Augmenting Android Data Collection with Machine Learning

Marvin Newlin
Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH, USA
marvin.newlin@afit.edu

***Abstract*—Android devices compose the majority of mobile devices on the market today. Data mining is an important aspect for researching security of Android devices. However, there is a lack of up to date datasets available for research purposes with Android devices. In this paper, we present a survey of Android logging and data collection tools that can be used to collect data for research with Android devices. Additionally, we explore some of the available mobile datasets and present the idea of augmenting Android data using machine learning and deep learning to synthetically generate data to increase the opportunity for research involving Android data.**

***Index Terms*—Android Logging, Data Mining, Mobile Data Collection, Machine Learning**

I. Introduction

Android devices make up a majority of the mobile device market share [1]. Research involving Android and other mobile device data requires datasets with which to conduct experiments. One issue in this area is that there is a lack of available up to date datasets to conduct experiments on mobile devices [2]. Part of this is due in part to the fact that there are privacy concerns involving the use of mobile device data because of the requirement for humans to use mobile devices in order to collect meaningful data [3]. Additionally, the logistics required for conducting experiments with a large number of participants can be burdensome.

One solution to this problem of dataset availability and privacy concerns is to synthetically generate data that possesses the same characteristics as real data which can then be used to conduct research on mobile devices. This process involves collecting data, generating synthetic data that models the real data and then evaluating the quality of that data for research purposes.

The rest of this paper is organized as follows. In Section II we detail some of the available tools for collecting data from mobile devices. Section III discusses some of the available mobile datasets and their pros and cons. Section IV discusses the idea of synthetically generating data in order to augment the process of conducting research involving mobile device data.

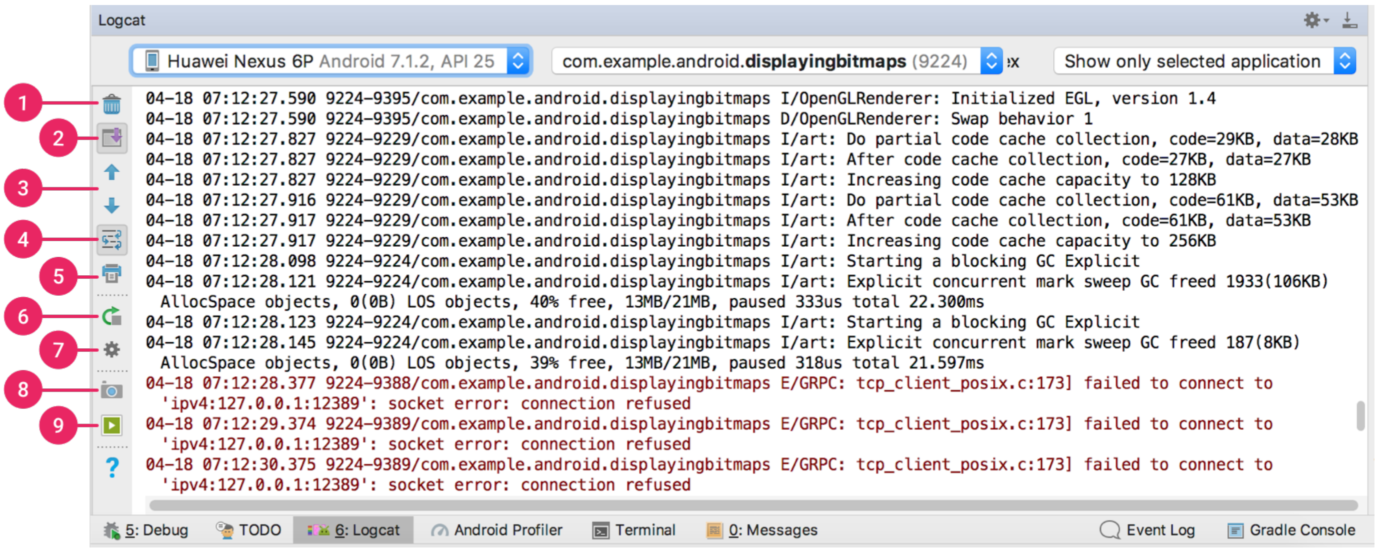
II. Android Logging Tools

In this section, we present a survey of various tools that exist for Android data collection. We have organized the tools into three categories: application logging tools, research data collection tools, and forensics data collection tools. The tools covered in this section were chosen because they are representative of their categories.

