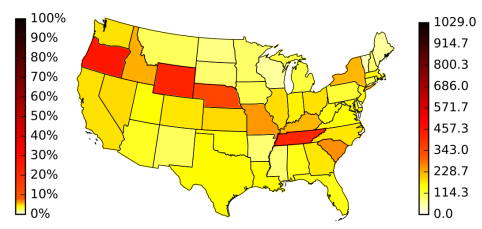
Figure 4.1: Path of the eclipse shadow across the world and United States. The narrow tracks are called the path of totality. Small blue dots inside the narrow track in (b) refer to U.S. cities. The highlighted states in (b) represent areas where total solar eclipse was visible.



(a) World distribution



(b) Percentage by state



(c) Normalized by POP

Figure 4.2: Worldwide and national spatial distribution of tweets. (a) The U.S. contributes more than 89% tweets. (b) States with a large population contribute more. (c) States along the path of totality tweet more per million people.

United States. Canada (3.4%) ranks as the second and was followed by Britain (2.1%). Any other country except the above three ones accounts for less than 0.4%.

Therefore, we focused on the U.S. to study the spatial distribution of eclipse-tagged tweets at the national level. Figure 4.2(b) illustrates the states with a large population, such as California, Texas, and New York, contribute most tweets. It is impressive that, after normalizing the number of tweets by the resident population per state [93], those states along the solar eclipse’s path of totality became dominating, which indicates that on average people staying close to the path are more likely to share this event on Twitter.

4.2.3 Interstate and Intrastate Human Movement

During *The 2017 Great American Eclipse*, it was reported millions of people traveled to chase this natural event in the path of totality. We analyzed human travel behaviors based on two types of Twitter geographical information, namely, profile locations and tweet locations. Figure 4.3 shows the distribution of tweets geo-tagged with exact latitude and longitude coordinates. In our dataset, most of such tweets are posted either along the eclipse’s path of totality, or from densely populated largest cities such as New York City, Boston, Washington, D.C., Seattle, San Francisco, and Los Angeles. The dense distributions in less populated states such as Idaho, Wyoming, and Missouri inspired us to study how people travel between states. On the other hand, the overcrowded distributions in Tennessee, Oregon, South Carolina, and Georgia where the path of totality crossed motivated us to look deeper into people’s movements inside states.

Interstate Movement

We treated Twitter users’ profile locations as their places of residence. If Twitter users posted eclipse-related tweets outside the state of residence, we assumed they traveled between states to chase the eclipse. Table 4.1 illustrates the 20 most popular interstate trips for chasing eclipse. The destinations of red trips are states where the total solar eclipse occurred. On the contrary, the blue ones represent cross-state travel towards states where the only partial eclipse was visible. It is interesting to note that the top three blue trajectories, i.e., New York

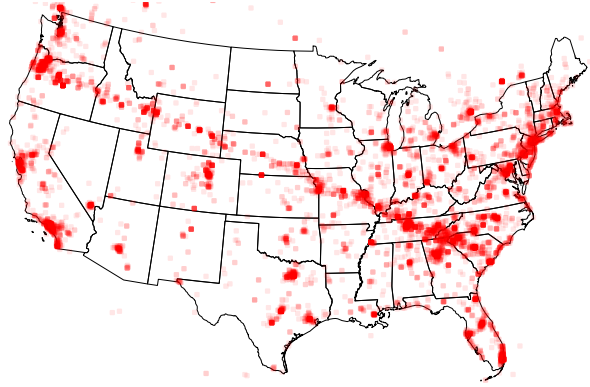


Figure 4.3: Distribution of tweets with exact latitude and longitude coordinates.

Table 4.1: The 20 most frequent Interstate movements

From	To	Count	From	To	Count
California	Oregon	252	Pennsylvania	New Jersey	71
Washington	Oregon	176	D.C.	Washington	66
North Carolina	S. Carolina	131	California	Texas	66
New York	New Jersey	122	Florida	Georgia	62
D.C.	Virginia	101	Missouri	Kansas	60
D.C.	Maryland	84	New York	S. Carolina	57
Florida	S. Carolina	80	New York	Florida	55
New York	California	79	New York	Tennessee	54
Colorado	Wyoming	78	Georgia	S. Carolina	52
Utah	Idaho	77	Texas	Tennessee	52

$\rightarrow$ New Jersey, D.C. $\rightarrow$ Virginia, and D.C. $\rightarrow$ Maryland, are more likely to be generated by commuters because these pairs of states are so close to each other that people may even live and work in two different states.

We then visualized the 50 most frequent interstate trips, as shown in Figure 4.4. The red arrows represent the trajectories to states through which the eclipse’s path of totality crossed. The blue arrows are the trajectories to states where the total eclipse did not occur. The width of the arrows indicates the frequencies of travels. The red arrows dominate the blue ones greatly, suggesting people did prefer states where the totality can be observed. However, the red coast-to-coast movements are less than the blue ones, which infers that most people probably chase the eclipse in nearby states instead of far-away states.

Finally, we summarized all 10,334 interstate trips of Twitter users during the event with respect to destination states, origin states, and interstate travel likelihoods. Figure 4.5(a) shows that all of the three most attractive destinations, i.e., Oregon, South Carolina, and Tennessee, are located in the path of totality, where people can experience the totality. The eight most popular origin states except Washington, D.C. have the largest populations, as demonstrated in Figure 4.5(b). However, after normalizing the number of interstate trips by state populations, we find the two most active origin states are Delaware and Wyoming which have the seventh least and the first least population among all states (see the number of people taking interstate trips per million people in Figure 4.5(c)). In addition, it seems that people living in the northern parts of the U.S. made more frequent interstate trips on average during this event.

Intrastate Movement

To further investigate the chasing behaviors at a detailed level, we select four states, i.e., Tennessee, Oregon, South Carolina, and Georgia along the eclipse totality path to study human intrastate movements. As shown in Figure 4.6, people also sought to find a suitable place to watch the eclipse inside states for convenience. In all of the four states, most of the starting locations are from major cities. To be more specific, Twitter users living in Nashville, Knoxville, Memphis and Chattanooga in Tennessee, Portland and Eugene in Oregon, Atlanta in Georgia, and Columbia, Charleston, and Greenville in South Carolina,

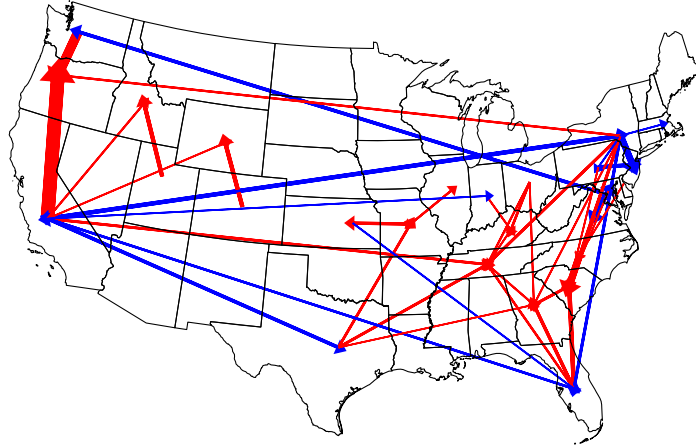


Figure 4.4: The 50 most popular interstate trips during the event.

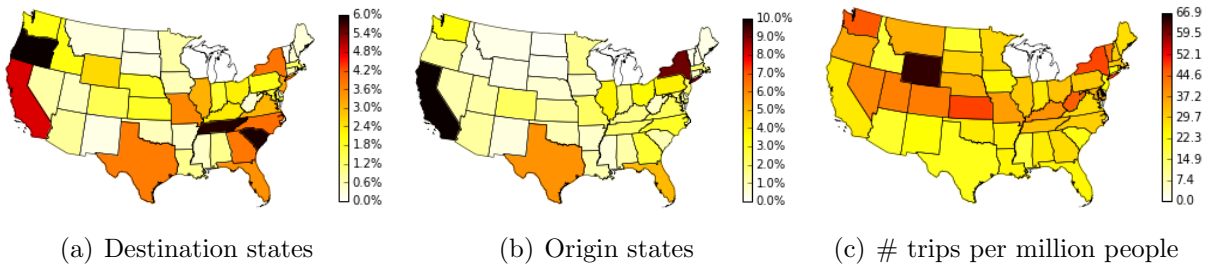


Figure 4.5: Interstate movement

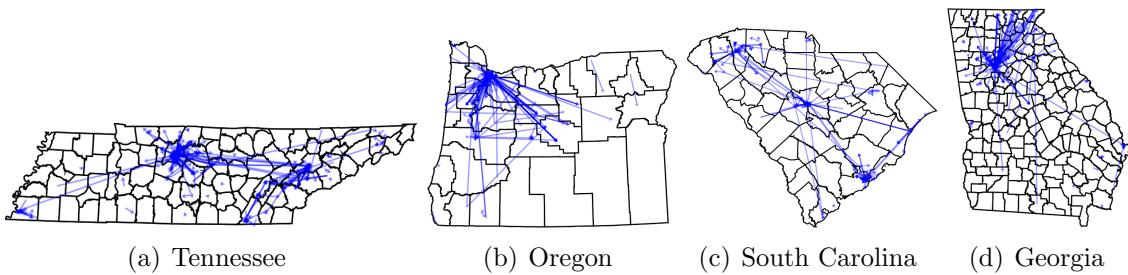


Figure 4.6: Intrastate movement

were more likely to take a trip to chase the eclipse. Directions of intrastate movements are generally towards the path of totality. For example, Twitter users in Atlanta trended to move to the northeast board of Georgia where the total eclipse occurred. It should be noted that moving ranges of all three major cities in South Carolina are much smaller than cities in other states, which could be explained by the fact that all of the three cities were located in the path of totality.

4.3 Geographic Patterns of Petitions for New Emojis on Twitter

4.3.1 Background and Tweet Collection

As the usage of emojis (and social media in general) evolves, new emojis are being continuously requested. According to the Emogi Research Team [109], 75% of mobile messaging app users were interested in having more emoji options, and this demand is more intense for those who more frequently use mobile messaging. Every year, the Unicode Consortium¹ updates the official list of Unicode emojis by judging and accepting proposals for new emojis from individuals, organizations and companies. For each candidate emoji, its evidence of frequency from Google Search, Bing Search, YouTube Search, and Google Trends must be submitted, and evidences from NGram Viewer and Wikipedia Search are optional. Besides the substantial efforts needed to collect such evidences, this method has several additional drawbacks. First, objects with higher search frequency does not always imply better fit as emojis. For example, although the word “mascot” is heavily searched, it is unlikely to be an emoji because there exists no specific image that can represent all mascots for different teams, events, organizations and universities. Second, this method completely ignores new emoji petitions directly generated by actual users, whereas users have first-hand information regarding the usage context and could contribute tremendously to generating valuable ideas. These users do not necessarily know how or have the time to make an official

¹<https://unicode.org/emoji/>

request. For example, Figure 4.7 shows a tweet, in which the poster comments that the daisy emoji does not exist and wonders how to get one.

To more comprehensively study the emoji requests, we investigate more than thirty million tweets mentioning the word “emoji,” collected from Oct. 2017 to Oct. 2018. First, we extract the requested emoji descriptions and calculate their corresponding frequencies. After filtering out emojis that already existed (i.e., extant emoji requests) and exploring why they were still being requested, we propose a WordNet [82] based emoji classifier to cluster requested emojis. Then, we study spatiotemporal patterns of these requests and explore possible reasons why new emojis were requested. In addition, we illustrate the existing relatedness, fairness, and equality problems reflected through emojis, and discuss the positive impacts of new emojis on society. Finally, we develop a web-based interactive emoji petition tracker — <https://call4emoji.org>, which allows users to analyze petitioned emojis in a real-time manner. In the following sections, we mainly focus on the geographic patterns of new emoji petitions.

4.3.2 Geographic Patterns

In our collected dataset, 2.8% tweets have geo tags. We use these geo-tagged tweets to profile geographic distributions of new emoji requests at both worldwide and national levels. The worldwide distribution of expected emojis is illustrated in Figure 4.8(a), where people in as many as 110 different countries petition for new emojis. As we collect tweets written in English, English-speaking countries, such as United States (73.6%), United Kingdom (10.9%), and Canada (3.2%), contribute the most of emoji requests. It is interesting that non-native English-speaking countries, such as China, Japan, Brazil and Mexico, also express their desires for new emojis even in English, which might be one evidence of the world’s passion for emojis.

Since the most requests were made in the United States, we then focus on the United States to explore the geographic distribution of emoji requests at the national level. Figure 4.8(b) shows the proportion of new emoji requests per state to the overall number of nation-wide emoji requests. As expected, states like California, Texas, New York and Florida make a large number of requests, while those states lying at the heartland have low



Follow

Why is there no daisy emoji? What's a girl to do to get one? It would like nice next to my name, no tea just facts

Figure 4.7: A sample emoji request

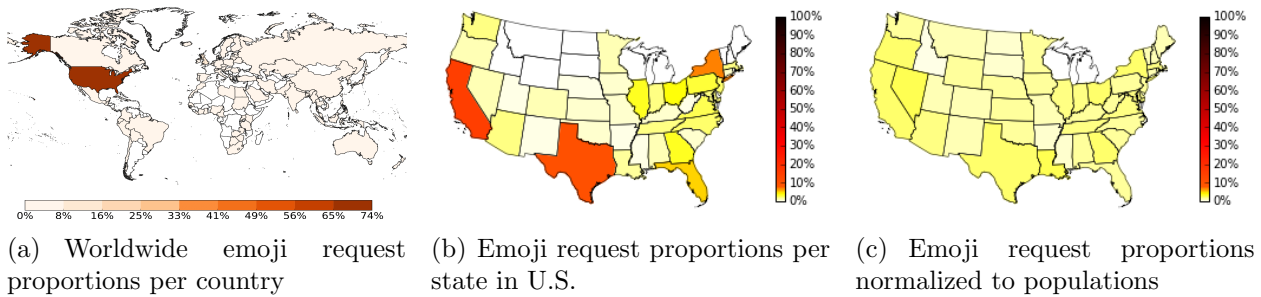


Figure 4.8: Geographic distributions. (a) English emoji requests are made from 110 countries, and the United States, United Kingdom, and Canada account for more than 80% emoji requests. (b) States with large populations request more emojis than those having small populations. (c) After normalizing requests using the state-wise population, states show similar patterns.

requesting percentages. We think this uneven distribution is mainly caused by the different populations in these regions. After normalizing by state population [93], the geographical distribution as shown in Figure 4.8(c) is relatively smooth and even across the country, which indicates that people in different states have a similar level of desire for new emojis. Note that we also conduct the national geographic distribution analysis using Twitter users’ profile locations, which are the residential locations specified in their public account profiles, and obtain similar results.

4.3.3 Place-Related Emoji Petitions

When analyzing the context of emoji requests, we found the location was one of the factors motivating people to call for new emojis. Places of interests at different levels, like a single landmark, tourist attractions, and even regions or countries, may encourage users to seek new place-related emojis. Many Twitter users visiting Paris claimed for an Eiffel Tower emoji, like “Paris first though!! why’s there no Eiffel Tower emoji?!”. Similarly, the Mickey and Minnie emoji was requested at Walt Disney World (WDW) Resort like “Guess where I am?!!! WDW (why is there no Mickey and Minnie emoji?!)”. People living in Hawaii and Texas asked for their state flag emojis respectively.

4.4 Geographic Analyses on Shared Dockless Electric Scooter Deployment

4.4.1 Background and Tweet Collection

Micromobility is an emerging term usually referring to the usage of docked and dockless lightweight devices (e.g., bikes) for short- and medium-length trips. As a new mode of micromobility, shared dockless electric scooters (e-scooters) are gaining popularity in recent years. A recent survey conducted in February 2019 showed 11% of Paris residents reported using e-scooters either frequently or from time to time [85]. Aiming at closing first- and last-mile transit gaps for residents, many ridesharing companies, such as *Lime*, *Bird*, and *Lyft*, deployed thousands of e-scooters in more than 60 cities across the United States. According

to the National Association of City Transportation Officials (NACTO) [86], people took 38.5 million trips on shared e-scooters in 2018.

Smartphones are one of the key enablers of e-scooter sharing service. To ride a shared e-scooter, users must download e-scooter apps, sign up, input payment information, and scan a QR code to unlock the e-scooter. On e-scooter apps, riders can check ready-to-go e-scooters parked nearby. After finishing the trip, riders make a payment on the app, and the e-scooter is locked automatically. However, most existing e-scooter studies ignored the feature that e-scooters must be operated through smartphones, leading to a missing research perspective from the riders’ comments shared via smartphones, especially on social networks.

We chose to use Twitter as our lens to examine shared e-scooters comprehensively through big social data analytics. Specifically, we monitored and tracked 5.8 million English tweets mentioning the word “scooter” or the scooter emoji 🛴 via the Twitter Streaming APIs in a real-time manner from October 6, 2018 to March 14, 2020. After cleaning data, we presented an overview of temporal (both monthly and hourly) and geospatial tweet distributions.

4.4.2 Geo-enabled Temporal Distribution

We profiled the temporal distributions of e-scooter related tweets using two time granularities, i.e., by month and by hour. Figure 4.9(a) and 4.9(b) illustrate the monthly percentage of posted tweets by different countries. Followed by New Zealand, United Kingdom, Canada, and Australia, the United States accounts for more than 82% of all collected tweets. As shown in Figure 4.9(a), months in summer contributed the highest monthly data volumes in the United States in 2019. There is a significant drop when comparing data volumes in Nov. and Dec. 2019 with that in Nov. and Dec. 2018. We think such a drop may be caused by two reasons. First, e-scooter users are more likely to discuss their riding experience online when trying e-scooters for the first time. Second, strict e-scooter regulations and policies, such as limiting the number of companies authorized to operate scooters in each city, had been imposed in many U.S. cities in 2019.

Besides the United States, we investigated the monthly data distributions of other seven countries in Figure 4.9(b). The two peaks in New Zealand and Canada occur in local summer months, which may indicate that e-scooters are used more frequently during summer.

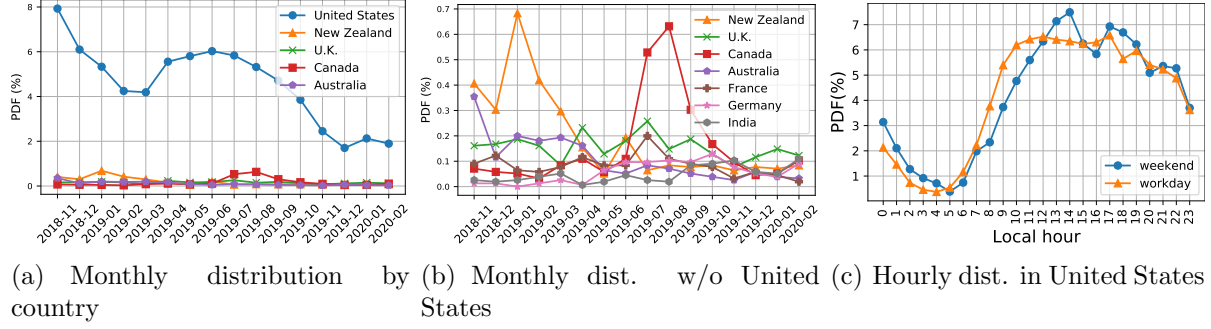


Figure 4.9: Time distribution from Oct. 6, 2018 to Mar. 14, 2020. The data of Oct. 2018 and Mar. 2020 are not plotted in (a) and (b) due to the data incompleteness.

Together with New Zealand, Australia shows a decreasing trend regarding the monthly data volume. On the contrary, Germany and India demonstrate an increasing trend. The amount of e-scooter tweets posted per month from United Kingdom and France are relatively stable.

We also explored the hourly tweet distributions regarding both workdays and weekends in the United States, as shown in Figure 4.9(c). As we expected, the tweet amount on every day is lower between 0:00 am and 7:00 am than that in the daytime. The most active time during weekdays is between 10:00 am to 17:00 pm (see the orange line). However, the peak time on weekends is between 12:00 pm and 19:00 pm (see the blue line). One possible reason is that many riders tend to start their outdoor activities later on Saturday and Sunday.

