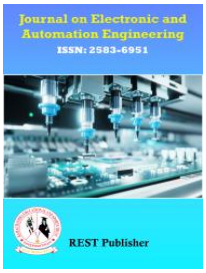
Journal on Electronic and Automation Engineering

Vol: 4(4), December 2025

REST Publisher; ISSN: 2583-6951 (Online)

Website: https://restpublisher.com/journals/jeae/

DOI: https://doi.org/10.46632/jeae/4/4/3



Ranking Fraud Detection in Google Play Store

S. Chandra Sekaran, N. Vijaya Lakshmi, R. Monisha, *R. Sumithira

P. S.V College of Engineering and Technology, Krishnagiri, Tamilnadu, India.

*Corresponding Author Email: sumithira31082003@gmail.com

Abstract: *The Google Play Store, a platform for distributing applications, is overflowing with thousands of new apps every day. Numerous developers, working individually or in teams, strive hard to make these applications successful. Amidst this immense competition from around the world, it becomes crucial for a developer to know if they are on the right track. Unlike the presence of popular actors increasing the chances of a movie's success even before its release, such a situation doesn't exist in app development. Since most Play Store apps are free, the revenue model regarding how in-app purchases, advertisements, and subscriptions contribute to an app's success is quite vague and not easily accessible. Therefore, the success of an app is generally determined not by the revenue it generates, but rather by the number of times it has been installed and the user ratings it has received throughout its lifespan. Hence, in this project, I have attempted to perform performance I conducted analysis and forecasting on the Google Play Store application dataset obtained from kaggle.com. Through the application of big data methods such as Hive, I attempted to discover connections among different variables in my dataset, including app pricing models (free versus paid), user feedback, and application ratings. Furthermore, I have attempted to make predictions regarding which user reviews are positive or negative using deep learning techniques.*

Key Word: *Machine Learning, Malware, Hardware-Based Detection, Hardware-Based Framework.*

1. INTRODUCTION

Big Data refers to information characterized by its massive volume. The term describes datasets that are not only extremely large but also continue to expand rapidly over time. These datasets are so vast and intricate that conventional data management tools cannot effectively store or process them. We can identify whether data qualifies as Big Data by examining certain characteristics.

Volume: This refers to the massive volumes of information generated continuously from various sources including social networking sites, vehicles, banking systems, and aviation.
Velocity: This describes how quickly data is generated, gathered, and processed for analysis.
Variety: This encompasses the diverse formats and types of data being collected and utilized.
Veracity: This addresses the reliability and accuracy of the data, as well as its security.
Value: This pertains to the usefulness and insights that can be derived from the data through analysis.

Big data is used in the following ways:
• Data collection
• Data classification
• Pattern recognition
• Finally, prediction
• Visualization

The Google Play Store, Android's leading app marketplace, faces significant issues with fraudulent activities that promote search ranking manipulation and malware distribution. Prior research has concentrated on examining app executables and permissions to detect malicious software. This paper presents Fair Play, an innovative system that identifies malware and apps involved in ranking fraud by detecting and analyzing digital traces left by fraudsters. Fair Play examines review patterns and combines these findings using language patterns and behavioral characteristics derived from comprehensive Google Play information, encompassing 87,000 apps, 2.9 million user reviews, and 2.4 million individual users.

2. LITERATURE REVIEW

1. Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar, and Thuan Huynh Chau

Deceptive practices within Google Play, the leading Android application marketplace, contribute to manipulated search rankings and the spread of malicious software. Earlier research has concentrated on analysing app code and permissions to detect malware. This paper presents Fair Play, an innovative system that identifies traces left by fraudulent actors and utilizes these traces to detect both malicious software and applications participating in search ranking fraud. Fair Play identifies questionable apps by analysing review patterns and distinctively integrates these review connections with language patterns and behavioural characteristics derived from Google Play data (encompassing 87,000 apps, 2.9 million user reviews, and 2.4 million reviewers gathered over six months). Fair Play demonstrates over 95 percent accuracy when categorizing benchmark datasets containing malware, fraudulent, and genuine applications. The research reveals that 75 percent of detected malware apps participate in search ranking manipulation. Fair Play has identified hundreds of fraudulent applications that bypass Google Bouncer's detection mechanisms. Analysis of over 1,000 reviews across 193 apps has uncovered a novel form of "coercive" review strategy where users face harassment.