A. Application Data Collection

Logcat is the default logging tool for all Android application development [4]. This tool is built into Android Studio and is a very useful tool for debugging Android applications. Logcat works directly on top of the Android Log Application Programming Interface (API) [5]. Five levels of information are defined in the Android Log API: error, warn, info, debug, and verbose. The error level provides information on the cause of the error and the type of exception that occurs in an error. The info level is designed for logging general information in the application. The warn level is for logging information that requires a warning message along with any exceptions necessary. The debug message is designed for the application development phase and usually provides information for the developer. These messages are generally removed from applications in the release build. The verbose type provides all information in the log. This includes all system messages, stacktraces, and any other information that is produced as a result of the log. As a result, log files with verbose log messages tend to be extremely large. All log messages also allow for a tag, a string that is conventionally the method name from which the error is coming. This allows for high granularity in the log messages.

Logcat provides a variety of information in its log messages. Logcat can dump stack traces, print debug information, or print application information [4]. Additionally, Logcat can be utilized via Android studio while developing applications or it can be used to view and filter logs via the Android Debug Bridge (ADB) [6]. One advantage of this feature with Logcat is that it allows for filtering of different log types to display only the type the user wants to view. Logs can be filtered on information level (one of the five mentioned previously), tag, or any regular expression, making it useful for focused data collection in applications [6]. An example of viewing logs in Logcat is displayed in Figure 1. Figure 1 displays examples of a few different types of logs, specifically, info, debug, and error logs. The info and debug log lines only include the message string within the log line and are displayed in black. Error log


Logcat
Huawei Nexus 6P Android 7.1.2, API 25
com.example.android.displayingbitmaps (9224)
Show only selected application
1
2
3
4
5
6
7
8
9
5: Debug
TODO
6: Logcat
Android Profiler
Terminal
0: Messages
Event Log
Gradle Console

```
04-18 07:12:27.590 9224-9395/com.example.android.displayingbitmaps I/OpenGLRenderer: Initialized EGL, version 1.4
04-18 07:12:27.590 9224-9395/com.example.android.displayingbitmaps D/OpenGLRenderer: Swap behavior 1
04-18 07:12:27.827 9224-9229/com.example.android.displayingbitmaps I/art: Do partial code cache collection, code=29KB, data=28KB
04-18 07:12:27.827 9224-9229/com.example.android.displayingbitmaps I/art: After code cache collection, code=27KB, data=27KB
04-18 07:12:27.827 9224-9229/com.example.android.displayingbitmaps I/art: Increasing code cache capacity to 128KB
04-18 07:12:27.916 9224-9229/com.example.android.displayingbitmaps I/art: Do partial code cache collection, code=61KB, data=53KB
04-18 07:12:27.917 9224-9229/com.example.android.displayingbitmaps I/art: After code cache collection, code=61KB, data=53KB
04-18 07:12:27.917 9224-9229/com.example.android.displayingbitmaps I/art: Increasing code cache capacity to 256KB
04-18 07:12:28.098 9224-9224/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-18 07:12:28.121 9224-9224/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 1933(106KB)
 AllocSpace objects, 0(0B) LOS objects, 40% free, 13MB/21MB, paused 333us total 22.300ms
04-18 07:12:28.123 9224-9224/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-18 07:12:28.145 9224-9224/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 187(8KB)
 AllocSpace objects, 0(0B) LOS objects, 39% free, 13MB/21MB, paused 318us total 21.597ms
04-18 07:12:28.377 9224-9388/com.example.android.displayingbitmaps E/GRPC: tcp_client_posix.c:173] failed to connect to
 'ipv4:127.0.0.1:12389': socket error: connection refused
04-18 07:12:29.374 9224-9389/com.example.android.displayingbitmaps E/GRPC: tcp_client_posix.c:173] failed to connect to
 'ipv4:127.0.0.1:12389': socket error: connection refused
04-18 07:12:30.375 9224-9389/com.example.android.displayingbitmaps E/GRPC: tcp_client_posix.c:173] failed to connect to
 'ipv4:127.0.0.1:12389': socket error: connection refused
```

Fig. 1. Example view of Logcat [6]

lines are displayed in red and also contain the corresponding stacktrace with the error message.