4.4.3 E-Scooter Deployment in the United States

We selected the United States as an example to study the geospatial distribution of e-scooter related tweets at the state level. The percentage of tweets posted from each state in the United States is demonstrated in Figure 4.10(a), where California (28.8%) and Texas (11.7%) account for more than 40% of all tweets. We also noticed six (CA, TX, GA, FL, NC, OH) out of the top ten states with the highest percentages were among the ten most populous states. After normalizing by state population, we obtained a relatively smooth

distribution, as shown in Figure 4.10(b). It is interesting to note that Washington D.C., one of the least populated states, generated the highest number of tweets per million residents. McKenzie [81] reported that scooter-share trips in Washington D.C. supported leisure, recreation, or tourism activities more than commuting, which may explain our findings.

We leveraged tweets geo-tagged with precise GPS (lat, lon) coordinates to explore the scooter deployment in cities. Within the United States, we collected 3359 exact GPS coordinates located in 579 cities. The exact GPS coordinates are demonstrated in Figure 4.11(a), where a deeper color indicates a higher GPS data density. Figure 4.11(b) shows the GPS data distribution aggregated by county. From the two figures, we can see that most extensive scooter deployments occurred in large cities at East Coast and West Coast, and other metropolises of the United States. Three cities in California contributed more than 15.6% of all GPS-tagged tweets – Los Angeles with a proportion of 7.9%, San Francisco (4.2%), and San Diego (3.5%). Washington D.C.(2.1%), New York (2.1%), and Miami (1.5%) ranked as the top three in the east coast cities. In addition, scooters were also very popular in metropolises including Chicago (2.6%), Austin (2.4%), Nashville (2.3%), Atlanta (2.2%), Dallas (1.7%), and Denver (1.5%).

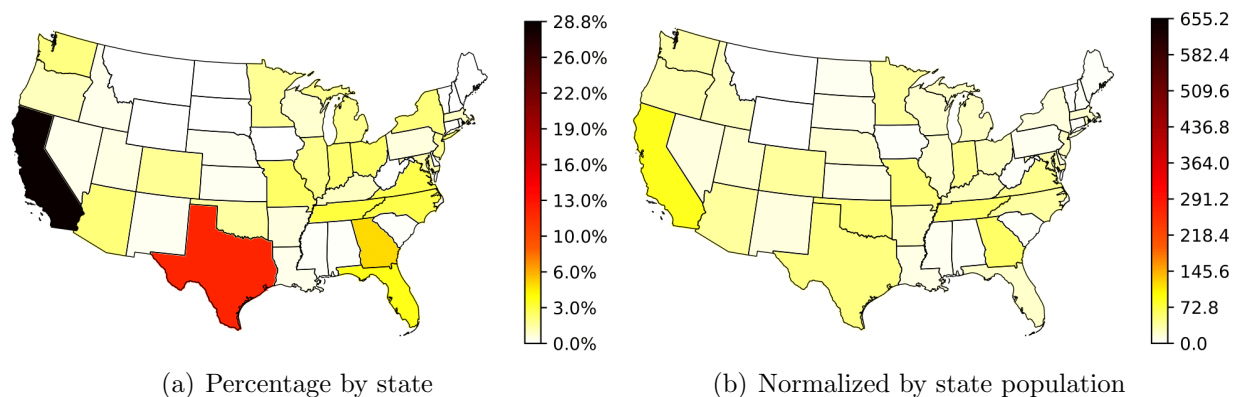


Figure 4.10: Geospatial distribution across United States

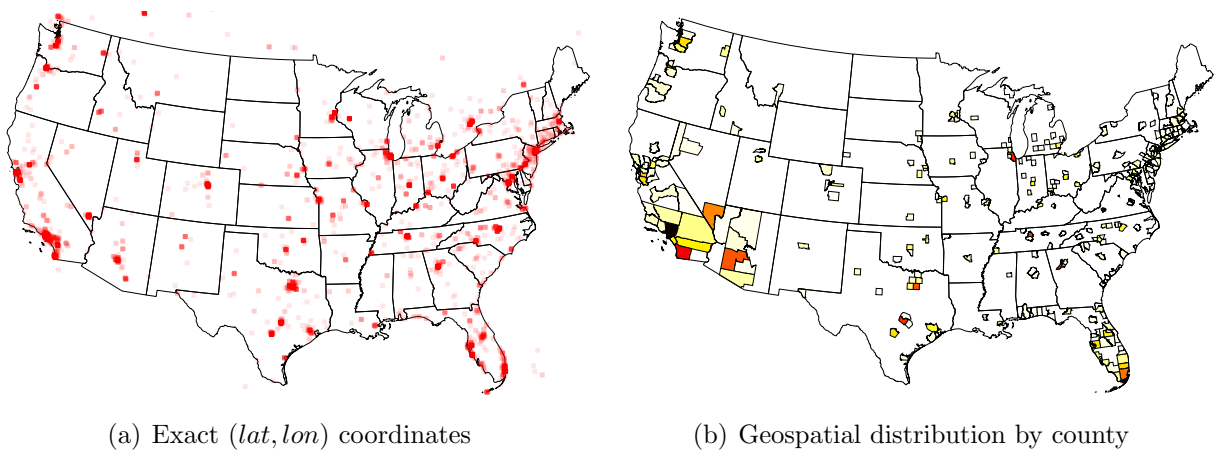


Figure 4.11: Distribution of scooter tweets with exact GPS coordinates at the city level

4.5 A Fine-Grained View of COVID-19 via Geo-tagged Tweets

4.5.1 Background and Tweet Collection

The Coronavirus Disease 2019 (COVID-19) pandemic transformed daily human lives globally. According to local pandemic conditions, countries worldwide adopted diverse regional regulations and policies to protect their residents and slow down the spread of COVID-19. Although countries like Sweden and South Korea did not lock down cities during the pandemic, most of the other countries, including China, Italy, Spain, and India imposed long and stringent lockdowns to restrict gathering and social contact. Inside the same country, different strategies and responses were also made by regions and cities at varying paces to fight against the COVID-19 crisis. Furthermore, people from different areas may express distinct attitudes and emotions on similar COVID-19 regulations due to local hospital resources, economic statuses, demographics, and many other geographic factors. Therefore, it is reasonable and necessary to take the location information into account when investigating COVID-19.

We took the U.S. as an example and built a large geo-tagged Twitter dataset for fine-grained investigations of the COVID-19 pandemic. Specifically, we utilized Twitter’s Streaming APIs to crawl real-time tweets containing a set of “coronavirus”, “wuhan”, “corona”, “nCoV” keywords related to the novel coronavirus outbreak since January 25, 2020.² After the World Health Organization (WHO) announced the official name of COVID-19 on February 11, 2020, we added “COVID19”, “COVID—19”, “coronapocalypse”, “Coronavid19”, “Covid_19”, “COVID-19”, and “covid” into our keyword set. We collected more than 170 million tweets generated by 2.7 million unique users from January 25 to May 10, 2020. We lost 38.5% tweets uniformly distributed among May 18 and Apr. 4 due to corrupted files, and missed 88 hours of data because of intermittent internet connectivity issues in the entire data collection period. To compensate for these data gaps, we sought for the COVID-19 dataset maintained by Chen et al. [20] and downloaded 16,459,659 tweets. We detected and examined the “place” attribute in collected tweet JSON files. If the embedded “country_code” was “US” and the extracted state was among the 50 states and Washington D.C. in the United States, we added the tweet into our geo-tagged dataset. After removing retweets, 650,563 geo-tagged unique tweets in the United States were collected. Among them, 38,818 tweets (5.96% of our dataset) were retrieved from the dataset proposed by Chen et al. [20]

4.5.2 Temporal and Geographic Distributions

Monthly and Daily Patterns

The monthly number of geo-tagged tweets in each state is shown in Figure 4.12, where March and April contributed the most tweet volumes. Figure 4.13 shows the daily distribution of geo-tagged tweets within the top 10 states, where Twitter users were most active. We lost around one-third detailed tweets between Mar. 18 and Apr. 4 due to the corrupted data. But we recorded the daily tweet counts during this period (see the dashed lines in Figure 4.13). Our crawlers shut down for 8 hours and 9 hours On Mar. 27 and Apr. 23 respectively, which caused the data gaps in the two days. Figure 4.13 demonstrates that daily tweet volumes

²This is two days after Wuhan lockdown.

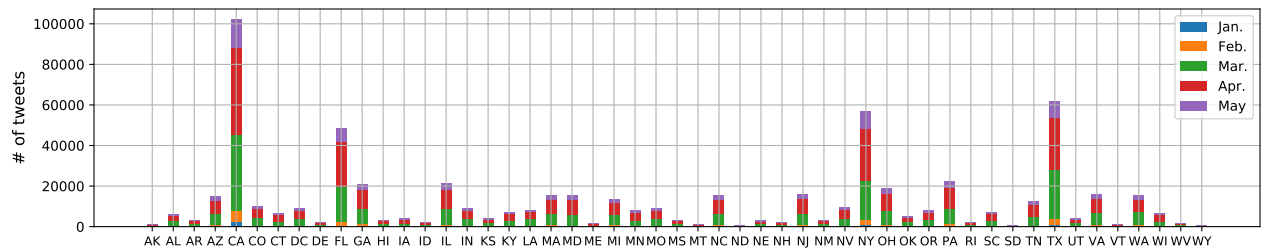


Figure 4.12: The monthly number of geo-tagged tweets in 50 states and Washington D.C. in the United States.

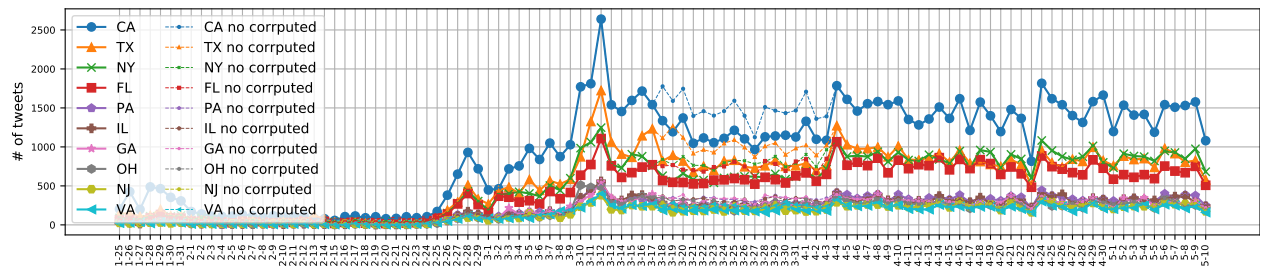


Figure 4.13: The daily number of tweets from the top 10 states generating most tweets.

generated by different states show similar trends. In fact, we tested statistical relationships regarding the daily volumes over time for all pairs of two arbitrary states, and found strong linear correlations existed among 93.2% state pairs with a Pearson's $r > 0.8$ and $p < 0.001$.

The daily patterns (if there was no corrupted data) inspired us to split the whole timeline into the following three phases. Phase 1 (from Jan. 25 to Feb. 24): people mentioned little about COVID-19 except for a small peak at the end of Jan; Phase 2 (from Feb. 25 to Mar. 14): the number of COVID-19 related tweets began to increase quickly, as many states had declared a state of emergency since Washington state did it on Feb. 29. Phase 3 (from Mar. 15 to May 10): people began to adjust to the new normal caused by COVID-19, such as working from home and city lockdowns.

Hourly Patterns

We also examined hourly tweeting distributions based on local time zones, as shown in Figure 4.14. The tweeting behaviors on workdays and weekends were studied separately because we wanted to figure out how the working status impacted on tweeting patterns. In Phase 1, there exists a tweeting gap from 8:00 to 16:59 between workdays and weekends. The tweeting peak occurred at 12:00-12:59 on weekends but at 17:00-17:59 on workdays. We think it may be explained by the fact that people engage at work during regular working hours and have little time to post tweets on workdays. But they become free to tweet concerns on COVID-19 after work. The hourly distribution patterns changed in Phase 2 when confirmed COVID-19 cases increased quickly in the United States. People posted COVID-19 tweets more frequently during business hours than at the same time slots on weekends. It is interesting to notice that a tweeting gap from 8:00 to 16:59, which is similar to that in Phase 1, reappeared in Phase 3 when most people had worked from home. These findings motivated us to take advantage of the tweeting frequencies on workdays and weekends to estimate work engagement in the future.

State-level Distribution

We extracted the state information from both general and exact tweet locations and calculated tweet volume percentages for each state, as shown in Figure 4.15(a). The most

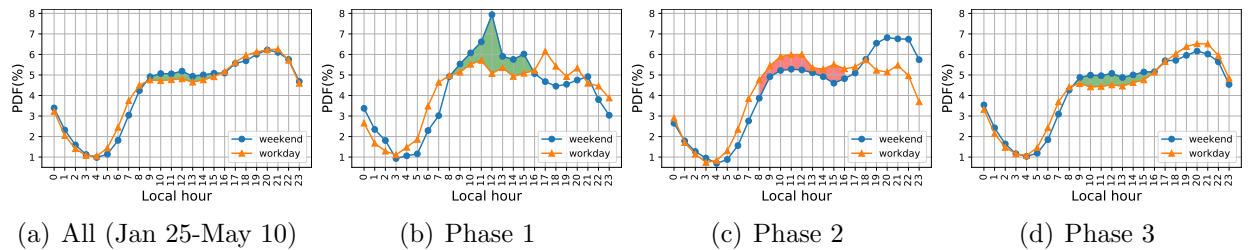


Figure 4.14: Hourly distribution in three phases: Phase 1 (Jan. 25 - Feb. 24), Phase 2 (Feb. 25 - Mar. 14), Phase 3 (Mar. 15 - May 10).

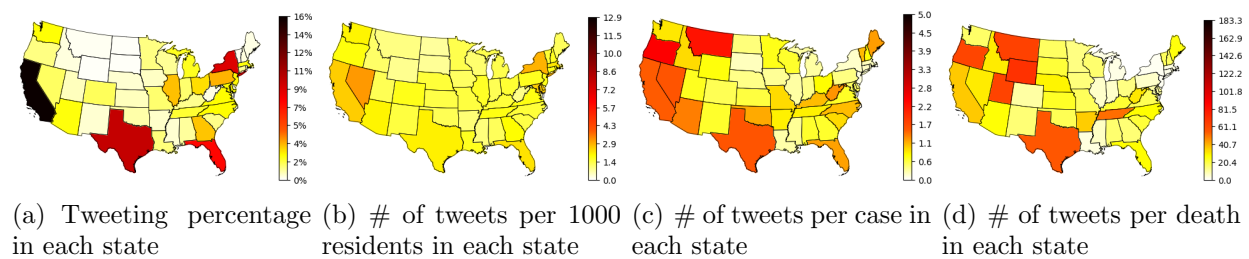


Figure 4.15: State-level geospatial distribution of geo-tagged tweets across the United States

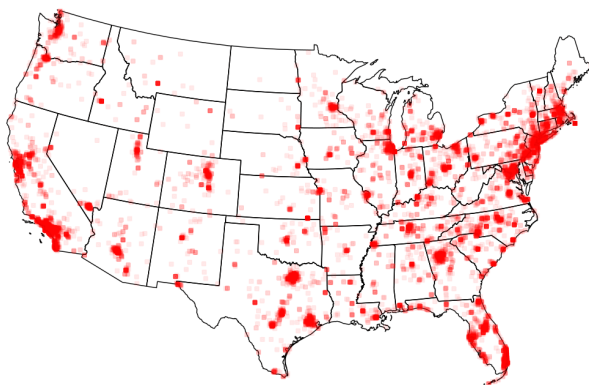
populated states, i.e., California, Texas, New York, and Florida, contributed the most tweets. In contrast, less populated states such as Wyoming, Montana, North Dakota, and South Dakota created the least tweets. We measured the relationship between tweet volumes and populations for all states, and found a strong linear correlation existed (Pearson’s $r = 0.977$ and $p < 0.001$).

Then we normalized tweet volumes using state residential populations. Figure 4.15(b) illustrates Washington D.C. posted the highest volume of tweets by every 1000 residents, followed by Nevada, New York, California, and Maryland. The rest states demonstrate similar patterns. We think the top ranking of Washington D.C. might be caused by its functionality serving as a political center, where COVID-19 news and policies were spawned.

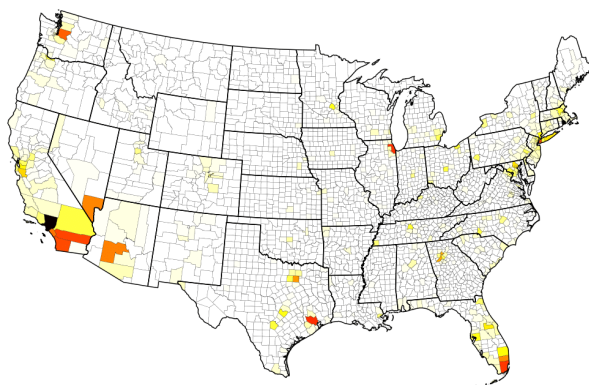
Unlike state populations, we did not find strong correlations between tweet counts and cumulative confirmed COVID-19 cases (Pearson’s $r = 0.544$ and $p < 0.001$) or deaths (Pearson’s $r = 0.450$ and $p < 0.001$). We further normalized tweet volumes based on COVID-19 cumulative number of cases and deaths in each state. Figure 4.15(c) and Figure 4.15(d) shows the average number of tweets generated by each COVID-19 case and each death. Note that Hawaii and Alaska (not plotted in Figure 4.15(c) and Figure 4.15(d)) ranked as the first and second in both scenarios. Residents in states like Oregon, Montana, Texas, and California reacted sensitively to both confirmed cases and deaths, as these states dominated in Figure 4.15(c) and Figure 4.15(d).

County-level Distribution

We utilized GPS locations to profile the geographic distribution of COVID-19 tweets at the county level because general tweet locations might not contain county information. In our collected geo-tagged tweets, 3.95% of them contained GPS locations. We resorted to Nominatim [87], a search engine for OpenStreetMap data, to identify the counties where each tweet GPS coordinate lay. Figure 4.16(a) and Figure 4.16(b) visualize the raw GPS coordinate and corresponding county distributions. Large cities in each state demonstrated a higher tweeting density than small ones. In fact, we found a strong correlations between GPS-tagged tweet counts and county populations (Pearson’s $r = 0.871$ and $p < 0.001$). But



(a) Exact (lat, lon) coordinates



(b) Geospatial distribution by county

Figure 4.16: Distribution of tweets tagged with exact GPS coordinates at the county level

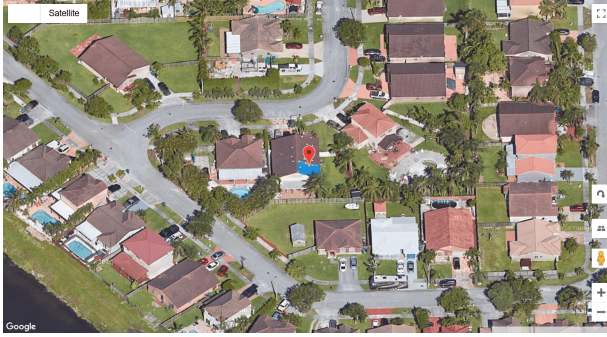
such correlations did not hold true for cumulative confirmed COVID-19 cases (Pearson’s $r = 0.590$ and $p < 0.001$) or deaths (Pearson’s $r = 0.497$ and $p < 0.001$).

4.5.3 Potential Privacy Leakage

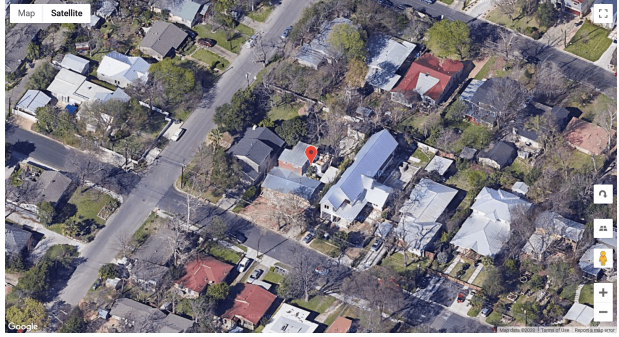
Many states imposed stay-at-home orders during the COVID-19 pandemic, confining people at home for months. We think GPS tagged tweets during this period are more likely to leak precise private location information, such as home addresses. Table 4.2 illustrate the top 20 parsed OpenStreetMap key-value results of GPS locations in our dataset. All of the three most popular key-value pairs, namely *place-house*, *building-yes*, and *highway-residential*, can be home locations (see Figure 4.17(a), Figure 4.17(b), Figure 4.17(c)). Note that *building-yes* determines the location is a building but cannot provide more specific building types. The key-value of *highway-residential* represents roads that are used for accessing residential areas and in residential areas. We also found home locations in the category of *building-house* (see Figure 4.17(d)).