2. Malware Detection Inside App Stores Based on Lifespan Measurements CARLOS CILLERUELO , ENRIQUE-LARRIBA, LUIS DE-MARCOS AND JOSE-JAVIER MARTINEZ-HERRÁIZ Computer Science Department, University of Alcalá, 28801 Alcalá de Henares, Spain Corresponding author: Carlos Cilleruelo (carlos.cilleruelo@uah.es)

Potentially harmful applications (PHAs) pose challenges similar to other malware types. Although Google works to keep its app ecosystem secure, the Google Play Store continues to be a major distribution channel for these harmful applications. This research proposes a machine learning-based approach for identifying potentially harmful applications in app marketplaces. Given that app marketplaces serve as primary entry points, there's a need for a system that can detect harmful applications either already present in or being submitted to these platforms. This system can identify harmful applications within marketplaces and function as an automated filtering tool to block the release of new harmful applications. The proposed approach utilizes static application analysis. While numerous static analysis solutions exist, this system's distinctiveness comes from its training methodology and dataset construction. Although it demonstrates lower accuracy than other machine learning models designed for detecting harmful applications, this study's approach can complement other static and dynamic machine learning models due to its unique training method, which relies on application lifecycle measurements.

3. METHODOLOGY

A. Existing System:

Following an extensive examination of research literature on Android applications, we identified substantial limitations in existing research concerning the prediction of Android application success, which inspired us to undertake this study. Current research predominantly emphasizes post-launch data, which, though valuable, fails to offer developers prospective direction. Our study aims to address this deficiency by forecasting critical success indicators, including installation numbers and user ratings, prior to an application's release on the Google Play Store. This forecasting ability holds significant importance as it equips developers with practical insights, enabling them to implement essential adjustments to their applications before market entry. Through enhancing elements such as features, design, and functionality informed by these forecasts, developers can boost user interaction and satisfaction, consequently increasing the application's probability of success in an intensely competitive marketplace. This forward-thinking strategy not only improves the prospects of a successful debut but also enables effective resource distribution and strategic decision-making, ultimately supporting the application's sustained success and financial performance.

Disadvantages:

- No data pre-processing was used.
- Low accuracy (78%-80%).

B. Proposed System:

Following the execution of the project using Hive and Deep Neural Network technologies, I documented the outcomes below. We present Fair Play, a system designed to efficiently identify fraudulent activities and malicious software on Google Play. Our primary achievements include:

- We propose and develop relevant, behavioral, and linguistic features that we use to train supervised learning algorithms to detect fraud and malware.
- We develop the concept of co-review graphs to model review relationships between users.
- We develop PCF, an efficient algorithm to identify temporally constrained, co-review pseudo-groups formed by reviewers with significantly overlapping co-review activities in short time windows.
- We utilize the temporal dimensions of review posting times to identify suspicious review spikes received by applications; we show that to compensate for negative reviews, a fraudster must post at least a certain number of positive reviews for an application with an R rating.

We identify applications with "unbalanced" review, rating, and installation counts, as well as applications with permission request ramps. Using linguistic and behavioral information,

- we identify genuine reviews, from which we
- extract user-identified fraud and malware indicators.

Advantages of Proposed system

This work is founded on the principle that fraudulent and malicious activities create distinctive patterns within app marketplaces.
Fair Play demonstrates exceptional performance, achieving accuracy rates exceeding 97% in identifying fraudulent and malicious applications, and surpassing 95% accuracy in distinguishing between malicious and benign apps.
Fair Play substantially surpasses the malware detection indicators developed by Sharma et al., and our findings reveal that malicious software frequently participates in manipulating search rankings.
Fair Play successfully identifies numerous fraudulent applications, numbering in the hundreds.
Fair Play enabled us to uncover a previously unknown form of coercive review manipulation, wherein application users are pressured into providing favourable reviews and are compelled to download and review additional applications.

C. Design of the system: File design is a crucial aspect that primarily depends on the computer's performance. File design addresses two main components: the size of the files and the unnecessary redundancy of data. At the same time, all files are designed to include all relevant information related to each entity. A single database with information about all entities would complicate the computer system. On the other hand, a relational database includes multiple data tables equipped with systems enabling their interconnection. The connections among table data can be processed, merged, and presented through database interfaces. The majority of relational databases offer capabilities to distribute data:

- Through network systems
- Over internet connections
- To various electronic devices including laptops and handheld devices
- To different software platforms

Thus, the relational database file is implemented in predicting mobile phone addiction using machine learning. The design of the underlying files is critical to the computer system. The performance of the computer depends on how it is designed. Great care has been taken to minimize the size of the files and unnecessary redundancy. At the same time, all files are designed to include all relevant information about each entity. A single database with information about all entities would further complicate the computer system. Each database table will be related to one another, which will be shown in the table structure and normalization process in the upcoming chapters.