Hirabe et al. in [7], utilize the existing Android operating system (OS) logging functions to log all touch operations on an Android phone. They utilized a combination of static analysis of logs and a system dump to analyze and report the logs produced by the touch operations on an Android phone. With this tool, they were able to successfully characterize single touch, multi-touch, single and multiple swipes, pinching in and out, and rotation.

B. *Research Logging Tools*

SystemSens is a system developed in 2011 by Xu et al. that is designed to capture device usage information in a research context [8]. The focus of SystemSens is on usage data such as battery and CPU usage. However, the tool also provides capability for logging calls, network, and cell tower information as well. They conducted experiments with their tool and found that it was successful in capturing the battery and CPU usage data without negatively affecting power usage.

The framework they developed consists of a phone section and a web server section that reports the information collected. Information on the phone is collected and the information is sent via HTTPS in JavaScript Object (JSON) format. Collected information is viewable in the web interface and can be exported into files from there. They also provide the ability for adding data collection sources aside from battery or CPU making it extensible.

Spolaor et al. developed the Data Extraction and Logging Tool for Android (DELTA). Recognizing a lack of tools for collecting Android data for research purposes, they developed this tool to collect data for research purposes. Their tool collects data in three different categories: sensor data, device/OS context data, and user interaction data. The sensor data category involves data collected from sensors like GPS and temperature. Device context category involves OS-level data like log files. User interaction data includes elements like touch operations. Overall, DELTA is able to collect data from 44 different sources, significantly better than its nearest competitor at 17 sources [9].

The novel part about DELTA is the development of the logging framework, the backbone of the tool. This framework allows for easy extension onto the tool and additionally allows for the customization of permission level based on the purpose of experiment, rather than overallocating permissions ahead of time making it more secure than similar tools [9].

DELTA also provides an extensible framework which other researchers and developers can make additions to in case the existing functionality doesn't fit their needs. DELTA's architecture is split into three components, desktop, phone, and web. The desktop components is used to build the experiment as an APK file which is then pushed to the phone and installed like a regular application. During the experiment, the phone sends the collected data over HTTPS in JSON format where it is collected and displayed via the web interface.

C. *Android Forensics Collection*

On the forensics data collection side, automated collection and reporting was introduced in [10] by the MITRE corporation. This tool is focused toward security auditors, forensics investigators, and others in the forensics arena. The tool they developed is called DroidWatch. It monitors and collects data and then stores it locally in a SQLLite database. Data from the application is then periodically sent via HTTPS to a server where the data can be viewed through a web interface [10]. The application is able to collect data from a variety of sources and events like application installation/removal, call logs, GPS location, text message data, and many other sources.

DroidSpotter is another forensics tool developed in 2013 by Kramer in [11]. This tool was designed to collect Android location data from applications and identify where location information is stored on an Android phone. This tool was introduced as a prototype and only has limited capabilities.

III. Available Mobile Datasets

In this section, we detail some of the publicly available datasets for research with Android and mobile devices in general. These datasets were chosen as they are representative of the available datasets and have been frequently referenced in other works.

A. Reality Mining Dataset

The Reality Mining Dataset was first introduced in 2006 in [12]. This dataset was developed by collecting different types of data from 100 different Nokia mobile phones used by participants at Massachusetts Institute of Technology (MIT). The dataset contains information such as call logs, Bluetooth devices near the phone, cell phone tower identification information, application usage, and phone status information. The dataset was anonymized and made publicly available. One of the downsides to this dataset is its age. Since the data was collected in 2004, the majority of the information is outdated and is not particularly useful for today's research contexts. However, the location data is still applicable today and continues to be the most utilized aspect of the dataset.

B. Nokia Mobile Dataset

The Nokia Mobile Dataset was introduced in 2010 in [13]. The dataset consists of data collected from 170 mobile phones from users in Lausanne, Switzerland. The Nokia dataset contains information from four categories: social interaction data, location data, media creation and usage data, and behavioral data. Social interaction data involves call log data, text message data and Bluetooth proximity data like the Reality Mining dataset. The location data includes GPS locations, cellphone network information, and wireless LAN location information. The media creation category includes information such as the location of media storage like images and videos. The behavioral data category contains information on applications used, and inference data on device utilization based on call logs and text messages. All of the data is anonymized to protect privacy information. The most useful aspect of this dataset, like the Reality Mining dataset is the location data.