Table 4.2: OpenStreetMap key-value results of GSP tagged tweets

Rank	Key	Value	Count	Rank	Key	Value	Count
1	place	house	3799	11	amenity	hospital	150
2	building	yes	1432	12	shop	supermarket	139
3	highway	residential	530	13	building	house	133
4	amenity	restaurant	320	14	tourism	hotel	114
5	highway	secondary	202	15	building	apartments	108
6	amenity	school	197	16	highway	motorway	104
7	highway	primary	196	17	amenity	fast_food	93
8	amenity	place_of_worship	188	18	amenity	university	91
9	highway	tertiary	155	19	amenity	townhall	84
10	building	commercial	152	20	amenity	cafe	80



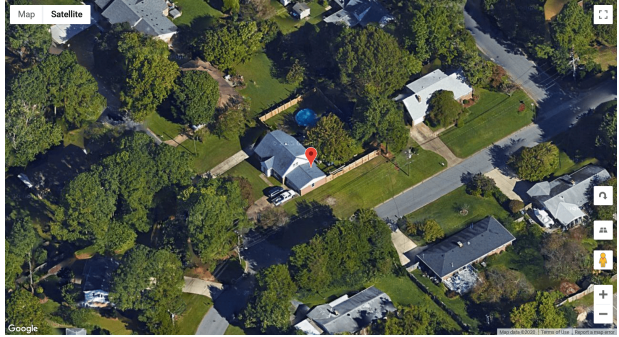
(a) Home location in *place-house* category



(b) Home location in *building-yes* category



(c) Home location in *highway-residence* category



(d) Home location in *building-house* category

Figure 4.17: Home location examples on Google Map

Chapter 5

Secure Sharing of Private Locations through Homomorphic Bloom Filters

In this chapter, we investigate the problem on how to keep users' location data secure, while at the same time allowing servers to perform basic operations on the data. We demonstrate secure computation primitives on location data where the servers should have no knowledge of the plaintext, i.e., the servers should not even keep the private keys to decrypt data. In this way, compromised servers will not cause leaked user data. We have proposed a generalized secure set membership check framework based on the recently developed homomorphic encryption and an advanced data structure called the bloom filter. We also use real-world datasets to evaluate the proposed practical prototypes, which are implemented on the open-source homomorphic libraries. The preliminary results demonstrate the feasibility and efficiency of the proposed approach as well as the security of the protocol designs. Note that this chapter is based on my published paper:

Secure Sharing of Private Locations through Homomorphic Bloom Filters. IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity, Best Paper Award), 2018.

5.1 Background and Framework Overview

5.1.1 Assumptions

Our system design consists of three parties: Alice, Bob, and the aggregation server. In a typical application scenario, Alice first selectively publishes her location datasets after encrypting them with her public key. Later, another user Bob sends his location queries encrypted by Alice’s public key to the aggregation server. The server is responsible for performing computation tasks on the encrypted data, and sends the (still encrypted) results to Alice. Finally, Alice decides if there is an intersection by decrypting the results via her own private key. We assume that Alice and Bob are usually not malicious. They will follow the protocol correctly, but once the protocol has ended they can perform any computation they want on the information (encrypted or otherwise). If Alice finds out that there is an intersection with the trajectory of the query, the further interactions between Alice and Bob are out of the scope of this protocol.

5.1.2 Background

As shown in Figure 5.1, the homomorphic encryption method and the bloom filter are the two core components in our protocol. Next, we give a little background information about the two components.

Homomorphic Encryption: This method takes a foundational role in our system. Briefly, in a fully homomorphic encryption scheme, the following equations hold true:

$$m_1 + m_2 = D(E(m_1) + E(m_2)) \quad (5.1)$$

$$m_1 * m_2 = D(E(m_1) * E(m_2)) \quad (5.2)$$

In these equations, $E()$ represents the encryption operation, and $D()$ represents the decryption operation. The computation that is performed with encrypted values can be translated to operations in the plain text domain. This allows parties to perform blind computation on encrypted values.

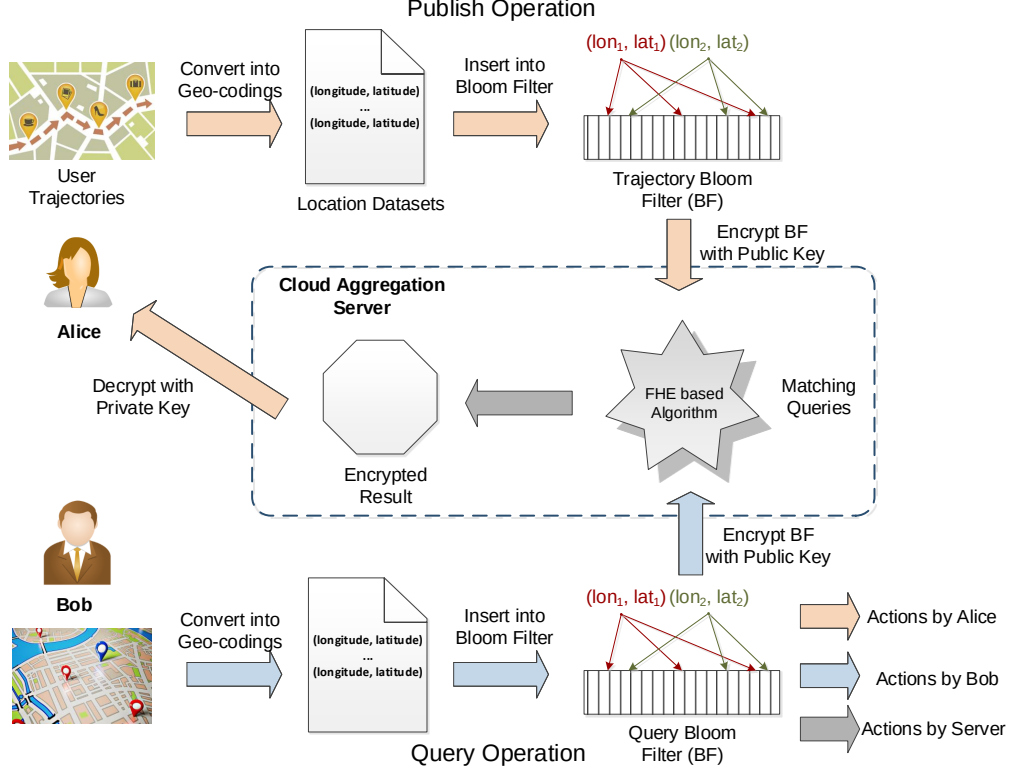


Figure 5.1: Computational model architecture

Bloom Filter: We use standard notations on bloom filters to represent their operations, as follows:

- m : the size (total number of bits) of the bloom filter;
- k : the number of hash functions;
- n : the number of elements inserted in the bloom filter;
- p : the false positive probability of the bloom filter;
- t : the number of bits flipped to one.

The probability of false positives p given a parameter setting (m, k, n) is calculated as:

$$p \approx (1 - e^{-\frac{kn}{m}})^k \quad (5.3)$$

For a given bloom filter size m and the number of inserted elements n , the number of hash functions k that minimizes the false positive is:

$$k = \frac{m}{n} \ln 2 \quad (5.4)$$

For a given number of inserted elements n and the desired false positive p , the required number of bits m is:

$$m = -\frac{n \ln p}{(\ln 2)^2} \quad (5.5)$$

We use these results later for parameter settings and the performance analysis of our protocol in Section 5.4.

5.1.3 Main Ideas on an Ideal Protocol

The main idea for an ideal protocol works as follows. The user Alice first inserts all locations that she wants to publish into a configured bloom filter BF_A , by hashing each of these locations k times into BF_A , and flipping corresponding bits as 1. After all locations are inserted, suppose that t bits at index $a_1, a_2, \dots, a_t$ in BF_A have been set as 1s. Our next goal is to encrypt the bloom filter BF_A properly. Specifically, we observe that its flipped bits can be represented as a polynomial:

$$f(x) = \prod_{i=1}^t (x - a_i) \quad (5.6)$$

Alice then sends the encrypted polynomial $f(x)$ either in the product form or the expanded form to the aggregation server. Alice also sends the details on the k hash functions, and the configuration parameters for BF_A . This is necessary as such information will be later used by Bob for encryption needs. Even though Alice sends the hash functions, as the $f(x)$ is encrypted with the public key, the aggregation server has no way to learn which bits have been set as 1s in BF_A .

The aggregation server next waits for the incoming queries. When doing the query, Bob does not send the raw location data to the server. Instead, he will first construct a new

bloom filter BF_B by obtaining the k hash functions and BF_A 's configuration parameters from the server. Then the queried location is hashed into BF_B by flipping k bits at index $b_1, b_2, \dots, b_k$. Next, Bob encrypts the k indices as $E(b_1), E(b_2), \dots, E(b_k)$ using Alice's public key, and sends them to the server to check whether BF_A has the corresponding bits flipped as 1s by evaluating $f(x)$ in the encrypted form as $E(f(E(b_i)))$. We know that, based on the nature of this encryption method, the following equation also holds true:

$$D(E(f(E(b_i)))) = f(b_i) \quad (5.7)$$

Furthermore, if the bit at index b_i in BF_A has been flipped to 1 by Alice earlier, this evaluation result must be 0 based on the product nature of the polynomial. Therefore, if we construct another polynomial as $\sum_{i=1}^k E(f(E(b_i))^2)$ (we use square is to prevent the evaluation may sometimes lead to negative numbers), we have:

$$H(b) = D\left(\sum_{i=1}^k E(f(E(b_i))^2)\right) = \sum_{i=1}^k f(b_i)^2 \quad (5.8)$$

Observe that in this equation, only when all bits at the k indices in BF_A have been flipped to 1, the result $H(b)$ will be 0. If $H(b)$ is not 0, then it means that the query is not in the bloom filter of Alice. We note that the multiplication steps for computing $f(b_i)^2$ is t (based on the factorized form of $f(x)$). So the total time of multiplication involved in computing $H(b)$ is $O(k * t)$. A large t , like 200, makes such a multiplication impractical, because the fast increased size of ciphertexts worsens the computation overhead greatly. Some techniques, such as relinearization [11] and approximate eigenvector method [38], have been proposed to balance the multiplication computation capacity and computation overhead, but they still do not work efficiently on a huge multiplication depth in real applications. Therefore, we design the following optimizations towards practical protocols containing a greatly reduced number of multiplication operations. With these optimizations, the protocols can be made practical. We call these protocols *practical optimizations* for this reason.

5.2 Optimized Protocols

5.2.1 Optimization 1: A Lightweight Homomorphic Bloom Filter

We now describe the first working design using fewer additions and multiplications compared with the idealized design above. Specifically, it requires k additions, no multiplication and one decryption to speed up the query. The pseudocode is shown in Algorithm 2. Alice first inserts her locations into a bloom filter BF_A using k hash functions, then encrypts the value of every bit $v_1, v_2, \dots, v_m$ in BF_A using her public key PK . The $E(v)$ is then submitted to the aggregation server. Next, Bob hashes his queried location into BF_B , a bloom filter configured with the same parameters as BF_A , where the bits at index $b_1, b_2, \dots, b_k$ are flipped to 1s. These k indices are submitted to the server in plaintext, which we think is still safe because the chosen k cryptographic hash functions ensure that the server can hardly compute the plain text from hashed values reversely. The server then sums all elements in $E(v)$ whose index is in b . At last, the server subtracts k , i.e., the length of b , from the sum and returns the ciphertext result to Alice for decryption using her secret key SK . The whole evaluation and decryption can be expressed as follows:

$$plain_result = D\left(\sum_{j=1}^k E(v_{b_j}) - k\right) \quad (5.9)$$

If *plain_result* equals 0, it implies that all values of v_{b_j} are same and equal 1. In other words, all bits at index b_j in BF_A are flipped as 1s. So, we can say Bob interacts Alice with a certain probability of false positive expressed by Equation 5.3. Otherwise, they do not intersect at the queried location 100%.

5.2.2 Optimization 2: An Improved Homomorphic Bloom Filter

To further improve the security of Bob's data in optimization O1, we propose an improved design as shown in Algorithm 3. Like optimization O1, Alice first inserts her locations into a bloom filter BF_A , where t bits at index $a_1, a_2, \dots, a_t$ have been set as 1s. Then Alice uses her public key PK to encrypt each negated a_i as the encrypted $E(-a_i)$, and submits

Algorithm 2: A Lightweight Design of the Homomorphic Bloom Filter

Data: BF_A : Alice's Bloom filter; PK : Alice's public key; SK : Alice's secret key;
 BF_B : Bob's Bloom filter.

Parameter: k : the number of hash functions; m : the size of BF_A

Alice: Bloom Filter Encryption

```
1  $v \leftarrow$  all bits in  $BF_A$  ;  
2 for  $i = 1 \rightarrow m$  do //  $m$  is the size of  $BF_A$   
3   |  $E(v_i) \leftarrow$  encrypt  $v_i$  using  $PK$  ;  
4 end  
5 send  $E(v)$  to the aggregation server ;
```

Bob: Query Encryption

```
6  $b \leftarrow$  the indices of bits flipped to 1s in  $BF_B$  ;  
7 send  $b$  to the aggregation server ;
```

Sever: Evaluation

```
8  $sum \leftarrow 0$  ;  
9 for  $j = 1 \rightarrow k$  do  
10  |  $sum \leftarrow sum + E(v_{b_j})$  ;  
11 end  
12  $cipher\_result \leftarrow sum - k$  ;  
13 send  $cipher\_result$  to Alice ;
```

Alice: Result Decryption

```
14  $plain\_result \leftarrow$  decrypt  $cipher\_result$  using  $SK$  ;  
15 if  $plain\_result = 0$  then  
16   | return intersection exists ;  
17 else  
18   | return no intersection ;  
19 end
```

Algorithm 3: An Improved Design of the Homomorphic Bloom Filter

Data: BF_A : Alice's Bloom filter; PK : Alice's public key; SK : Alice's secret key;
 BF_B : Bob's Bloom filter.

Parameter: k : the number of hash functions; t : the number of bits flipped as 1 in BF_A

Alice: Bloom Filter Encryption

```
1  $a \leftarrow$  the indices where bits are flipped to 1s in  $BF_A$  ;
2 for  $i = 1 \rightarrow t$  do
3    $E(-a_i) \leftarrow$  encrypt  $-a_i$  using  $PK$  ;
4 end
5 send  $E(-a)$  to the aggregation server ;
```

Bob: Query Encryption

```
6  $b \leftarrow$  the indices where bits are flipped to 1s in  $BF_B$  ;
7 for  $j = 1 \rightarrow k$  do
8    $E(b_j) \leftarrow$  encrypt  $b_j$  using  $PK$  ;
9 end
10 send  $E(b)$  to the aggregation server ;
```

Sever: Evaluation

```
11 for  $j = 1 \rightarrow k$  do
12   for  $i = 1 \rightarrow t$  do
13      $cipher\_result_{ij} \leftarrow (E(b_j) + E(-a_i)) * z$  ;
14   end
15 end
16 send  $cipher\_result$  to Alice ;
```

Alice: Result Decryption

```
17 for  $j = 1 \rightarrow k$  do
18    $flag \leftarrow FALSE$  ;
19   for  $i = 1 \rightarrow t$  do
20      $plain\_result_{ij} \leftarrow$  decrypt  $cipher\_result_{ij}$  using  $SK$  ;
21     if  $plain\_result_{ij} = 0$  then
22        $flag \leftarrow TRUE$  ;
23       break ;
24     end
25   end
26   if  $flag = FALSE$  then
27     return no intersection ;
28   end
29 end
30 return intersection exists ;
```

them to the aggregation server. After launching a query, Bob first requests Alice’s PK and her k hash functions from the aggregation server. Then Bob hashes his queried location into BF_B with flipping bits at index $b_1, b_2, \dots, b_k$ to 1s. The indices are then encrypted as $E(b_1), E(b_2), \dots, E(b_k)$ by PK and sent to the server. Once receiving the encrypted query from Bob, the server starts the evaluation based on $E(-a)$ and $E(b)$. Specifically, for each $E(b_j)$, the server performs $E(b_j) + E(-a_i)$, where $i = 1, 2, \dots, t$, and gets t encrypted results which are then decrypted as $D(E(b_j) + E(-a_i))$ respectively by Alice using her secret key SK . Once one of the t encrypted results of b_j equals 0, it means the bit at index b_j in BF_A is set as 1. If all bits at index $b_1, b_2, \dots, b_k$ in BF_A are set as 1s, we can conclude Bob has an intersection with Alice at a certain confidence level which is determined by BF_A ’s rate of false positives (we analyze the impact of this false positive rate later). Otherwise, Bob has no intersection with Alice at the queried location 100%. Note that in this design, the information of Bob might still be leaked to Alice no matter whether they have trajectory intersections or not, because Alice can obtain the value of b_j by:

$$b_j = \frac{\sum_{i=1}^t [D(E(b_j) + E(-a_i)) + a_i]}{t} \quad (5.10)$$

To address this problem, we introduce the randomness during the evaluation by multiplying $E(b_j) + E(-a_i)$ by a random positive integer z . Thus, if $D(E(b_j) + E(-a_i))$ equals 0, $D((E(b_j) + E(-a_i)) * z)$ still equals 0, while other values vary dramatically.

5.2.3 Optimization 3: A Bit-wise Homomorphic Bloom Filter

Although the optimization O1 and O2 are feasible, they are built entirely on top of the underlying homomorphic encryption scheme, and treat such a scheme as a black box. The advantage of doing this is that even if we change the implementation of the homomorphic encryption, these designs still work without any modification.

However, we also observe that the recently proposed homomorphic encryption schemes are usually based on bit-wise operations, a feature that is highly similar to the underlying scheme of bloom filters. This similarity reveals an opportunity for cross-layer optimization. Specifically, a large number of homomorphic encryption schemes encrypt data in a bit-wise manner, i.e., each bit in the plaintext is encrypted as a separate ciphertext. Later,

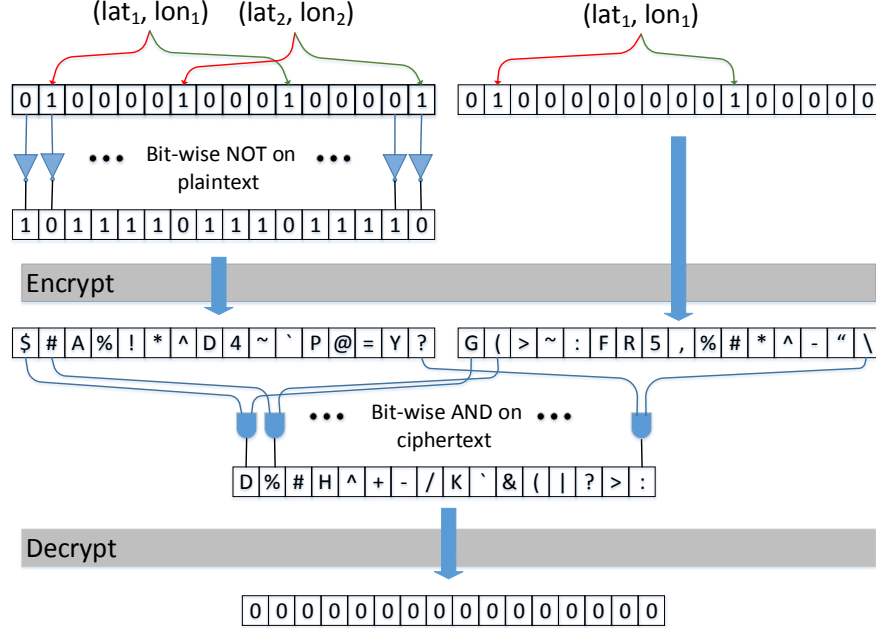


Figure 5.2: Bloom filter encrypts a query with the bit-wise encryption scheme

the computation is represented as a boolean circuit with XOR and AND gates, where the input is the ciphertext for each encrypted bit. As this mechanism breaks down an arbitrary computation into bit operations, it may lead to highly complex circuits. However, in our case, after transforming the original query operation that involves several arithmetic operations into the query operation based on bloom filter, the query operation can be easily decomposed to a single bit-wise operation.