C. PhoneLab Testbed

In 2013, Nandugudi et al. introduced PhoneLab, a testbed for dataset development for mobile phones [3]. Their motivation is based on a lack of available up-to-date datasets for research uses. The present a system design for open experimentation to collect mobile data for research experiments. The testbed consists of 288 Samsung Galaxy Nexus phones. The dataset they developed contains information such as display information, CPU usage, location data, power consumption, and several other categories.

One aspect to notice about these datasets is that they generally involve mostly device data. There are very few if any datasets available that contain log-based text data. In Section IV we propose a solution to remedy this lack of data with synthetically generated data.

IV. Synthetic Data Generation

As detailed in Section III, there is a lack of datasets that involve text-based log data for research. In this section, we explore a solution that involves synthetically generating data from collected log data.

The end goal for synthetically generating data is to utilize machine learning from big data to help generate a model for realistic data which can then be used to create synthetically generated datasets for research purposes. To do this, a small amount of data could be generated by the collection tools detailed in Section II or other methods. Then, utilizing a variety of machine learning techniques, extract the important features of the data and based on those features, generate more data. From the generated data can be continuously reevaluated to improve the accuracy of the generated data.

Implementing this model, although it sounds straightforward, is not an easy task. One of the main underlying ideas of synthetic data generation is called the Generative Adversarial Network (GAN) introduced in 2014 by Ian Goodfellow [14]. The GAN is composed of a generator and a discriminator. The generator and discriminator are generally made up of a deep learning framework such as a Recurrent Neural Network (RNN). The purpose of the generator is to analyse the data and then generate a synthetic output that mimics the input data. The output from the generator is then sent to the discriminator, whose job it is to perform the ML task of classification by classifying the data as real or synthetic. The response from the discriminator is then fed back into the generator, which improves its generation based on the feedback from the discriminator. This game, depicted in Figure 2, continues until the output from the generator converges and the discriminator cannot distinguish between the generator and the real data [14].

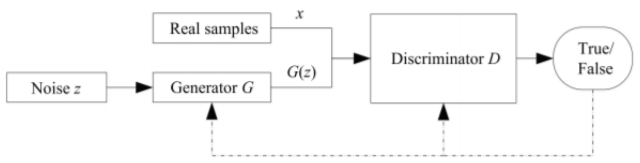


Fig. 2. Generative Adversarial Network architecture [15].

The original idea of GAN as proposed by Goodfellow et al. works well for continuous data (e.g. images or audio). However, when dealing with text or categorical data, it tends to not work as well. Text based mobile log data falls into the discrete category. To this end, some improvements to GAN have been made. The Wasserstein GAN (WGAN) was introduced in 2017 and utilizes a different distance measurement than the original GAN to measure the distance between outputs. This approach solves some of the divergence and training issues that the original GAN framework suffered from [16]. Some additional improvements have been made on the WGAN to improve its functionality on discrete data.

"Improved Training of Wasserstein GANs" [17] discusses the limitations that WGAN produces with weight clipping and demonstrate some of its flaws. The process of weight clipping involves changing the weights associated with the underlying deep learning framework (e.g. a Recurrent Neural Network). One of the flaws with this method is that it hinders the convergence of the algorithm and can contribute to mode collapse, a condition where the generated samples are similar but the mode of the generated samples diverge away from the real sample. To alleviate this problem, they propose a new method of a gradient penalty to maintain the convergence properties of the Wasserstein GAN without the instability produced by weight clipping. The gradient penalty induces a cost on gradients (updates in the search direction) that cause divergence from an optimum value. In their evaluation of this method, they found that this improvement on WGAN helped improve its convergence with discrete data.

Log files are considered semi-structured data because they contain log lines like the ones displayed in Figure 1. These log lines have fields that are certain lengths like a timestamp and also contain variable length type fields like a comment field. This combination of fixed length and variable length fields classify log files as semi-structured data. Additionally, since the log lines in log files are ordered in time, log files are considered sequential semi-structured data. Analogizing this process to synthetically generating images with GANs, we can think of log lines as the individual pixels and the log file as the "image" we want to generate. Generating the individual log lines, as with individual pixels is easy. The hard part is capturing the semantics of the entire file made up of the synthetic log lines. Using the image analogy again, suppose that the real image is an image of a face. Generating the pixels in the image is easy but generating pixels that make up a face is harder. Thus, while each line may be syntactically correct, the relationships between each line are hard to capture from the real input data. Thus, it is difficult to output synthetic log files that possess these overarching relationships. The next step toward developing synthetic data is to capture these relationships with metrics to determine the "realness" of synthetic data. These metrics are key in providing the mathematical properties that enable GANs to work in the way that they do. Once these metrics are developed, then the process of generating synthetic data can be accomplished.