We represent the bloom filter as a bit array, which can in turn be represented as an array of integers. We then use the integer array as encryption input, and are able to naturally obtain the encrypted bloom filter without using the polynomial methods as developed in the previous subsections. Specifically, we re-design the query operation as shown in Figure 5.2. Suppose Alice constructs a bloom filter BF_A with all her location dataset. Bob also constructs a bloom filter BF_B with only the location he wants to query. To check whether the location queried by Bob exists in Alice's location set, we only need to check if all bits flipped as 1s in BF_B are also set as 1s in BF_A , which can be decomposed into two bit-wise operations. First, perform a bit-wise AND operation between the two bloom filters

to get a bit array containing wanted bits in BF_A . Then, perform a bit-wise XOR operation between BF_B and the resulting bit array from the last step to check whether all wanted bits are set as 1. Therefore, the query operation can be expressed as:

$$query_result = BF_A \& BF_B \oplus BF_B \quad (5.11)$$

If the queried location of Bob exists in Alice's location set, the *query_result* in Equation 5.11 will be zero. (Equation 5.11 will fail when both bloom filters are empty, i.e., both Alice and Bob have no trajectory, but this can be easily prevented.) Furthermore, the operation can be simplified as:

$$query_result = \neg BF_A \& BF_B \quad (5.12)$$

Note that the bit-wise NOT operation only requires one input, thus can be performed locally before the encryption occurs. This can further reduce the required operation overhead under encryption form.

With Equation 5.5, we can calculate the optimal m given a certain number of elements n and required false positive level p . We use l to represent the length of an integer. Then the system needs to encrypt $\lceil m/l \rceil$ integers. During the query, the system needs to calculate encrypted AND operation for $\lceil m/l \rceil * l$ bit. In the last step, the system needs to decrypt $\lceil m/l \rceil$ integers to get the result. Before the computation, both the Alice and Bob need to send their bloom filter to the server which is $\lceil m/l \rceil$ integers. After the computation has been done, $\lceil m/l \rceil$ integers need to be sent back as the result.

5.3 Security Analysis

As the goal of our system is to provide secure and private protocols for users to share location data without leaking sensitive information, we now analyze the security of this system in the presence of *semi-honest adversaries* from the perspectives of Alice, Bob, and the server.

5.3.1 Alice’s Privacy and Actions

Observe that in our protocol designs, as long as Alice correctly encrypts her data with the public key, her security and privacy is well protected. Therefore, users will not be discouraged from adopting this service and publishing their location data due to privacy concerns.

On the other hand, Alice is able to decrypt the results received from the aggregation server only when there are intersections. In optimizations O1 and O3, Alice only gets random integers unless it is zero, indicating that there exists an intersection. In optimization O2, we introduced intentional randomness during the evaluation to prevent Alice from identifying Bob’s position.

5.3.2 Bob’s Privacy and Actions

As discussed in Section 5.3.1, all of the optimizations O1 to O3 reveal no Bob’s location data to Alice. Next, we consider whether Bob’s location data is leaked to the aggregation server. In optimizations O2 and O3, Bob submits his query to the server in an encrypted way, which guarantees the server has no access to Bob’s data. In optimization O1, the server may only obtain Bob’s positions of the bits flipped as 1 but not his location data in plaintext. Strong cryptographic but computationally intensive hash functions, such as password-based key derivation functions, can make the server’s brute-force attacks on Bob’s location data less effective.

5.3.3 The Aggregation Server’s Actions

We assume that the messages between Alice, Bob, and the server are encrypted, so the communication channel can be secure. For optimization O1, the server’s potential brute-force attack on Bob has been presented in Section 5.3.2. For optimizations O2 and O3, even if the server is compromised and is able to read and write all messages, information is not leaked to the server due to the very nature of the homomorphic encryption: all computations are based on encrypted data, not plain text.

On the other hand, the server is indeed able to log the IP addresses and user accounts of the users. But we consider this not a security problem as this is the standard operations

of social networks and apps. Finally, the server may sabotage the system by refusing to act as the intermediate server. In practice, this does not often happen as servers lack the motivations to block users arbitrarily.

5.4 Evaluation

In this section, we report experimental results of our homomorphic bloom filter protocols based on a real-world smartphone mobile dataset. First, we briefly describe the dataset and the experimental setup. Then, the parameter settings of the protocols are presented. Finally, we compare the computation and communication overhead of different protocols.

5.4.1 Datasets and Experimental Setup

Our dataset, provided by a telecommunication carrier in China, consists of thousands of cellular data access records in a small city. More specifically, this dataset includes 7607 unique users, 121 cell towers, and was collected by from 6:00 pm to 9:00 pm on a Sunday in 2014. Each mobile data access record contains the coordinate of the cell tower with which the user communicates. A sequence of records of a user can be viewed as her history trajectory. We randomly pick users and generate queries for evaluation. All experiments are performed on a PC with a 3.10GHz Xeon E3-1220 processor, and 32GB memory, running Red Hat 4.8.5.

5.4.2 Parameter Settings

We group the users in our dataset into 4 categories, by the number of unique towers they visited: group 1 (1-15 points), group 2 (16-30 points), group 3 (31-45 points), group 4 (above 45 points). The default false positive probability is set as 0.1. According to Equation 5.4 and Equation 5.5, we use a 250-bit bloom filter with 4 hash function for group 1 & 2, a 500-bit bloom filter with 4 hash function for group 3 & 4.

We implement the optimization O1 and O2 on an integer encryption system (SEAL) [65], and the optimization O3 on a binary encryption system based on [117]. In SEAL, we set

polynomial modulus as “ $1x^{1024}+1$ ”, coefficient modulus as $FFFFFFFF00001$ and plaintext modulus as 256. Under this setting, we calculate the size of a freshly encrypted integer in SEAL as $1025 \times 48 \times 2 = 98400$ bits, where 1025 is the number of coefficients in one polynomial, 48 is the number of bits per coefficient occupies, and 2 is the size of the array of polynomials. To minimize the difference of the size of the ciphertext, for the binary encryption system, the security parameter λ is set as 4, and each bit in the plaintext is encrypted into a 1024-bit wide ciphertext.

5.4.3 Computation Overhead

The computation overhead can be decomposed into three parts: the encryption time, the query computation time and decryption time, where the encryption happens at both Alice and Bob. We randomly perform 20 queries for each group of users and compare the computation overhead of all three optimizations in Figure 5.3.

As we can see, in optimization O1, the encryption overhead at Alice dominates the overall computation overhead, as Alice requires to encrypt every bit in her bloom filter BF_A . In optimization O2, the encryption overhead at Alice is much larger than encryption overhead at Bob, because the number of integers to be encrypted is determined by the number of flipped 1s in the bloom filter. This also explains why the encryption overhead increases as the number of location points increase. Since optimization O2 incurs large intermediate results to be transmitted back to Alice, the decryption overhead (almost 5 seconds) is high. The computation overhead of optimization O3 can reach 24 seconds in total (11 seconds for encryption), which is the highest among all three protocols. We think it is mainly caused by the setting of the security parameter λ in our evaluation. The binary homomorphic encryption library we used requires λ to be set as a power of two. To keep the size of ciphertext generated by this library and the SEAL on the same scale, the λ is set as 4 in our experiments. If we minimize the λ to 2, both the computation overhead and the ciphertext size decline exponentially.

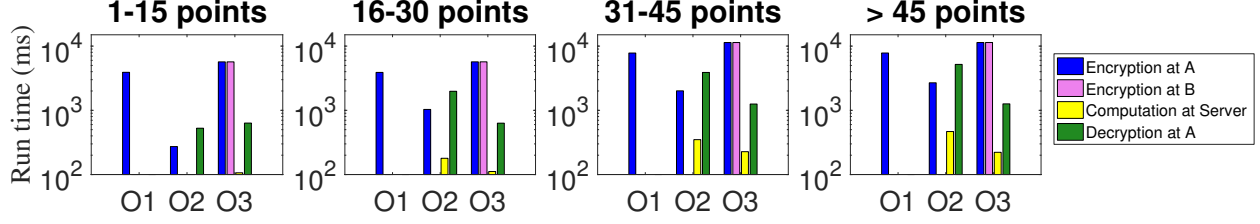


Figure 5.3: Computation overhead

5.4.4 Communication Overhead

The communication overhead mainly consists of three parts: the ciphertext submitted to the server by Alice, the ciphertext submitted to the server by Bob, and the computation results sent from the server to Alice. We use the size of the ciphertext to be transmitted as the main measure for communication overhead. We compare the communication overhead of all three optimizations in Figure 5.4.

As we can see that in optimization O1, most of the overhead happens at sending ciphertext from Alice to the server, as each bit in the bloom filter is treated as an integer. Thus, optimizations O2 and O3 beat optimization O1 regarding the communication overhead generated by the data transmission from Alice to the server. For optimization O2, besides the heavy communication overhead at transmitting ciphertext from Alice to the server, a large number of encrypted intermediate results sent from the server to Alice also cause a high communication overhead. Optimization O3, which adopts a bit-wise encryption scheme, has the lowest communication overhead with only tens of KB in size. All three parts of communication overhead for optimization O3 keep the equal, as the bloom filters of Alice and Bob are always treated as integer arrays with the same size in the bit-wise computations.

5.4.5 Protocol Accuracy

As the bloom filter has possible false positives, we take the users in group 3 (31-45 points) as an example to demonstrate the computation overhead in Figure 5.5 and communication overhead in Figure 5.6 given different false positive possibility p . We set the corresponding

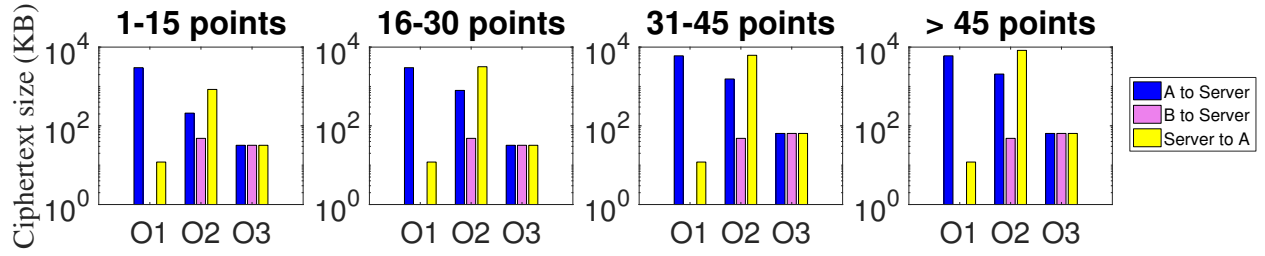


Figure 5.4: Communication overhead

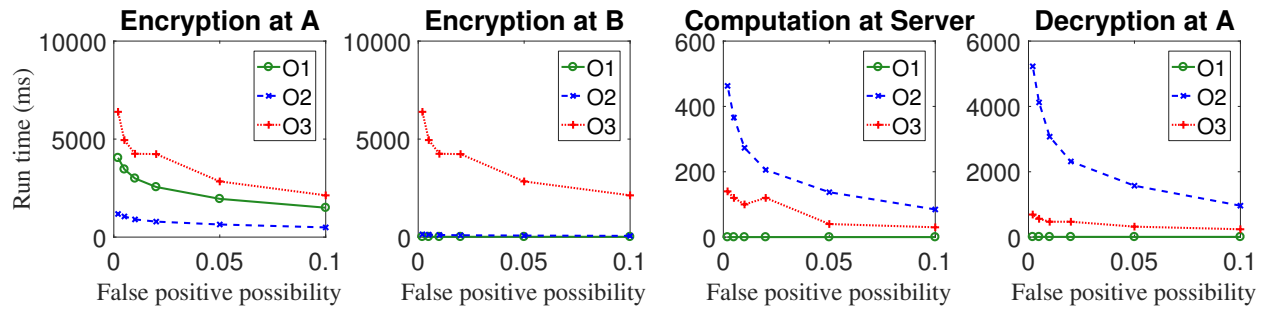


Figure 5.5: Accuracy vs computation overhead

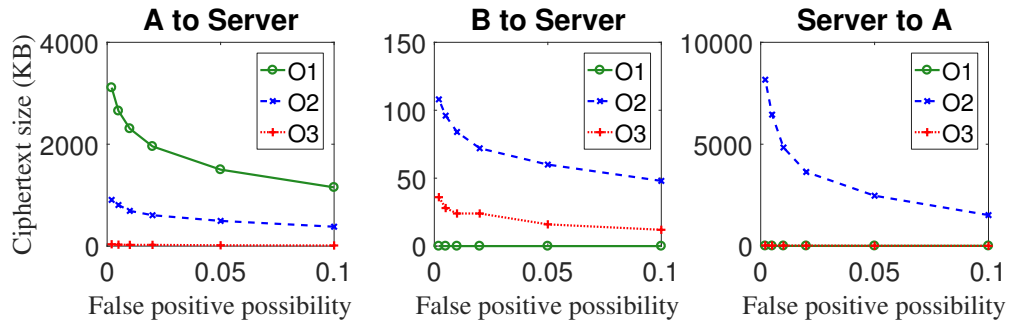


Figure 5.6: Accuracy vs communication overhead

number of hash functions k and the size of bloom filter m based on Equation 5.4 and Equation 5.5.

For optimization O1, as the encryption at Alice and transmission of data from Alice to server dominate the computation overhead and communication overhead respectively, they both decrease as p increases, whereas other overheads are quite stable. For optimization O2, both computation overhead and communication overhead decrease as p increases with the exception that the encryption overhead at Bob keeps relatively stable. Similar to optimizations O1 and O2, optimization O3 also shows a trend of decreasing overhead as p increases.

Chapter 6

SenCAPTCHA: A Mobile-First CAPTCHA Using Orientation Sensors

As most location-based services are operated on mobile devices, it is reasonable to imagine bots mimic a mobile environment to attack remote location servers such as querying locations constantly, or submitting fake locations. To mitigate these attacks, we propose the SenCAPTCHA, a novel orientation sensor based CAPTCHA system designed for mobile devices, to distinguish between humans and bots. SenCAPTCHA utilizes animal keypoint detection as the hard problem to prevent bots, while humans could solve it by slightly tilting their smartphones. Our usability studies demonstrate that compared to other existing CAPTCHAs (audio-, image-, text-, and video-based), SenCAPTCHA has higher perceived usability and faster completion times. Additionally, it is identified as the favorite CAPTCHA by over half of our participants. Note that this chapter is based on my published paper:

Y. Feng, Q. Cao, H. Qi, and S. Ruoti. SenCAPTCHA: A Mobile-First CAPTCHA Using Orientation Sensors. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (formerly known as UbiComp), 2020.

6.1 SenCAPTCHA

SenCAPTCHA is a mobile-first CAPTCHA scheme leveraging the orientation sensors available on mobile devices to provide a CAPTCHA that works even on devices with

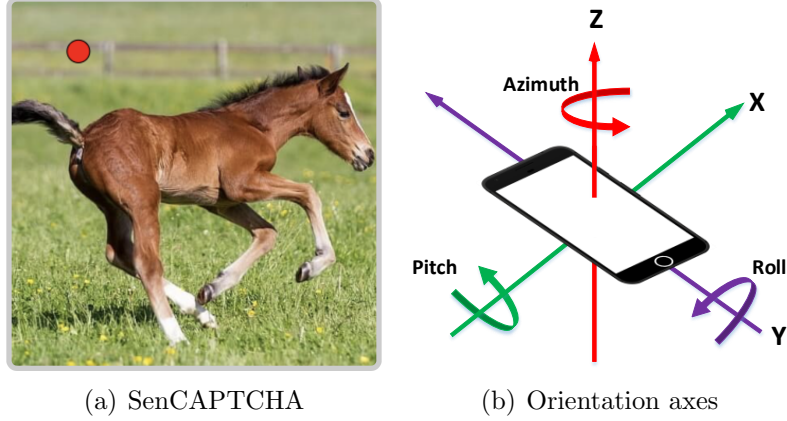


Figure 6.1: The red moving ball is controlled by built-in orientation sensors (X and Y axes) on mobile devices. Users tilt their devices to move the ball to target positions to pass the SenCAPTCHA test.

constrained screen real estate (e.g., smartphones, smartwatches). In SenCAPTCHA, users are shown a picture of an animal and are asked to tilt their phones to guide a red ball into the center of that animal’s eye (see Figure 6.1). The design of SenCAPTCHA was guided by three principles:

- **Low effort.** SenCAPTCHA requires no expert knowledge and is designed to be easily solved by most users. Instead of relying on conventional I/O interface (e.g., keyboard, mouse, touchscreen), SenCAPTCHA utilizes orientation sensors, offering a natural, novel, and efficient user experience for mobile devices.
- **Enjoyable.** We model SenCAPTCHA after sensor-based games on mobile devices that ask users to complete various tasks by rotating their phones. By adopting a more game-like approach, we hope to reduce the burden of completing SenCAPTCHA (similar to other puzzle-based CAPTCHAs).
- **Clean UI.** SenCAPTCHA avoids blurred visual elements, often found in text-based and video-based CAPTCHAs, increasing its visual appeal and ease of use.



Figure 6.2: Image mutations to improve attack resilience

6.1.1 Resilience to Bots

To prevent bots from solving SenCAPTCHA, we take advantage of the fact that detecting animal facial keypoints is a hard problem. One challenge for animal facial keypoint detection is that, due to the high annotation and collecting cost for training, the amount of annotated data of animals is too limited to effectively leverage Convolutional Neural Networks (CNNs), which have shown impressive performance on human faces. For example, the AFLW [63] dataset consists of 25,993 human faces, while the horse and sheep datasets from previous work [92, 125] only have 3,717 and 600 images, respectively. Furthermore, even if a CNN could be trained, it would need to be specific to each animal species requiring another classifier to first identify the animal. Another challenge is that animal images generally have more diverse backgrounds—for example, they are often taken in environments where the animal somewhat blends into the background.

To further improve SenCAPTCHA’s resilience to attacks, we randomly rotate, zoom and translate, or tile the animal picture as shown in Figure 6.2. This makes keypoint detection harder by changing the shape of the animal, modifying facial features, or partially obscuring the face preventing the creation of an accurate bounding box [125, 92], even though labeled datasets are available for training models. For example, we observed the average failure rate across keypoints of a deep learning model proposed in [92] increased from 8.36% to over 90% when image mutations were applied on a large dataset containing 3717 horse images. We

measured the security and usability impact of these changes, and this is discussed later in the paper (see Section 6.2 and Section 6.3 respectively).

6.1.2 System Overview

SenCAPTCHA works in a client-server model and is broken up into two components. A server-side component provides the background image and determines when the CAPTCHA is solved. A HTML5/JavaScript-based client-side component displays the CAPTCHA to the user and records the ball’s movement to send to the server for verification. Importantly in this separation of duties, the client-side portion never learns the location of the eye, as the client-side component runs inside the attacker’s environment.

As shown in Figure 6.3, the workflow of SenCAPTCHA can be broken up into three primary phases: (1) SenCAPTCHA is initialized; (2) the user tilts their devices to move the ball into the animal’s eye; and (3) the ball’s trajectory is analyzed to distinguish between authentication human users and bots. Most operations are performed on the server with only the UI-rendering and sensor-reading handled by the client.

Initialization

During the initialization phase, the server needs to create a puzzle image, determine how close the ball must get to the target, select a starting position for the ball, and send this information to the client.

Creating Puzzle Image To create the puzzle image, the server first selects an image from its image corpus. This corpus contains a set of images along with their respective keypoints. It then mutates the chosen image using one of the following three techniques (see Figure 6.2):

- **Rotate.** The image is rotated by a random angle. The image is then scaled such that no white-space is shown to the user. If the eye falls outside the area shown to the user or is too close to the edge, then another randomly selected rotation angle is chosen.
- **Zoom and translate.** A random horizontal scaling factor and a separate random vertical scaling factor are used. After scaling the image, it is translated to ensure that



the eye is visible to users (within the area shown to users) and is not too close to image edges.

- **Tile.** The image is broken into several tiles (currently fixed at 9 [3x3], though it could also be randomly selected), then those tiles are randomly arranged and displayed to the user. As with other methods, if the eye is too close to image edges, it will be moved to another random location.

As part of this mutation, the server will also calculate the new location of the image’s keypoint. Note that each mutation can take a single image from the image corpus and generate many unique puzzles. For example, the tile mutation takes a single image, breaks it into nine tiles, and then randomly orders those tiles, producing one of $9! = 362,880$ different puzzles. This process ensures that even with a small corpus of images, we can generate a massive number of puzzles for SenCAPTCHA’s use.

Calculate Target Distance Next, the server calculates how close the center of the ball must be to the center of the puzzle’s keypoint for the puzzle to be solved. This distance (d) is measured in and is calculated based on a server-side *tolerance* value that indicates how close the two centers must be as a percentage of the overall image size. Note that there is an inverse relationship between tolerance and the difficulty of solving the puzzle. The formula for calculating d from tolerance is given in Equation 6.1.

$$d = tolerance * \frac{width + height}{2} \quad (6.1)$$

The radius of the ball is made equal to distance d to ensure that if the user sees the ball touch the keypoint, the puzzle will also be solved. We do impose a minimum radius of 5 pixels for the ball to ensure that it is visible on small form factors with limited resolution.

Select Starting Position The ball the user needs to move starts at a random position. This position is picked by selecting one option from each of the following sets and combining them: {top, middle, bottom} and {left, center, right}.

Send Information to the Client The server-side component sends the client-side component the following data: (1) the mutated image, (2) the ball’s starting location, and (3) the ball’s radius. The client-side code will then handle rendering the image to a `HTML Canvas` element. The ball is also rendered to this element at the indicated starting point with the given radius. The ball itself is colored red, with a solid black outline.

Moving the Ball

To move the ball, users tilt their devices.¹ To track the device’s tilt, we use the azimuth-pitch-roll convention to represent the mobile device’s orientation angles, as shown in Figure 6.1(b). Among the three angles, only the pitch angle (degrees of the rotation about the X axis) and the roll angle (degrees of the rotation about the Y axis) are used to compute the ball’s position, since the azimuth angle denotes a compass-like direction value and contributes nothing to the ball’s movement. The pitch angle β and roll angle γ change when tilting mobile devices upwards/downwards and rotating mobile devices around their left/right edges respectively. β reports values between -180° and 180° , while γ reports values between -90° and 90° .

To access the current pitch angle β and roll angle γ through mobile browsers, the client-side component listens to the `deviceOrientation` event, which reports fresh orientation sensor readings whenever the device is rotated (i.e., tilted). Each time this event is fired, we calculate the δ between the previous rotation and the current rotation and use this to move the ball. If the rotation crosses an axis (e.g., goes from 79° to -79°), we catch these large changes and transition them to the small change they actually represent (e.g., 2°). The actual distance the ball moves is controlled by a ball speed parameter set by the server (currently, this is $\frac{1}{30}$ of the image for each degree of rotation). The movement of the ball itself is constrained to the boundaries of the image.

Each time the ball moves by at least a single pixel (modern devices are sensitive to record many sub-pixel moves), the new location is sent the server which checks whether the CAPTCHA has been completed—i.e., whether the distance between the center of the ball and the center of the keypoint is less than d pixels. If the two are within d pixels, the puzzle

¹SenCAPTCHA also supports moving the ball using touch or the mouse, but this is disabled by default.

is solved, and the client is notified of the puzzle’s completion. Note that users do not need to hover over the keypoint or click a button to finish the puzzle. Instead, they are just required to move the ball within the appropriate distance of the keypoint. In Section 6.7.3 we discuss other potential options for puzzle completion. If users fail to complete the CAPTCHA within a time limit (e.g., 1 minute), they are considered to have failed the CAPTCHA.

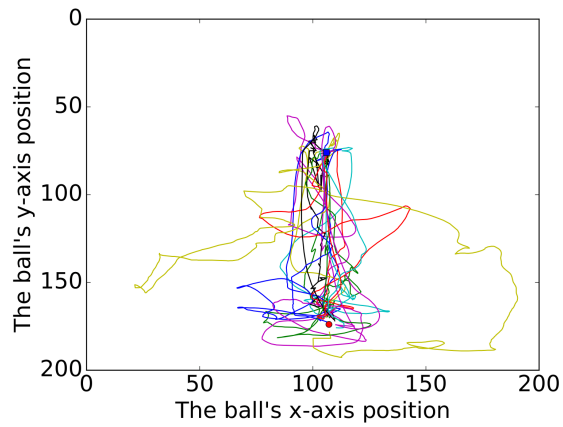
Trajectory Analysis

After the ball has been moved into the center of the animal’s eye, the server will check the history of ball locations (i.e., trajectory) to determine if the CAPTCHA was completed by humans or by bots. When completing SenCAPTCHA, we expect that humans will move the ball directly towards the animal’s eye. In contrast, the design of SenCAPTCHA makes it difficult for bots to determine the eye’s location with high confidence, therefore they will need to employ heuristic search strategies to find the eye from a set of possible locations. As such, it becomes possible to distinguish between human- and bot-based completion by comparing how closely the ball’s recorded trajectory matches the optimal trajectory (a straight line from the starting location to the animal’s eye). This comparison is made using a Dynamic Time Warping (DTW) algorithm [96, 91], with the attempt being considered human-generated if the calculated warping distance is below a given threshold.²

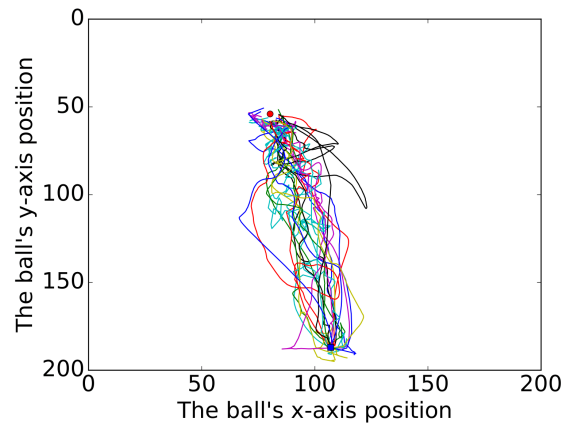
To confirm that human trajectories resemble the ideal trajectory, we completed hundreds of SenCAPTCHA puzzles within our research group and recorded those trajectories. As shown in Figure 6.4, while these trajectories do show deviation from the optimal path, they generally stay close to that path. This is different from bot-based approaches, which will have to use various search heuristics to find the eye. These results were further confirmed by the results from our user study (see Section 6.3).

To further differentiate between humans and bots, it may be possible to train a machine learning algorithm on the user’s trajectory and see if it looks human-like and resembles past attempts from the user (similar to what NoCAPTCHA does with user mouse movement). Such approaches could help detect bots even if they get lucky and correctly detect where the

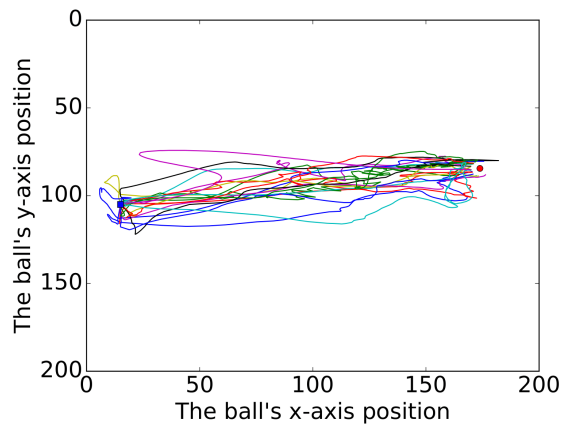
²Thresholds for our DTW-based classification are based on both the behaviors of bots (see Section 6.2) and empirical data gathered from our first user study (see Section 6.3).



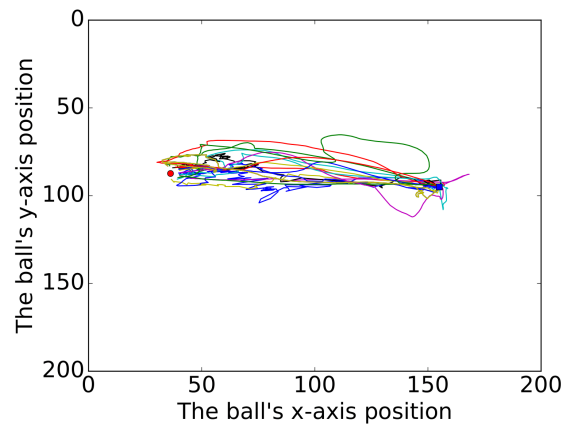
(a) Top to bottom



(b) Bottom to Top



(c) Left to right



(d) Right to left

Figure 6.4: Example human-generated trajectories

eye is. Still, we note that it is likely that an advanced adversary could trick the trajectory analysis if it knew the location of the image’s keypoint. As such, it is important to remember that SenCAPTCHA’s security is primarily based on the difficulty of finding the keypoint.

6.2 Security Analysis

In this section, we analyze possible attack approaches on SenCAPTCHA. We categorize possible attacks into brute-force attacks and machine learning based attacks. Specifically, brute-force attacks check every possible position for the eye until the correct position is found. Machine learning based attacks take advantage of visual recognition techniques to identify the eye’s position automatically.

6.2.1 Brute-force Attacks

The most intuitive brute-force attack is random guessing, which checks arbitrary locations one by one until reaching the target point. For each pair of the starting and ending point, we can estimate its distribution of random guessing DTW distances through sampling. Recall that SenCAPTCHA allows the ball launching from one position of the four corners, the four middle points at the edge, and the canvas center. Considering the symmetries of the canvas, we only need to study three representative starting locations: corner, middle point at edge, and center. Suppose the canvas size is 100 by 100, and the ball starts from the corner. Then we randomly select (x, y) as the target point, where both x and y are integers. Thus, we have 9,999 target points since the start location cannot be the target point at the same time.

For each (x, y) pair, we run the random guessing method 1000 times, generating 1000 trajectories randomly. We then calculate all DTW distances between each simulated trajectory and the optimal path, i.e., the straight line connecting the start location and the target location (x, y) . Next, we use the normal distribution $f(d|\mu, \sigma^2)$ to fit the probability density of DTW distances, where d represents the DTW distance, μ denotes the mean of DTW distances, and σ is the standard deviation. The cumulative distribution function of $f(d|\mu, \sigma^2)$ is denoted by $F(d)$. If we take the $\mu - 2.5\sigma$ as the threshold to distinguish humans from bots (i.e., if the DTW distance is larger than the threshold, it is classified as bots), the

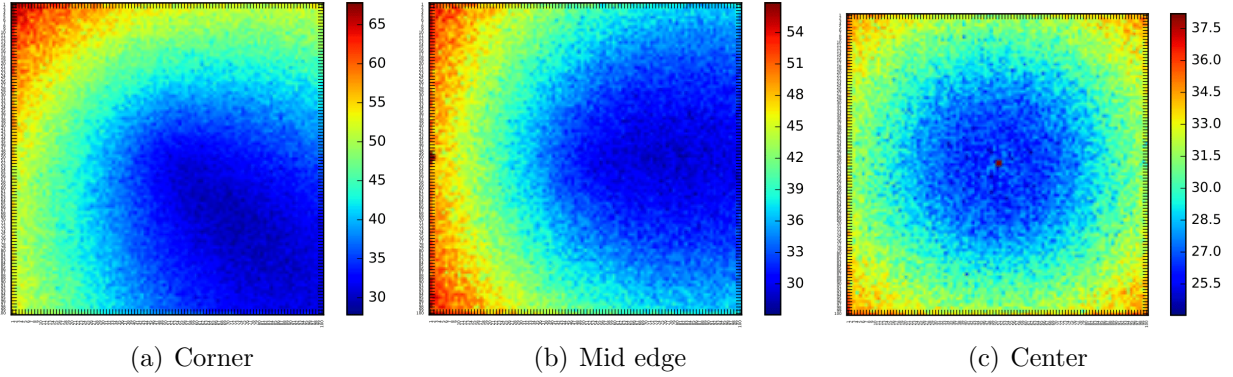


Figure 6.5: Values of $\mu - 2.5\sigma$ for random guessing attacks

successful attack rate will be $F(\mu - 2.5\sigma)$, which is less than 0.63%. Thus, for all the 9,999 target points with a starting location from the corner, we plot their $\mu - 2.5\sigma$ in Figure 6.5(a). Similarly, we simulate and calculate the $\mu - 2.5\sigma$ for starting location from the middle point of edges and the center, as shown in Figure 6.5(b) and 6.5(c) respectively. Base on the $\mu - 2.5\sigma$ values in Figure 6.5, a DTW threshold of 25 will be safe to distinguish bots from humans. We also checked 2385 trajectories by participants in Section 6.3 and found 93% responses were classified correctly.

6.2.2 Machine Learning Based Attacks

In this subsection, we deployed three machine learning based attacks, namely Scale-Invariant Feature Transform (SIFT) model [76], Triplet Interpolated Feature (TIF) model [125], and Interspecies Knowledge Transfer (IKT) model [92], to localize the animal facial keypoints on mutated images that are used in SenCAPTCHA.

Datasets and Evaluation Metric. In the TIF [125] and IKT papers [92], the authors published 50 labeled sheep images and 3717 labeled horse images respectively. Each sheep image contained a face bounding box and eight labeled keypoints, i.e., Left Outer Ear (LOE), Left Inner Ear (LIE), Right Outer Ear (ROE), Right Inner Ear (RIE), Left Eye (LE), Right Eye (RE), Nose (N), and Mouth (M). Each horse image was labeled five keypoints, i.e., Left Eye (LE), Right Eye (RE), Nose (N), Left Mouth Edge (LM), and Right Mouth Edge (RE).

We observed the released horse images had been cropped accordingly, and used the rectangle composed by four image edges as the bounding box. When measuring the detection success rate, we followed the same metric used in [125, 92]: If the Euclidean distance between the detected keypoint and its corresponding ground-truth keypoint was less than 10% of the face bounding box size, it was regarded as a success. We evaluated SIFT on both datasets, TIF on the sheep dataset, and IKT on the horse dataset.

SIFT Attacks. As SIFT is robust to uniform scaling, orientation, and illumination changes [76], it has been widely used for object recognition. We applied SIFT on both the sheep and horse datasets to detect facial keypoints. For each dataset, we calculated the average detection success rate for three types of mutated (zoomed, rotated, and tile) images. Considering the relatively small size of the sheep dataset, we generated ten zoomed and tile images randomly for each sheep image to expand the dataset. Note that we did not perform data augmentation on rotated images because SIFT is invariant to orientation. The average detection success rates of SIFT on the sheep and horse datasets are illustrated in Table 6.1(a) and Table 6.1(c) respectively. All success rates for any facial keypoints on the two datasets are below 10%. The lowest average success rates for all eight sheep keypoints and five horse keypoints are lower than 7.3% and 3.6% respectively (see the last columns in Table 6.1(a) and Table 6.1(c).) These low success rates indicate SenCAPTCHA is able to resist SIFT-based attacks with a high probability.

TIF Attacks. The TIF method extracts triplet interpolation features and calculates a shape-indexed feature using three anchor landmarks [125]. TIF is sensitive to the given face bounding box when identifying keypoints and it will not work effectively if we intentionally crop partial animal faces due to the incorrectly computed bounding box, as shown in Figure 6.6(b). When an image is rotated or tiled randomly, TIF fails to detect facial keypoints as well. Examples are shown in Figures 6.6(a) and 6.6(c).

Similar to data augmentation adopted in SIFT attack analysis, we generated ten randomly zoomed, rotated, and tile images for each sheep image to increase the diversity of data. Table 6.1(b) illustrates keypoint detection success rates (% of predicted keypoints whose Euclidean distance to the corresponding ground-truth keypoint is less than 10% of the face bounding box size) regarding eight different facial keypoints on three types of image

Table 6.1: Success rates for SIFT-, TIF-, and IKT-based animal keypoint detection

(a) SIFT-based detection on the sheep dataset

	LOE	LIE	LE	RE	RIE	ROE	N	M	ALL
Zoom	0.8%	5.5%	8.6%	9.6%	5.8%	3.0%	6.6%	5.6%	7.3%
Rotate	0.7%	1.1%	1.0%	1.3%	1.2%	0.8%	1.4%	1.2%	1.1%
Tile	3.3%	4.9%	4.2%	4.7%	4.4%	3.2%	5.3%	4.9%	4.4%

(b) TIF-based detection on the sheep dataset

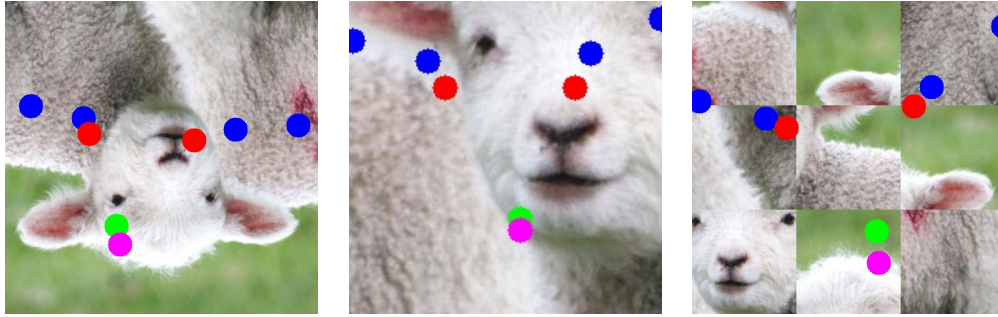
	LOE	LIE	LE	RE	RIE	ROE	N	M	ALL
Zoom	1.0%	6.0%	6.8%	6.6%	4.4%	1.0%	7.0%	4.2%	4.6%
Rotate	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.2%	0.0%	0.0%
Tile	15.2%	16.6%	14.4%	12.6%	14.0%	12.6%	9.2%	10.4%	13.1%

(c) SIFT-based detection on the horse dataset

	LE	RE	N	LM	RM	ALL
Zoom	3.6%	3.0%	2.8%	2.6%	2.9%	3.1%
Rotate	3.8%	3.6%	3.8%	3.5%	3.3%	3.6%
Tile	3.1%	3.0%	3.3%	3.1%	3.4%	3.2%

(d) IKT-based detection on the horse dataset

	LE	RE	N	LM	RM	ALL
Zoom	9.4%	6.1%	11.6%	10.0%	11.5%	9.2%
Rotate	2.2%	2.1%	0.5%	1.4%	2.0%	1.5%
Tile	2.8%	2.6%	3.4%	4.2%	4.0%	3.4%



(a) Rotation

(b) Zoom

(c) Tile

Figure 6.6: TIF attacks. ● (eye), ● (nose), ● (ear), ● (mouth)

mutations. The average success rates on both zoomed and rotated images are below 5%. Although the average success rate of tiles is above 10%, the tolerance in SenCAPTCHA (ranging from 2% to 3% of the bounding box size) is much smaller than the failure detection threshold—i.e., 10% of the bounding box size. This implies that the true success rate when using a TIF-based attack against SenCAPTCHA would be lower than 10% in practice.

IKT Attacks. As a deep transfer learning model, IKT achieves state-of-the-art animal facial keypoint detection performance by warping the animal face shape to make them more human-like, and then adapting the pre-trained human keypoint detector [92]. However, due to the assumption of the known face bounding boxes, IKT cannot deal with the animal images with partial faces, as shown in Figure 6.7(b). In addition, Figures 6.7(a) and 6.7(c) show that IKT’s performance is affected dramatically by rotations and misplaced keypoints.

For each image mutation, we first applied it on the horse dataset, and then trained a new IKT model from scratch. We used the same testing images that were used in the IKT paper [92] for evaluation. Note that the testing images were not unseen by our trained IKT models, although it might improve the detection accuracy. Table 6.1(d) summarizes the success rates regarding five different facial keypoints for three types of image mutations. It is interesting to note that the average success rate on zoomed images is 9.2%, which is much higher than that of rotated images (1.5%) and tiling (3.4%). This high success rate on zoomed images implies one limitation of IKT: in order to identify the angle of interest needed to build a mapping to human faces, IKT requires at least three keypoints existing within the bounding box, which indicates the impossibility of training IKT on images with less than three keypoints.

6.3 First Usability Study—Methodology

We conducted an IRB-approved, Amazon Mechanical Turk-administered usability study of SenCAPTCHA. In this study, we measured overall impressions regarding SenCAPTCHA as well as how various parameters affect SenCAPTCHA’s perceived usability and time to complete: four image manipulations, three puzzle tolerances, and nine starting positions.