V. Conclusions and Future Work

Android devices are a major focus of research but there is a lack of available up to date datasets for conducting research on mobile devices. In this paper we have explored different types of tools available for collecting data from mobile devices. Some of the available datasets for research have been discussed along with their pros and cons. Additionally, we have discussed a possible solution to the lack of datasets in the form of synthetically generated data and have discussed the idea of utilizing a Generative Adversarial Network to generate this data.

As mentioned in Section IV, synthetic generation of log data will solve the problems that come along with large scale experiments. GANs will make for a good solution to the problem if they can be adapted to the discrete realm which semi-structured data is a part of. Future work in this arena will focus on metrics to determine how "real" synthetically generated data is. Currently, there exists a lack of metrics that can capture the overarching relationships that exist in real log files. Once these metrics are developed, generative solutions such as GAN can be applied to discrete data and the generation of synthetic log data will be possible.

References

[1] X. Xia, C. Qian, and B. Liu, "Android security overview: A systematic survey," *2016 2nd IEEE International Conference on Computer and Communications, ICCC 2016 - Proceedings*, pp. 2805–2809, 2017.

[2] E. Welbourne and E. Tapia, "CrowdSignals: A Call to Crowdfund the Community's Largest Mobile Dataset," in *UbiComp '14*, Seattle, WA, 2014. [Online]. Available: http://dx.doi.org/10.1145/2638728.2641309

[3] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen, "PhoneLab: A Large Programmable Smartphone Testbed," in *SENSEMINE13*, Roma, Italy, 2013. [Online]. Available: www.phone-lab.org

[4] "Logcat command-line tool — Android Developers." [Online]. Available: https://developer.android.com/studio/command-line/logcat

[5] "Log — Android Developers." [Online]. Available: https://developer.android.com/reference/android/util/Log

[6] "Write and View Logs with Logcat — Android Developers." [Online]. Available: https://developer.android.com/studio/debug/am-logcat.html

[7] Y. Hirabe, Y. Arakawa, and K. Yasumoto, "Logging all the touch operations on Android," in *2014 7th International Conference on Mobile Computing and Ubiquitous Networking, ICMU 2014*. IEEE, 1 2014, pp. 93–94. [Online]. Available: http://ieeexplore.ieee.org/document/6799073/

[8] H. Falaki, R. Mahajan, and D. Estrin, "SystemSens: A Tool for Monitoring Usage in Smartphone Research Deployments," in *MobiArch '11 Proceedings of the sixth international workshop on MobiArch*, Bethesda, Maryland, 2011, pp. 25–30. [Online]. Available: https://dl.acm.org/citation.cfm?id=1999923

[9] R. Spolaor, E. D. Santo, and M. Conti, "DELTA: Data Extraction and Logging Tool for Android," *IEEE Transactions on Mobile Computing*, vol. 17, no. 6, pp. 1289–1302, 6 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8067446/

[10] J. Grover, "Android forensics: Automated data collection and reporting from a mobile device," *Digital Investigation*, vol. 10, pp. S12–S20, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2013.06.002

[11] J. Kramer, "DroidSpotter: A Forensic Tool for Android Location Data Collection and Analysis," Ph.D. dissertation, Iowa State University, 2013. [Online]. Available: https://lib.dr.iastate.edu/etd/13407

[12] N. Eagle and A. Pentland, "Reality mining: Sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255–268, 2006.

[13] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila, "Towards rich mobile phone datasets: Lausanne data collection campaign," in *Proc. ICPS*, Berlin, 2010. [Online]. Available: https://pdfs.semanticscholar.org/25d8/9cc6b650ee7ea094f6eb5f58ab8ac0802f65.pdf

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," *ArXiv*, 2014. [Online]. Available: http://www.github.com/goodfeli/adversarial

[15] "Generative Adversarial Network Architecture — Download Scientific Diagram." [Online]. Available: https://www.researchgate.net/figure/Generative-Adversarial-Network-Architecture_fig1_321865166

[16] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv*, 1 2017. [Online]. Available: https://arxiv.org/pdf/1701.07875.pdf

[17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," *ArXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028