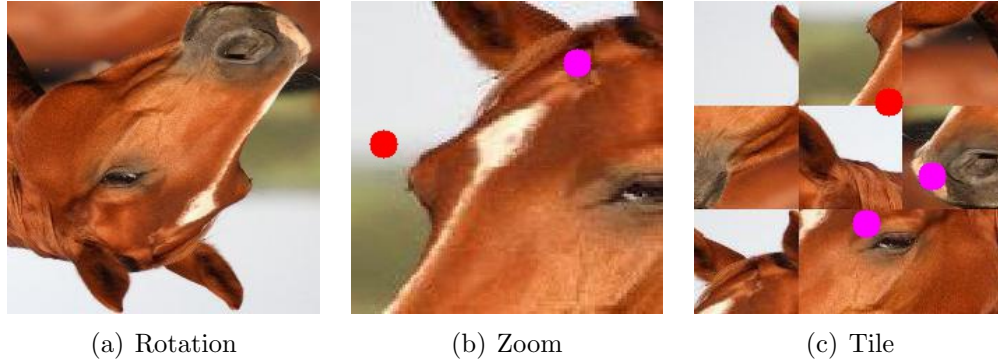


Figure 6.7: IKT attacks. ● (eye), ● (nose), ● (mouth)

The study ran over the course of five hours on October 9th, 2019. In total, 308 participants completed our usability study, with data from 270 of them used in our analysis.

6.3.1 Study Setup

Participants took 1–15 minutes to complete the study, with an average time of 3 minutes. Participants were paid \$1.25 USD for their participation. Before beginning the study, participants read an IRB-approved consent statement that informed them that they would need to use a mobile phone or tablet to complete the study. The survey itself also checked whether participants were using a mobile phone or tablet and would not let them complete the study if they were not.

After consenting to be a part of the study, participants were then asked a set of simple demographic questions (gender and age). They were also asked how often they check their mobile phones. The survey software also automatically gathered information about the user’s mobile device (e.g., OS version, browser version, screen resolution).

After providing this information, users were told that they would complete a series of where they “will be shown a picture of an animal and will tilt your phone to move a red ball into the middle of the animal’s eye.” On the page with these instructions, they were given a chance to practice moving a ball around an image. When they were ready to continue, they could click the next button.

At this point, participants were randomly assigned two different treatments:

1. Image mutation: no mutation, rotation, zoom and translate, or tile.
2. Puzzle tolerance: 0.02, 0.025, 0.03

Participants then completed nine SenCAPTCHA puzzles. The base image used in each puzzle was randomized, but we ensured that participants were never shown the same image more than once in the study. Each of the nine images used a different starting position for the ball, with the order of these starting images randomized. For the image mutations, all parameters were randomized each time (see Section 6.1).

Participants were given one minute to solve each CAPTCHA. If they did not solve the CAPTCHA within that minute, they were shown a message (using JavaScript’s `alert` function) telling them that the puzzle had timed out. When the click “OK” to this message, the next puzzle was automatically started. The time it took participants to complete the CAPTCHA was recorded along with the trajectory used to complete the CAPTCHAs was recorded for each puzzle, including whether the user timed out.

After completing the nine puzzles, participants were told that the puzzle they just completed was intended to be used as a CAPTCHA. This was done so that users could contextualize their answers in the remainder of the study, because feedback for SenCAPTCHA as a fun puzzle is likely to be very different than feedback for SenCAPTCHA as a CAPTCHA. After being informed of SenCAPTCHA’s intended use, participants were asked to complete the ten questions from the System Usability Scale (SUS) [15, 14]. We chose to use the SUS scale for several reasons. First, SUS has been shown to be a good indicator of perceived usability [115], is consistent across studies [95], as has reported norms that we can compare our results against [98].

Finally, participants were asked if they had any other feedback regarding SenCAPTCHA. After completing the survey, participants received a code that they would input into the Amazon MTurk site to get paid.

6.3.2 Data Cleanup

In total, 308 participants completed the survey, but several were unable to get SenCAPTCHA working. Of the participants, 266 completed every puzzle, 2 participants completed all but one puzzles, 2 completed all but two puzzles. The remaining 38 participants failed to complete any puzzles. Based on feedback from these 38 participants, we learned that in many cases SenCAPTCHA would simply not run on their devices. There was a variety of reasons for this: (a) in Safari on iOS 13, by default websites are not allowed to use sensor data, (b) some distributions of Android do not report screen orientation events to the browser if screen rotation has been locked, (c) and other distributions always refuse to provide sensor data to the browser (likely tied to a distribution-specific setting). As we were using this study to evaluate SenCAPTCHA, we remove data from the 38 participants who were unable to ever use SenCAPTCHA, but did include the data from participants where there was only a problem once or twice, leaving us with 270 participants for which we report results. For those 270 participants, we reported all available data, though for the four participants that had trouble completing SenCAPTCHA at least once, we only included the completion times for the puzzles where SenCAPTCHA worked for them.

6.3.3 Qualitative Analysis

To analyze the qualitative feedback provided by users, we used an open coding approach based on the constant-comparative method [39]. Two coders together reviewed every response, assigning one or more codes to describe the sentiment expressed in the response. Codes were only assigned if both reviewers agreed on the assignment. At the end of the coding process, the coders made a second pass over the data to merge related codes and ensure consistent usage across the data set. They also selected representative quotes to be used to help describe each code.

6.3.4 Demographics

Participants in our study were pulled from Amazon Mechanical Turk. Other than the restriction that participants must complete the study with a smartphone or tablet, there

were no other limitations on who could participate in the study. Demographics from our study are given in Table 6.2.

6.3.5 Limitations

While we informed participants that SenCAPTCHA was intended to be used as a CAPTCHA, we did not actually have users test it in this capacity. As such, the results might not fully reflect SenCAPTCHA’s true usability. While such a measurement was sufficient for our intended purposes, future work could try SenCAPTCHA in-situ to see if reactions change.

6.4 First Usability Study—Results

In this section, we report on the System Usability Scale (SUS) scores for SenCAPTCHA, the time taken to complete SenCAPTCHA puzzles, and qualitative feedback. Quantitative results are summarized in Table 6.3. We use a one-way ANOVA as our omnibus test with Tukey’s as our post-hoc pairwise statistical test. Where appropriate, we have used a Bonferroni correction to adjust for family-wise error in our statistical tests.

6.4.1 System Usability Scale

The differences between mutations ($F(3, 266) = 0.52, p = 1.0$) and tolerances ($F(2, 267) = 0.27, p = 1.0$) were not statistically significant. SUS scores from India are lower than scores from other countries (omnibus— $F(2, 267) = 96.03, p < 0.001$, India vs. USA— $p < 0.001$, India vs. Other— $p < 0.001$, USA vs. Other— $p = 0.87$). We initially hypothesized that this might be due to individuals in India having older devices but found no support for this in our data. Also, there is a statistically significant difference for SUS scores based on age ($F(4, 265) = 10.47, p < 0.001$), with users 18–34 giving lower scores than users 35–64 (all pairwise tests within these groups are not statistically significant). Finally, there is a statistically significant difference for scores based on mobile phone usage ($F(5, 264) = 3.38$,

Table 6.2: Demographics for first usability study

Category	Demographic	Count	%
Gender	Male	170	63.0%
	Female	100	37.0%
	Prefer not to say	0	0.0%
Age	18–24	55	20.4%
	25–34	145	53.7%
	35–44	50	18.5%
	45–54	12	4.4%
	55–64	8	3.0%
	Prefer not to say	0	0.0%
	65 years or older	0	0.0%
Mobile Phone Usage	Constantly	31	11.5%
	6–15 times an hour	32	11.9%
	3–5 times an hour	71	26.3%
	1–2 times an hour	86	31.9%
	Less than once an hour	47	17.4%
	Prefer not to say	3	1.1%
Country	India	139	51.5%
	USA	122	45.2%
	Other	9	3.3%

Table 6.3: Quantitative results from the first usability study

Category	Value	Count	Mean	Mdn	SD	Mean	Mdn	SD
			SUS			Time (seconds)		
Overall		270	66	61	19	4.7	3.7	3.5
Mutation	None	70	67	63	20	4.6	3.4	3.1
	Rotate	68	64	59	18	4.9	3.5	5.0
	Zoom	70	66	61	20	4.6	3.9	2.5
	Tile	62	64	60	19	4.8	4.0	2.7
Tolerance	0.02	92	65	60	19	5.5	4.1	4.8
	0.025	105	67	63	20	4.1	3.5	2.2
	0.03	73	64	58	18	4.5	3.5	2.7
Age	18–24	55	58	58	19	4.2	3.4	3.7
	25–34	145	63	58	19	5.0	3.8	3.9
	35–44	50	79	74	18	4.2	3.7	1.0
	45–54	12	81	81	15	4.3	4.2	1.5
	55–64	8	87	89	8	5.8	4.7	3.3
Mobile Phone Usage	Constantly	31	71	65	19	3.6	3.3	1.6
	6–15 times an hour	32	73	75	19	3.9	3.5	2.4
	3–5 times an hour	71	61	53	20	4.4	3.5	2.8
	1–2 times an hour	86	66	63	18	5.2	4.0	3.5
	Less than once an hour	47	63	58	19	5.1	4.0	5.0
	Prefer not to say	3	42	40	8	11.5	11.2	2.3
Country	India	139	53	50	11	5.2	4.0	3.9
	USA	122	78	84	18	5.6	3.6	7.8
	Other	9	81	90	17	4.1	3.5	2.2

$p = 0.03$), though post-hoc testing finds only a single statistically significant difference between “6–15 times an hour” and “3–5 times an hour” ($p = 0.04$).

Comparing our SUS results to norms reported by Sauro and Lewis [98], we see that the SUS score for SenCAPTCHA falls in line with the mean norm reported Sauro and Lewis. This is encouraging as we would expect CAPTCHAs to be annoying and fall below the mean norm. Additionally, we note that for older participants and participants not from India, the average SUS scores are much higher and fall between the 90th and 97th percentile for SUS scores, a truly great achievement.

6.4.2 Time to Complete

The only statistically significant difference in completion time is for mobile phone usage ($F(5, 264) = 4.03$, $p < 0.01$), inasmuch as users who “Prefer not to say” are slower than all other groups (all pairwise tests with “Prefer not to say” are statistically significant, no others are). The differences between mutations ($F(3, 266) = 0.12$, $p = 1.0$), tolerances ($F(2, 267) = 4.302$, $p = 0.07$), ages ($F(4, 265) = 1.12$, $p = 1.0$), and countries ($F(4, 265) = 3.55$, $p = 0.15$) are not statistically significant.

6.4.3 Qualitative Feedback

Just under two-thirds of participants left some feedback in the optional feedback field, though around half of this feedback was about the survey—i.e., generic comments about it being a good survey. In general, participants had positive perceptions of SenCAPTCHA: 17 (6.3%) participants indicated that it was enjoyable, 11 (4.1%) that it was easy, 8 (3.0%) that it was interesting, and 58 (21.5%) with another generic positive responses. 5 (1.9%) participants explicitly stated that it was better than other CAPTCHA systems. For example, one participant stated,

“This would be SO much better compared to the traditional captchas. I love it, seriously whoever came up with, this great job I really hope it is integrated on sites very soon. I would love to see traditional ones be completely replaced by

this. It's much less annoying, and it seems a lot more consistent. Thank you for letting me be a part of this."

There were very few negative comments. 5 (1.9%) participants had a generic comment about not liking SenCAPTCHA and 8 (3.0%) participants worried that SenCAPTCHA might not work in some situations: for users with accessibility needs—2 (0.7%) participants, on older phones—1 (0.4%) participant, if screen rotation was locked—1 (0.4%) participant, the sensor might fail—1 (0.4%) participant, and unspecified worry—3 (1.1%) participants. For example, one participant said,

"I found this easy to use but I could see some people, especially older people, having a hard time with it."

5 (1.9%) participants also mentioned needing to change their iOS settings to get SenCAPTCHA working, which helped us track down and fix it in our second user study.

6.5 Second Usability Study—Methodology

In our first study, we found that users generally had positive attitudes towards SenCAPTCHA and that they completed the CAPTCHAs very quickly. To understand how SenCAPTCHA's performance compares to other existing CAPTCHA schemes, we conducted an IRB-approved, Amazon Mechanical Turk-administered, within-subject usability study test of five different CAPTCHA systems. The study ran over the course of seven hours on November 12th, 2019. In total, 202 participants completed our second usability study.

The five systems we tested are,

- **SenCAPTCHA.** Based on our security evaluation, we used the rotation and tile image mutations. Based on our first usability study, we set the difficulty parameter to 0.25.
- **Text-based.** We used Securimage [89], an open-source free PHP CAPTCHA script, to generate text-based CAPTCHA challenges. To complete the CAPTCHA, users had to enter the two words displayed. These words were pulled from a dictionary consisting

of 1,488 common six-letter English words. They were also slightly distorted with added background noise.

- **Audio-based.** We again used Secureimage to generate audio-based CAPTCHA challenges. To complete the CAPTCHA, users had to enter six random alphanumeric characters that were read aloud. The audio sample had background noise inserted.
- **Image-based.** We implemented an image-based CAPTCHA that displays 16 images to users in a four-by-four grid and asks them to select images matching some noun (e.g., horses, dogs). Each CAPTCHA contains three to six different types of images, with the number of target images ranging from one to five. Images were pulled from the CIFAR-10 dataset and filtered to select the clearest images.
- **Video-based CAPTCHA:** We implemented a video-based CAPTCHA that displayed five capital letters moving across the image from left to right. The letters loop around every 2.7 seconds and rotate independently within a range of 30 degrees.

In each case, we implemented the non-SenCAPTCHA systems to be of comparable difficulty (i.e., same level of distortion or obfuscation) to what we have observed online. In each case, we also erred on the side of making these CAPTCHAs easier to provide a fair comparison to SenCAPTCHA. For consistency, we had all the systems allow users to refresh the CAPTCHA and get a new challenge. We also standardized the user interface for each CAPTCHA to be as similar as possible. If users gave an incorrect answer for the CAPTCHA, they were informed of the mistake and a new challenge was generated for them.

6.5.1 Study Setup

Participants took 3–27 minutes to complete the study, with an average time of 9.5 minutes. Participants were paid \$1.25 for their participation. Before beginning the study, participants read an IRB-approved consent statement, which informed that they would need to use a mobile phone or tablet to complete the study. The survey itself also checked whether participants were using a mobile phone or tablet and would not let them complete the study if they were not.

After providing this information, participants completed tasks for each of the five different types of CAPTCHAs (order of the different types was randomized). For each type of CAPTCHA, participants completed three consecutive CAPTCHA instances, with the option to skip a given instance if it took them more than one minute to complete. After completing all three instances, participants provided feedback on their experience using the After-Scenario Questionnaire (ASQ) [67], which consists of three questions answered using a seven-point Likert scale:

Q1: Overall, I am satisfied with the ease of completing this type of CAPTCHA

Q2: Overall, I am satisfied with the amount of time it took to complete this type of CAPTCHA

Q3: Overall, I am satisfied with the support information (online help, messages, documentation) when completing this type of CAPTCHA

After completing all five types of CAPTCHAs, participants were asked to indicate which of the five was their favorite and why it was their favorite. They were also asked to indicate which of the five, if any, they wouldn't want to use in practice and why they wouldn't want to use them. For each of these questions, the ordering of the systems was randomized. Finally, they were given a chance to provide any additional feedback they might want to give.

6.5.2 Data Cleanup

In total, 202 participants completed the study, though for 36 participants the sensor API failed to initialize leaving them unable to use SenCAPTCHA. As with our first usability study, we are unsure of what causes this, though it looks like it might be related to device-specific limitations on the use of the orientation sensor by websites.³ For those 36 participants, we do not report their SenCAPTCHA ASQ scores or timing data nor use this data in statistical (repeated-measure) tests as these participants didn't use SenCAPTCHA and as such can't compare it to other CAPTCHA systems. We also do not report on their favorite system or most disliked systems, as the purpose of these questions was to

³For example, iOS 13.1 disallows the use of the orientation sensor by websites unless this is changed in the device's settings. iOS 13.2 added an API to request that permission programmatically (which we do).

compare performance against SenCAPTCHA. We do report their ASQ scores and timing for the other CAPTCHAs, as we believe these data points are unaffected by the problems with SenCAPTCHA and can serve as meaningful norms to be compared against by future CAPTCHA research.

6.5.3 Qualitative Analysis

To analyze the qualitative feedback provided by users, we used the same open coding approach based on the constant-comparative method [39] as for the first user study.

6.5.4 Demographics

Participants in our study were pulled from Amazon Mechanical Turk. Other than the restriction that participants must complete the study with a smartphone or tablet, there were no other limitations on who could participate in the study. Demographics from our study are given in Table 6.4.

6.6 Second Usability Study—Results

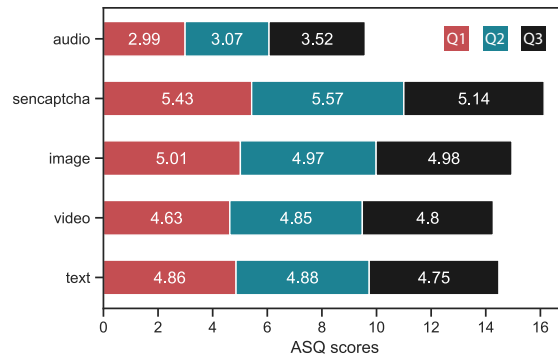
In this section, we report on After-Scenario Questionnaire (ASQ) scores, time taken to complete each system, participants’ favorite and most disliked systems, and qualitative feedback. Results are summarized in Figure 6.8. We used a one-way, repeated measures ANOVA as our omnibus test (except where noted) with Tukey’s as our post-hoc pairwise statistical test. For brevity, we only report on the significance of these pairwise tests, not their individual p-values.

6.6.1 After-Scenario Questionnaire (ASQ)

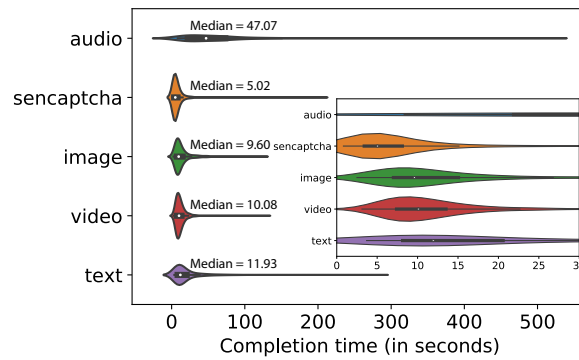
SenCAPTCHA receives the highest rating of any of the CAPTCHA systems (16.14), with audio-based CAPTCHAs receiving the lowest score (9.58). There was a statistically significant omnibus effect for ASQ scores (two-way, repeated measures ANOVA— $F(4, 2907) = 123.61$, $p < 0.001$), with all pairwise tests between SenCAPTCHA and

Table 6.4: Demographics for second usability study

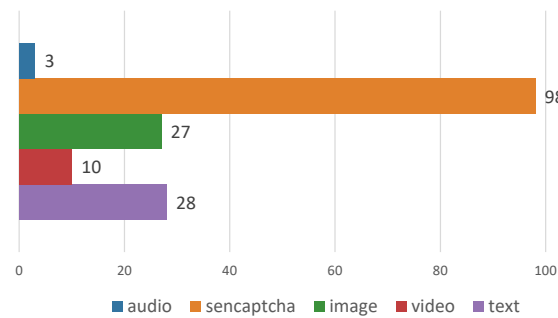
Category	Demographic	Count	%
Gender	Male	127	62.9%
	Female	74	36.6%
	Prefer not to say	1	0.5%
Age	18–24	16	7.9%
	25–34	132	65.3%
	35–44	37	18.3%
	45–54	11	5.4%
	55–64	3	1.5%
	Prefer not to say	1	0.5%
	65 years or older	2	1.0%
Mobile Phone Usage	Constantly	28	13.9%
	6–15 times an hour	31	15.3%
	3–5 times an hour	61	30.2%
	1–2 times an hour	57	28.2%
	Less than once an hour	25	12.4%
	Prefer not to say	0	0.0%
Country	India	73	36.1%
	USA	117	57.9%
	Other	12	5.9%



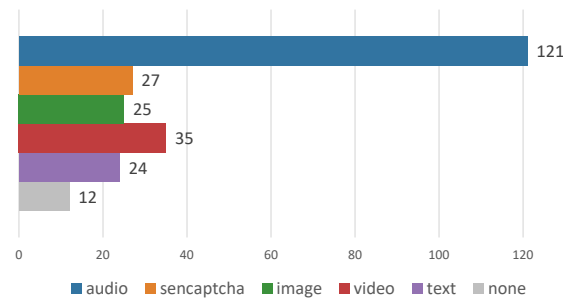
(a) Mean ASQ score



(b) Mean completion time



(c) Favorite system



(d) Most disliked system

Figure 6.8: Quantitative results from the second usability study

other CAPTCHAs being statistically significant. All pairwise tests between Audio and other CAPTCHAs were also statistically significant. None of the other pairwise tests were statistically significant.

6.6.2 Time to Complete

SenCAPTCHA had the lowest average completion time (5.02 seconds), while audio had the longest average completion time (47.07 seconds). The remaining systems had similar average completion times (around 10 seconds). These differences were statistically significant ($F(4, 198) = 288, p < 0.01$), with all but two pairwise tests (SenCAPTCHA vs. video-based and image-based vs. video-based) being statistically significant.

6.6.3 Favorite System

Of the 166 participants that used all five CAPTCHAs, 98 (59.0%) indicated that SenCAPTCHA was their favorite system. This significantly outperformed the other four CAPTCHAs: text-based—28 (16.9%) participants, image-based—27 (16.3%) participants, video-based—10 (6.0%) participants, audio-based—3 (1.8%) participants. This difference was statistically significant (χ^2 test— $\chi^2 = 134, p < 0.001$)

Participants strongly disliked the audio-based CAPTCHAs, with 121 (72.9%) expressing this sentiment. Participants' dislikes in regard to other CAPTCHAs were evenly split: video-based—35 (21.1%) participants, SenCAPTCHA—27 (16.3%) participants, image-based—25 (15.1%) participants, text-based—24 (14.5%) participants. 12 (7.2%) participants indicated that they did not dislike any of the CAPTCHA systems. These differences were statistically significant (χ^2 test— $\chi^2 = 197, p < 0.001$), with audio being the only pairwise statistically significant difference ($p < 0.001$).

6.6.4 Refreshes, Failures, and Skips

Participants rarely refreshed the provided CAPTCHAs (means: 0.00–0.08 refreshes/attempt, median: 0). Failures were a little more common, though still uncommon for CAPTCHAs other than audio-based (mean failures/attempt: audio—1.66, SenCAPTCHA—N/A,

image—0.46, video—0.11, text—0.59). Skipping (available after trying for one minute) was also rare, except for audio-based CAPTCHAs (mean number of instances skipped: audio—0.68, SenCAPTCHA—0.00, image—0.02, video—0.00, text—0.07).

6.6.5 Qualitative Feedback

Most participants provided feedback regarding their favorite and least favorite systems. In general, 121 (72.9%) participants indicated that their favorite system was easy, 44 (26.5%) that it was fast, and 23 (13.9%) that it was enjoyable.

SenCAPTCHA received the lion’s share of positive comments: easy—54 (32.5%) participants, fast—27 (16.3%) participants, enjoyable—21 (12.7%) participants, novel—10 (6.0%) participants, interesting—7 (4.2%) participants. Additionally, 5 (3.0%) participants indicated that they liked how SenCAPTCHA did not require any typing, and 4 (2.4%) indicated that SenCAPTCHA was mobile friendly.

For the remainder of the systems, the following were the more common answers (more than 5 responses):

- 8 (4.8%) participants though that image-based CAPTCHAs were fast.
- 26 (15.7%) participants thought image-based CAPTCHAs were easy, 26 (15.7%) though the same about text-based CAPTCHAs, and 9 (5.4%) though so about video-based CAPTCHAs.
- 6 (3.6%) participants felt that image-based CAPTCHAs were understandable.

In regards to things they disliked about the CAPTCHA systems, the overwhelming majority of feedback was from audio-based CAPTCHAs: hard to hear or understand—66 (39.8%), too hard in general—24 (14.5%), too slow—20 (12.0%), can’t be used in a public place—16 (9.6%), too much background noise making it difficult to understand—12 (7.2%), and annoying—11 (6.6%). For example, one person stated,

“I could not hear the letters and numbers with all the static in the background; plus the computerized voice was awful. In short; I hated it and if I ever encountered it in life; I would just quit that page or application.”

The only other comment appearing more than five times was that 14 (8.4%) participants felt that SenCAPTCHA was hard. One participant indicated that,

“It was so annoying and not easy to do if you have a squirmy baby on your lap.”

Participants largely ignored the final optional feedback field, and there is no meaningful data to report in that regard.

6.7 Discussion

In this section, we summarize the lessons learned, discuss the viability of deploying SenCAPTCHA in production, and identify future research directions.

6.7.1 Lessons Learned

The results from our research are rather straightforward. Users enjoyed SenCAPTCHA, with nearly two-thirds of participants preferring SenCAPTCHA to other CAPTCHA systems. Participants indicated that they found SenCAPTCHA to be easy and enjoyable, demonstrating that SenCAPTCHA achieved its three design goals. Our results also demonstrate that users are open to sensor-based CAPTCHAs, which was unclear before this research.

The fact that users choose SenCAPTCHA over existing schemes also indicates that users are not satisfied with the current state of CAPTCHAs. While this might be intuitive, it is still illustrative to be demonstrated in practice. In particular, our research shows that users especially dislike audio-based CAPTCHAs. This is somewhat surprising as audio-based CAPTCHAs are commonly found as a backup method for other types of CAPTCHAs. This suggests that it might be worthwhile exploring different backup CAPTCHAs.

Need for Additional CAPTCHA Research

An interesting omission in the results for our second user study was that no users indicated that they no longer saw a need for CAPTCHAs or that they did not need SenCAPTCHA because they did not encounter CAPTCHAs on their mobile devices. This

suggests that even with the emergence of NoCAPTCHA, that users are still encountering CAPTCHAs, including on their mobile devices. Taken together with the positive reception of SenCAPTCHA, this suggests that there remains a need for additional research into building better CAPTCHAs. Not only are users likely still encountering CAPTCHAs, but they seem to be very open and interested in new CAPTCHAs that are easier, faster, and more enjoyable than current solutions. This runs counter to the current trend in the field, which has seen a sharp decline in the amount of research on usable and secure CAPTCHA schemes.

Relationship to Prior Work

Our results regarding the relative ease of use and completion times for text-based, image-based, and video-based CAPTCHAs are in line with prior work. For example, Reynaga et al. [94] evaluated Picatcha [83], an image-based CAPTCHA and showed that completion of this CAPTCHA was faster than text-based CAPTCHAs, similar to our own results.

One notable contrast between our results and past results is in relationship to work by Brodić et al. [13]. In their research, they found that young Internet users were able to solve CAPTCHAs more quickly than older users. Based on data from both our user studies, this pattern does not seem to hold true for SenCAPTCHA, which was equally quick for users to complete regardless of age.

Implementation Details for Audio-Based CAPTCHAs

While implementing our audio-based CAPTCHA using the Secureimage library, we noticed that when we programmatically started the playback of the audio file using JavaScript (in response to clicking a play button), that it was difficult to hear the first few fractions of a second of the playback. While this might seem minor, it significantly decreased the success rate for solving the audio-based CAPTCHAs. In contrast, if we used the native playback interface to begin playback, then these first few fractions of a second were audible and success rates were much higher (up to 50% higher). We are not sure what made this difference but mention it here as a potential problem with the design of the Secureimage library, as the programmatic playback approach is its default.

6.7.2 SenCAPTCHA in Production

As with most research, the current version of SenCAPTCHA is not production-ready, but rather a proof-of-concept implementation used to demonstrate the viability of a sensor-based CAPTCHA. To prepare SenCAPTCHA for deployment, it would be necessary to address the following challenges.

First, it would be necessary to have a larger corpus of images. Leveraging the mutations, we are able to generate a large number of puzzles from a single image—for example, using the tile mutation, each image from our corpus can be used to generate $9! = 362,880$ puzzles. Still, the base image corpus should likely number between several hundred to a couple thousand. This would provide a rich data source from which to generate mutated images and would increase the difficulty of trying to reverse the mutations. Based on our experience of generating the current image corpus, we estimate that this process would take no more than 100 hours, something that is a very reasonable time frame for the development of production-ready software.

Second, work needs to be done to understand why SenCAPTCHA sometimes fails to get data from the sensor API. In our second user study, 36 (21.7%) participants were unable to use SenCAPTCHA due to this problem. Additionally, development work is needed to locate this bug and fix it. It might also be necessary to collaborate with browser and mobile phone vendors to ensure that the sensor is available for SenCAPTCHA’s use. Alternatively, SenCAPTCHA could be deployed as is, but fallback to another CAPTCHA if it is unable to access the sensor. If SenCAPTCHA were to become popular with this type of deployment, it would incentivize vendors to directly support sensor-based CAPTCHAs.

Third, it would be necessary to ensure that the current ball (a red ball with a black border) is easily visible when placed over all images in the corpus. If not, it might be necessary to calculate the color of the ball to use such that it will contrast with the underlying image.

6.7.3 Future Research Directions

In the remainder of this section, we identify future research directions for SenCAPTCHA.

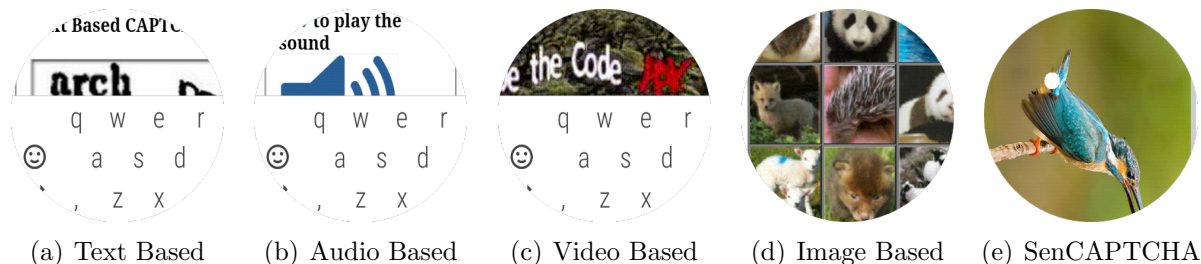


Figure 6.9: How current CAPTCHAs would appear on a smartwatch.

SenCAPTCHA on Smartwatches

We believe that as smartwatches continue to increase their functionality, the likelihood that users will encounter CAPTCHAs on their smartwatches will likewise increase. For example, (a) viewing a website that will only display its content after presenting a CAPTCHA to prevent content scraping; (b) submitting a comment (using dictation) to a website that uses a CAPTCHA to prevent comment spam; (c) voting in an online pool that uses CAPTCHAs to prevent ballot stuffing. In each of these cases, the websites may have a responsive UI that displays cleanly on a smartwatch but will still need to show a CAPTCHA to protect against bots who would pretend to be using smartwatches if CAPTCHAs were disabled for that form factor. While the frequency of CAPTCHAs on smartwatches will likely never match their frequency on desktops or mobile phones (where account creation and management are handled), it is still worthwhile to prepare for a possible (and we believe likely) future where CAPTCHAs are occasionally encountered on smartwatches.

We believe that SenCAPTCHA would be an ideal solution for CAPTCHAs on smartwatches. We illustrate the display of multiple CAPTCHA designs on a 46mm Moto 360 smartwatch in Figure 6.9. Observe that those CAPTCHAs relying on users' text input have a very bad performance, because the on-screen keyboard occupies more than half of the screen. Some sub-figures in image-based CAPTCHAs are hidden, which makes it hard for users to identify what the whole image looks like. On the other hand, users can still play SenCAPTCHA as it fits on the small screens more easily, and it does not require much screen space to display its content properly.

Future user studies of SenCAPTCHA running on smartwatches could help confirm our intuitions regarding SenCAPTCHA’s performance on smartwatches. It could also help identify requirements and issues that are unique to smartwatches, helping the design of future mobile-first CAPTCHAs.

Adversarial Machine Learning

Recent advances in adversarial machine learning have shown that it is possible to make small perturbations in images such that humans cannot detect the change, but machine learning classifiers are rendered unable to classify the image. Such techniques could also be applied to SenCAPTCHA allowing it to show images without any discernable mutations, further increasing its usability and security. This approach would not only benefit SenCAPTCHA, but other CAPTCHAs that also rely on image recognition being hard for machine learning (e.g., the image-based CAPTCHA from our user study).

SenCAPTCHA Using Mouse and Touch

Our implementation of SenCAPTCHA had support for moving the ball using mouse and touch, but we did not enable this functionality in our user studies as we were primarily interested in seeing how a sensor-based CAPTCHA would be received by users. During development, we extensively used the mouse-based ball movement for testing and found it to be extremely efficient and easy to use. Future research is needed to validate that users would prefer this modality, but it seems likely considering the positive reception of SenCAPTCHA on mobile devices.

It would also be possible to implement a version of SenCAPTCHA that allowed users to directly tap on the eye. We chose not to do this as we wanted to avoid the “fat finger” problem on small form factor devices, for which we think the sensor version of SenCAPTCHA is well suited. Nevertheless, future research could evaluate whether a tap-based version of SenCAPTCHA would work well on larger mobile devices such as tablets.

Alternative Completion Criteria

In the current implementation of SenCAPTCHA, the puzzle is completed as soon as the center of the ball and the target are within a set distance d . While this increases the speed and ease of solving SenCAPTCHA, it also somewhat limits its security as an attacker will solve the puzzle as long as the real target is on the path to the guessed target (modulo the fact that they have to pass the DTW check). We identify three alternate completion criteria that could be used to further increase the security of SenCAPTCHA. For each of these criteria, future research would be needed to establish the usability impact of adopting these approaches.

First, users could be required to hover on or near the keypoint once they have moved the ball to it. The amount of time spent hovering would likely be very short, likely no longer than half a second. This would ensure that the user is confident of the keypoint's position. With these criteria, it would likely be necessary to have the hover radius be greater than d to accommodate users' hands shaking during the hover period. Both the hover radius and timing would need to be established through additional user testing.

Second, instead of having users move a ball over a single target, we could ask them to move the ball over two different targets. After moving the ball over the second target, SenCAPTCHA would be completed immediately (i.e., without hover) just as it is now. The trajectory would then be examined to validate that the user had correctly changed trajectory after moving to the first keypoint. This would also increase the challenge of solving SenCAPTCHA by requiring an adversary not to detect just one, but two keypoints. The order in which the user would select the keypoints could likely be left up to the user. Care would need to be taken when selecting images to ensure that the keypoints are spaced a reasonable distance from each other.

Third, users could confirm that the ball is near the keypoint by clicking the screen. We choose to use a screen click as opposed to clicking a button because in our informal testing we found that asking a user to click a button causes them to rotate their screen to position their thumb over the button, whereas asking them to tap anywhere on the screen allows them to

complete the CAPTCHA with no additional wrist movement. Still, usability testing would be needed to confirm that this approach works in practice.

Using SenCAPTCHA to Create Training Data Sets

In Von Ahn et al.'s [120, 118] on ReCAPTCHA, they described how ReCAPTCHA could be used to have users label text in images, then using those labels to help digitize books. Such an approach is especially brilliant as it takes work that is otherwise meaningless to the end-user (CAPTCHAs help servers not users) and helps produce an outcome (book digitization) that may be of use to the end-user. The question naturally arises then if SenCAPTCHA could be used to label keypoints on images to similar help with research in computer vision.

Unfortunately, the current implementation of SenCAPTCHA does not lend itself to accomplish that goal. In contrast, the two alternative completion strategies listed above have the potential to be used to label keypoints. For the second (using two keypoints) and third (clicking) completion criteria, we can use unlabeled images and label the point that the user selects as the keypoint. Based on measurements from multiple users, we can calculate a more precise position for the keypoint. Still, care must be taken that only a small number of unlabeled images are presented to users, as the system has no way of verifying the correctness of the presented solution.

Chapter 7

Conclusion

This dissertation focuses on mobile location data analytics, privacy, and security studies. We model the geospatial distribution of individual apps via cellular data traffic, and analyze usages and privacy issues for shared locations on social networks. Then, we present homomorphic bloom filters to enhance the security of shared locations. Finally, SenCAPTCHA is proposed for mobile devices to fight against the bots that attack remote location servers and submit fake locations.

In the cellular location work, we propose a multi-level mixture of kernel density estimation (mlKDE) to profile the geospatial distribution of individual mobile apps. Taking a multi-granularity view on the data, we are able to find the optimal weights for each granularity level. Within each level, we can use the latitude-longitude information of cell towers, and gather the data volumes transmitted through that tower, to estimate the Gaussian kernel density. Such density estimates can concisely highlight where the users are collectively located, and serve as hierarchical components in the proposed mlKDE. Across multiple levels, the trained weights for each level inform us about the app’s behavior with respect to other apps within the same group. We demonstrate that the geospatial distribution profiling of apps can be smoothed using the aggregated data from the population and group levels. Our paper provides a general framework for modeling spatial time series data for individuals (e.g., apps, products) that naturally have hierarchies of granularity.

In the social network location work, we give four scenarios to show the usages of shared locations on Twitter. We first chase the 2017 Great American Eclipse on Twitter and

demonstrate the interstate and intrastate human mobility. Then, a geographic pattern of petitions for new emojis are visualized to show geo-tagged tweets are helpful to understand where petitions occur. Next, we use the scooter case study to show the scooter trends in English-speaking countries and the state-level and county-level deployment in the United States. Finally, we propose a geo-tagged COVID-19 Twitter dataset to understand the fine-grained public reactions.

In the homomorphic bloom filter work, we develop a distributed protocol that allows one party to determine, in a private and secure manner, whether or not the trajectory of a second party has an intersection with specific locations of interest. Our design is fully flexible, meaning that each user is able to specify what kind of datasets they would like to make visible, and be queried by other users. The methodology is based on developing a generalized set membership check approach, using an advanced data structure called the bloom filter. To demonstrate its feasibility and usability, we offer three working prototypes, which are implemented on the open-source homomorphic libraries. Our preliminary results illustrate the performance and overhead of the proposed approaches as well as the security of the protocol designs.

In the SenCAPTCHA work, we design and implement a mobile-first CAPTCHA using orientation sensors. SenCAPTCHA works by showing users an image of an animal and asking them to tilt their device to guide a red ball into the center of that animal’s eye. SenCAPTCHA is especially useful for devices with small screen sizes (e.g., smartphones, smartwatches). We describe the design of SenCAPTCHA and demonstrate that it is resilient to various machine learning based attacks. We also report two usability studies of SenCAPTCHA involving a total of 472 participants; our results show that SenCAPTCHA is viewed as an “enjoyable” CAPTCHA and that it is preferred by over half of the participants to other existing CAPTCHA systems.

Bibliography

- [1] Aharony, N., Pan, W., Ip, C., Khayal, I., and Pentland, A. (2011). Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing*, 7(6):643–659. [18](#)
- [2] Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C. A., and Strand, M. (2015). A guide to fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2015:1192. [13](#)
- [3] Baird, H. S., Moll, M. A., and Wang, S.-Y. (2005). Scattertype: a legible but hard-to-segment captcha. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 935–939. IEEE. [14](#)
- [4] Bakillah, M., Li, R.-Y., and Liang, S. H. (2015). Geo-located community detection in twitter with enhanced fast-greedy optimization of modularity: the case study of typhoon haiyan. *International Journal of Geographical Information Science*, 29(2):258–279. [4](#)
- [5] Barnes, T. J. and Wilson, M. W. (2014). Big data, social physics, and spatial analysis: The early years. *Big Data & Society*, 1(1):2053951714535365. [5](#)
- [6] Bendrath, R. (2009). Global technology trends and national regulation: Explaining variation in the governance of deep packet inspection. In *International Studies Annual Convention, February*, pages 15–18. [28](#)
- [7] Bilogrevic, I., Jadliwala, M., Joneja, V., Kalkan, K., Hubaux, J.-P., and Aad, I. (2014). Privacy-preserving optimal meeting location determination on mobile devices. *IEEE transactions on information forensics and security*, 9(7):1141–1156. [13](#)
- [8] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426. [7](#)
- [9] Bongard, M. (1970). *Pattern Recognition*. Spartan Books, Rochelle Park, NJ. [15](#)
- [10] Borge-Holthoefer, J., Rivero, A., García, I., Cauhé, E., Ferrer, A., Ferrer, D., Francos, D., Iniguez, D., Pérez, M. P., Ruiz, G., et al. (2011). Structural and dynamical patterns

- on online social networks: the spanish may 15th movement as a case study. *PloS one*, 6(8):e23883. [11](#)
- [11] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2012). (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM. [68](#)
- [12] Brakerski, Z. and Vaikuntanathan, V. (2014). Lattice-based fhe as secure as pke. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 1–12. ACM. [13](#)
- [13] Brodić, D., Amelio, A., and Janković, R. (2018). Exploring the influence of captcha types to the users response time by statistical analysis. *Multimedia Tools and Applications*, 77(10):12293–12329. [112](#)
- [14] Brooke, J. (2013). Sus: a retrospective. *Journal of usability studies*, 8(2):29–40. [97](#)
- [15] Brooke, J. et al. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7. [97](#)
- [16] Bursztein, E. (2012). How we broke the nucaptcha video scheme and what we proposed to fix it. See <https://www.elie.net/blog/security/how-we-broke-the-nucaptcha-video-scheme-and-what-we-propose-to-fix-it/>, Accessed March. [15](#)
- [17] Bursztein, E., Aigrain, J., Moscicki, A., and Mitchell, J. C. (2014). The end is nigh: Generic solving of text-based captchas. In *WOOT*. [14](#)
- [18] Casino, F., Domingo-Ferrer, J., Patsakis, C., Puig, D., and Solanas, A. (2015). A k-anonymous approach to privacy preserving collaborative filtering. *Journal of Computer and System Sciences*, 81(6):1000–1011. [12](#)
- [19] Chan, T.-Y. (2003). Using a test-to-speech synthesizer to generate a reverse turing test. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 226–232. IEEE. [14](#)

- [20] Chen, E., Lerman, K., and Ferrara, E. (2020). Covid-19: The first public coronavirus twitter dataset. *arXiv preprint arXiv:2003.07372*. [56](#)
- [21] Chew, M. and Tygar, J. D. (2004). Image recognition captchas. In *International Conference on Information Security*, pages 268–279. Springer. [15](#)
- [22] Choudhary, S., Saroha, R., Dahiya, Y., and Choudhary, S. (2013). understanding captcha: Text and audio based captcha with its applications. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6). [15](#)
- [23] Conti, M., Guarisco, C., and Spolaor, R. (2016). Captchastar! a novel captcha based on interactive shape discovery. In *International Conference on Applied Cryptography and Network Security*, pages 611–628. Springer. [15](#)
- [24] Coward, M. (2009). Deep packet inspection optimizes mobile applications. *EDN (Electrical Design News)*, 54(19):37. [28](#)
- [25] Datta, R., Li, J., and Wang, J. Z. (2005). Imagination: a robust image-based captcha generation system. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 331–334. ACM. [15](#)
- [26] Department, S. R. (2020). *Smartphone users worldwide 2016-2021*. [1](#)
- [27] Dong, C., Chen, L., and Wen, Z. (2013). When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800. ACM. [13](#)
- [28] Dredze, M., Paul, M. J., Bergsma, S., and Tran, H. (2013). Carmen: A twitter geolocation system with applications to public health. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*. [4](#)
- [29] Elson, J., Douceur, J. R., Howell, J., and Saul, J. (2007). Asirra: a captcha that exploits interest-aligned manual image categorization. In *ACM Conference on Computer and Communications Security*, volume 7, pages 366–374. Citeseer. [15](#)
- [30] EPIC (2018). *Location privacy issues and suggestions*. [5](#)

- [31] Ericsson (2014). Ericsson mobility report. <http://www.ericsson.com/res/docs/2014/ericsson-mobility-report-june-2014.pdf>. 2
- [32] Facebook (2019). *The Robots File of Facebook*. 4
- [33] Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144. 6, 13
- [34] Fu, Y., Xiong, H., Ge, Y., Yao, Z., Zheng, Y., and Zhou, Z.-H. (2014). Exploiting geographic dependencies for real estate appraisal: A mutual perspective of ranking and clustering. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1047–1056. ACM. 10
- [35] Gao, H., Liu, H., Yao, D., Liu, X., and Aickelin, U. (2010a). An audio captcha to distinguish humans from computers. In *Electronic Commerce and Security (ISECS), 2010 Third International Symposium on*, pages 265–269. IEEE. 14
- [36] Gao, H., Yao, D., Liu, H., Liu, X., and Wang, L. (2010b). A novel image based captcha using jigsaw puzzle. In *Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on*, pages 351–356. IEEE. 15
- [37] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, STOC '09*, pages 169–178, New York, NY, USA. ACM. 6, 13
- [38] Gentry, C., Sahai, A., and Waters, B. (2013). Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology—CRYPTO 2013*, pages 75–92. Springer. 68
- [39] Glaser, B. G. (1965). The constant comparative method of qualitative analysis. *Social problems*, 12(4):436–445. 98, 106
- [40] Golle, P. (2008). Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542. ACM. 15

- [41] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*. [14](#)
- [42] Gossweiler, R., Kamvar, M., and Baluja, S. (2009). What’s up captcha?: a captcha based on image orientation. In *Proceedings of the 18th international conference on World wide web*, pages 841–850. ACM. [15](#)
- [43] Grant-Muller, S. M., Gal-Tzur, A., Minkov, E., Nocera, S., Kuflik, T., and Shoor, I. (2014). Enhancing transport data collection through social media sources: methods, challenges and opportunities for textual data. *IET Intelligent Transport Systems*, 9(4):407–417. [11](#)
- [44] Guarino, B. (2018). *Wildlife fled the sky and bees went silent during the 2017 total solar eclipse*. [42](#)
- [45] Guerar, M., Merlo, A., and Migliardi, M. (2018). Completely automated public physical test to tell computers and humans apart: A usability study on mobile devices. *Future Generation Computer Systems*, 82:617–630. [16](#)
- [46] Hallgren, P., Ochoa, M., and Sabelfeld, A. (2015). Innercircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, pages 1–6. IEEE. [13](#)
- [47] Harlow, S. and Harp, D. (2012). Collective action on the web: A cross-cultural study of social networking sites and online and offline activism in the united states and latin america. *Information, communication & society*, 15(2):196–216. [11](#)
- [48] Hawelka, B., Sitko, I., Beinat, E., Sobolevsky, S., Kazakopoulos, P., and Ratti, C. (2014). Geo-located twitter as proxy for global mobility patterns. *Cartography and Geographic Information Science*, 41(3):260–271. [4](#), [11](#)
- [49] Holman, J., Lazar, J., Feng, J. H., and D’Arcy, J. (2007). Developing usable captchas for blind users. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 245–246. ACM. [14](#)

- [50] Hsieh, C.-C. and Wu, Z.-Y. (2013). Anti-sift images based captcha using versatile characters. In *Information Science and Applications (ICISA), 2013 International Conference on*, pages 1–4. IEEE. [14](#)
- [51] Inc., G. (2018). *Google reCAPTCHA*. [14](#)
- [52] Inc., L. M. T. (2010). Video-based captchas now available for sites and blogs. See www.prnewswire.com/news-releases/video-based-captchas-now-available-for-sites-and-blogs-97471319.html. [15](#)
- [53] Incapsula, I. (2016). *Bot Traffic Report*. [8](#)
- [54] Instagram (2019). *Instagram API Platform*. [4](#)
- [55] Jahanbakhsh, K. and Moon, Y. (2014). The predictive power of social media: On the predictability of us presidential elections using twitter. *arXiv preprint arXiv:1407.0622*. [4](#)
- [56] Jr, R. E. A. (2017). *US Cities in Path of Total Solar Eclipse*. [43](#)
- [57] Jung, C.-T., Tsou, M.-H., and Issa, E. (2015). Developing a real-time situation awareness viewer for monitoring disaster impacts using location-based social media messages in twitter. In *International Conference on Location-Based Social Media Data, Athens, GA, USA March*, pages 12–14. [4](#)
- [58] Jurdak, R., Zhao, K., Liu, J., AbouJaoude, M., Cameron, M., and Newth, D. (2015). Understanding human mobility from twitter. *PloS one*, 10(7). [4](#), [11](#)
- [59] Kaur, R. and Choudhary, P. (2015). A novel captcha design approach using boolean algebra. In *2015 5th International Conference on IT Convergence and Security (ICITCS)*, pages 1–7. Citeseer. [15](#)
- [60] Kerschbaum, F. (2012). Outsourced private set intersection using homomorphic encryption. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 85–86. ACM. [13](#)

- [61] Kluever, K. A. and Zanibbi, R. (2009). Balancing usability and security in a video captcha. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, page 14. ACM. [15](#)
- [62] Kochanski, G., Lopresti, D. P., and Shih, C. (2002). A reverse turing test using speech. In *INTERSPEECH*. [14](#)
- [63] Koestinger, M., Wohlhart, P., Roth, P. M., and Bischof, H. (2011). Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2144–2151. IEEE. [83](#)
- [64] Krutz, R. L. and Vines, R. D. (2010). *Cloud security: A comprehensive guide to secure cloud computing*. Wiley Publishing. [6](#)
- [65] Laine, K. and Player, R. (2016). Simple encrypted arithmetic library-seal (v2. 0). Technical report, Technical report, September. [76](#)
- [66] Lazar, J., Feng, J., Brooks, T., Melamed, G., Wentz, B., Holman, J., Olalere, A., and Ekedebe, N. (2012). The soundsright captcha: an improved approach to audio human interaction proofs for blind users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2267–2276. ACM. [14](#)
- [67] Lewis, J. R. (1991). Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the asq. *ACM Sigchi Bulletin*, 23(1):78–81. [105](#)
- [68] Li, M., Zhu, H., Gao, Z., Chen, S., Yu, L., Hu, S., and Ren, K. (2014a). All your location are belong to us: Breaking mobile social networks for automated user location tracking. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 43–52. ACM. [2](#)
- [69] Li, Y., Gai, K., Qiu, L., Qiu, M., and Zhao, H. (2017). Intelligent cryptography approach for secure distributed big data storage in cloud computing. *Information Sciences*, 387:103–115. [12](#)

- [70] Li, Y.-M., Chou, C.-L., and Lin, L.-F. (2014b). A social recommender mechanism for location-based group commerce. *Information Sciences*, 274:125–142. [18](#)
- [71] Liao, C.-J., Yang, C.-J., Yang, J.-T., Hsu, H.-Y., and Liu, J.-W. (2013). A game and accelerometer-based captcha scheme for mobile learning system. In *EdMedia+ Innovate Learning*, pages 1385–1390. Association for the Advancement of Computing in Education (AACE). [16](#)
- [72] Lichman, M. and Smyth, P. (2014). Modeling human location data with mixtures of kernel densities. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 35–44. ACM. [10](#), [11](#), [22](#), [24](#)
- [73] Lillibridge, M. D., Abadi, M., Bharat, K., and Broder, A. Z. (2001). Method for selectively restricting access to computer systems. US Patent 6,195,698. [8](#), [14](#)
- [74] Lin, R., Huang, S.-Y., Bell, G. B., and Lee, Y.-K. (2011). A new captcha interface design for mobile devices. In *Proceedings of the Twelfth Australasian User Interface Conference-Volume 117*, pages 3–8. Australian Computer Society, Inc. [16](#)
- [75] Liu, D., Bertino, E., and Yi, X. (2014). Privacy of outsourced k-means clustering. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 123–134. ACM. [12](#)
- [76] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110. [91](#), [92](#)
- [77] Lu, Z., Feng, Y., Zhou, W., Li, X., and Cao, Q. (2018). Inferring correlation between user mobility and app usage in massive coarse-grained data traces. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(4):153. [6](#)
- [78] Luis von Ahn, Manuel Blum, N. J. H. and Langford, J. (2000). *The CAPTCHA Web Page*. [8](#)
- [79] Maghrebi, M., Abbasi, A., and Waller, S. T. (2016). Transportation application of social media: Travel mode extraction. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1648–1653. IEEE. [11](#)

- [80] Matthews, P., Mantel, A., and Zou, C. C. (2010). Scene tagging: image-based captcha using image composition and object relationships. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 345–350. ACM. [15](#)
- [81] McKenzie, G. (2019). Spatiotemporal comparative analysis of scooter-share and bike-share usage patterns in washington, dc. *Journal of transport geography*, 78:19–28. [54](#)
- [82] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41. [49](#)
- [83] Mujumdar, D. and Poliseti, S. (2011). A platform to monetize usable & secure captchas for desktop and mobile devices (picatcha). *Retrieved Dec*, 20:2014. [112](#)
- [84] Noguera, J. M., Barranco, M. J., Segura, R. J., and Martínez, L. (2012). A mobile 3d-gis hybrid recommender system for tourism. *Information Sciences*, 215:37–52. [18](#)
- [85] ODOXA (2019). *New modes of travel are disrupting transport in Ile-de-France*. [51](#)
- [86] of City Transportation, N. A. (2020). *84 Million Trips Taken on Shared Bikes and Scooters Across the U.S. in 2018*. [52](#)
- [87] OpenStreetMap (2018). *Nominatim: a search engine for OpenStreetMap data*. [41](#), [60](#)
- [88] Paul, U., Subramanian, A. P., Buddhikot, M. M., and Das, S. R. (2011). Understanding traffic dynamics in cellular data networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 882–890. IEEE. [2](#)
- [89] Phillips, D. (2019). *Securimage*. [103](#)
- [90] Qazi, U., Imran, M., and Offi, F. (2020). Geocov19: a dataset of hundreds of millions of multilingual covid-19 tweets with location information. *SIGSPATIAL Special*, 12(1):6–15. [12](#)
- [91] Rabiner, L. R. and Juang, B.-H. (1993). Fundamentals of speech recognition. [88](#)

- [92] Rashid, M., Gu, X., and Lee, Y. J. (2017). Interspecies knowledge transfer for facial keypoint detection. *arXiv preprint arXiv:1704.04023*. [83](#), [91](#), [92](#), [95](#)
- [93] Review, W. P. (2018). *United States Population 2018*. [44](#), [51](#)
- [94] Reynaga, G., Chiasson, S., and van Oorschot, P. C. (2015). Exploring the usability of captchas on smartphones: Comparisons and recommendations. In *NDSS Workshop on Usable Security USEC*. [112](#)
- [95] Ruoti, S., Roberts, B., and Seamons, K. (2015). Authentication melee: A usability analysis of seven web authentication systems. In *Proceedings of the 24th International Conference on World Wide Web*, pages 916–926. International World Wide Web Conferences Steering Committee. [97](#)
- [96] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49. [88](#)
- [97] Sander, T., Young, A., and Yung, M. (1999). Non-interactive cryptocomputing for nc/sup 1. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 554–566. IEEE. [13](#)
- [98] Sauro, J. and Lewis, J. R. (2016). *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann. [97](#), [102](#)
- [99] Seidl, D. E., Paulus, G., Jankowski, P., and Regenfelder, M. (2015). Spatial obfuscation methods for privacy protection of household-level data. *Applied Geography*, 63:253–263. [12](#)
- [100] Shafiq, M. Z., Ji, L., Liu, A. X., Pang, J., and Wang, J. (2012). Characterizing geospatial dynamics of application usage in a 3g cellular data network. In *INFOCOM, 2012 Proceedings IEEE*, pages 1341–1349. IEEE. [2](#), [10](#)
- [101] Shi, L., Agarwal, N., Agrawal, A., Garg, R., and Spoelstra, J. (2012). Predicting us primary elections with twitter. URL: <http://snap.stanford.edu/social2012/papers/shi.pdf>. [4](#)

- [102] Shirali-Shahreza, M. and Shirali-Shahreza, S. (2006). Drawing captcha. In *28th International Conference on Information Technology Interfaces, 2006.*, pages 475–480. IEEE. 16
- [103] Sivakorn, S., Polakis, I., and Keromytis, A. D. (2016a). I am robot:(deep) learning to break semantic image captchas. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 388–403. IEEE. 14
- [104] Sivakorn, S., Polakis, J., and Keromytis, A. D. (2016b). I’m not a human: Breaking the google recaptcha. *Black Hat*. 14
- [105] Starostenko, O., Cruz-Perez, C., Uceda-Ponga, F., and Alarcon-Aquino, V. (2015). Breaking text-based captchas with variable word and character orientation. *Pattern Recognition*, 48(4):1101–1112. 14
- [106] Statista (2015). Number of apps available in leading app stores as of july 2015. <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>. 20
- [107] Statista (2019). *Most popular social networks in the United States in October 2018, based on monthly active users (in millions)*. 4
- [108] Tayade, P. C. and Bartere, M. M. (2015). Comprehensive study on performance analysis of various captcha systems. *International Journal of Current Engineering and Technology*, 5(1). 15
- [109] Team, E. R. (2016). *Emoji Report*. 48
- [110] Tencent (2015). Tencent announces 2015 third quarter results. <http://www.tencent.com/en-us/content/ir/news/2015/attachments/20151110.pdf>. 28
- [111] Thiagarajan, A., Biagioni, J., Gerlich, T., and Eriksson, J. (2010). Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM. 2

- [112] TimeAndDate (2017). *August 21, 2017 — Great American Eclipse (Total Solar Eclipse)*. [43](#)
- [113] Tongaonkar, A., Dai, S., Nucci, A., and Song, D. (2013). Understanding mobile app usage patterns using in-app advertisements. In *Passive and Active Measurement*, pages 63–72. Springer. [18](#)
- [114] Tran, T. and Lee, K. (2016). Understanding citizen reactions and ebola-related information propagation on social media. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 106–111. IEEE. [12](#)
- [115] Tullis, T. S. and Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. In *Usability professional association conference*, volume 1. Minneapolis, USA. [97](#)
- [116] Twitter (2018). *Introduction to Tweet JSON*. [42](#)
- [117] Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer. [13](#), [76](#)
- [118] Von Ahn, L. (2008). Human computation. In *2008 IEEE 24th international conference on data engineering*, pages 1–2. IEEE. [117](#)
- [119] Von Ahn, L., Blum, M., and Langford, J. (2004). Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):56–60. [15](#)
- [120] Von Ahn, L., Maurer, B., McMillen, C., Abraham, D., and Blum, M. (2008). recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468. [14](#), [117](#)
- [121] Wang, F., Wang, H., Xu, K., Raymond, R., Chon, J., Fuller, S., and Debruyne, A. (2016). Regional level influenza study with geo-tagged twitter data. *Journal of medical systems*, 40(8):189. [4](#)

- [122] Wikipedia (2020). *Facebook–Cambridge Analytica data scandal*. [1](#)
- [123] Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., and Venkataraman, S. (2011). Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 329–344. ACM. [2](#), [10](#)
- [124] Xu, Y., Reynaga, G., Chiasson, S., Frahm, J.-M., Monroe, F., and van Oorschot, P. C. (2012). Security and usability challenges of moving-object captchas: Decoding codewords in motion. In *USENIX security symposium*, pages 49–64. [15](#)
- [125] Yang, H., Zhang, R., and Robinson, P. (2016). Human and sheep facial landmarks localisation by triplet interpolated features. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–8. IEEE. [83](#), [91](#), [92](#)
- [126] Yang, T.-I., Koong, C.-S., and Tseng, C.-C. (2015). Game-based image semantic captcha on handset devices. *Multimedia Tools and Applications*, 74(14):5141–5156. [16](#)
- [127] Zheng, Y., Li, M., Lou, W., and Hou, Y. T. (2012). Sharp: Private proximity test and secure handshake with cheat-proof location tags. In *European Symposium on Research in Computer Security*, pages 361–378. Springer. [13](#)
- [128] Zheng, Y., Yi, X., Li, M., Li, R., Shan, Z., Chang, E., and Li, T. (2015). Forecasting fine-grained air quality based on big data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2267–2276. ACM. [10](#)
- [129] Zhou, Z. and Matteson, D. S. (2015). Predicting ambulance demand: a spatio-temporal kernel approach. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2297–2303. ACM. [10](#)

Vita

Yunhe Feng is a Ph.D. candidate in computer science at the University of Tennessee, Knoxville. He received the B.E. degree in computer science and technology and the M.E. degree in computer technology from the Beijing University of Technology, Beijing, China, in 2011 and 2014, respectively. His research interests include data mining and knowledge discovery, mobile security and privacy, and machine learning.