4. TESTING

A. Unit testing: Unit testing is a testing technique used to verify the correctness of individual components of a program, typically at the function or method level. In object-oriented systems, unit tests are commonly applied at the class level, with basic tests covering constructors and destructors. These tests are usually created by developers during the coding process using a white-box approach, ensuring that each unit performs as intended. A single function may be subjected to multiple test cases to evaluate different execution paths and boundary conditions. While unit testing cannot independently guarantee the overall correctness of a software system, it plays a crucial role in verifying that individual software components function correctly in isolation. Unit testing serves as a fundamental component of software development, utilizing diverse defect identification and prevention methods to minimize development risks, duration, and

expenses. Unit testing is performed during the implementation stage of the software development life cycle helps in detecting and eliminating bugs early, thereby improving software quality and enhancing the efficiency of subsequent testing phases.

B. Integration Testing: Integration testing is a form of software testing focused on confirming that different components work together properly according to the system's design specifications. Components can be combined either gradually or simultaneously, with the incremental approach typically preferred because it makes identifying and resolving interface problems easier. The purpose of this testing is to uncover bugs in how integrated modules communicate and interact with each other. The process involves progressively combining and testing increasingly large groups of verified components that align with the system architecture, continuing until the entire system functions cohesively.

C. Functional Testing: Functional testing involves verifying that specific code actions or features work correctly. These tests are typically based on code requirements documentation, though some development approaches use use cases or user stories instead. The core question functional testing addresses is whether users can perform certain tasks or whether particular features operate as intended. These tests systematically demonstrate that tested functions work according to business requirements, technical specifications, system documentation, and user guides. Functional testing focuses on:

- Valid Input: Ensuring appropriate input types are accepted
- Invalid Input: Confirming inappropriate input types are rejected
- Functions: Verifying identified functions operate correctly
- Output: Testing that specified application outputs are produced
- Systems/Procedures: Confirming that connected systems or procedures are properly triggered

System Testing: This examines a fully integrated system to confirm it satisfies its requirements. An example might include testing user login, followed by creating and modifying entries, generating or printing outputs, performing summary operations or deleting/archiving entries, and finally logging out.
Performance Testing: This assesses system speed and efficiency to ensure results are generated within specified timeframes according to performance requirements. It's categorized as black box testing.

D. White-Box Testing: Software testing includes both black box and white box approaches. White box testing, also called glass box, structural, clear box, open box, or transparent box testing, examines internal code and infrastructure by checking predetermined inputs against expected outputs.

This method tests the internal workings of an application and requires programming knowledge to create test cases. Its main objective is to examine input and output flow through the software while enhancing security.
The "white box" terminology reflects the internal system perspective—the clear or transparent nature allows viewing inside the software's structure. Test cases originate from the design phase of software development. White box testing employs techniques like data flow testing, control flow testing, path testing, branch testing, and statement and decision coverage to create defect-free software.

White-box testing techniques:

- Data Flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

E. Data Flow Testing: This technique analyzes how data moves through a program by tracking variable data flow and gathering information about specific process points. It examines the control flow to explore variable sequences according to event sequences, focusing on where variables receive values and where those values are used.

F. Control Flow Testing: A white box technique that determines the execution sequence of program statements through control structures. Testers select portions of larger programs to establish testing paths, primarily for unit testing. Test cases are represented through control graphs showing all possible execution paths using nodes, edges, decision nodes, and junction nodes.

G. Branch Coverage Testing: This technique ensures all control flow graph branches are covered by testing all possible outcomes (true and false) of each decision point at least once. While similar to decision coverage, branch coverage specifically ensures every branch of each decision point executes. Statement coverage, a widely-used white box method,

ensures all source code statements execute at least once and calculates the ratio of executed statements to total statements.

H. Decision Coverage Testing: This white box technique provides coverage for Boolean expressions by reporting their true and false outcomes. Decision points occur in control flow statements (like if, do-while, and case statements) that have multiple possible outcomes. Decision coverage tests all possible outcomes of every Boolean condition using control flow graphs or charts.

I. Greybox Testing: Grey box testing examines software applications with partial knowledge of internal structure, combining black box and white box approaches. It accesses internal code for test case design (like white box testing) while testing functionality at the user level (like black box testing). This method particularly identifies context-specific errors in web systems. When testers find defects, they can modify code to fix issues and retest immediately. It examines all layers of complex software systems to expand testing coverage, testing both the presentation layer and internal code. It's primarily used for integration and penetration testing.

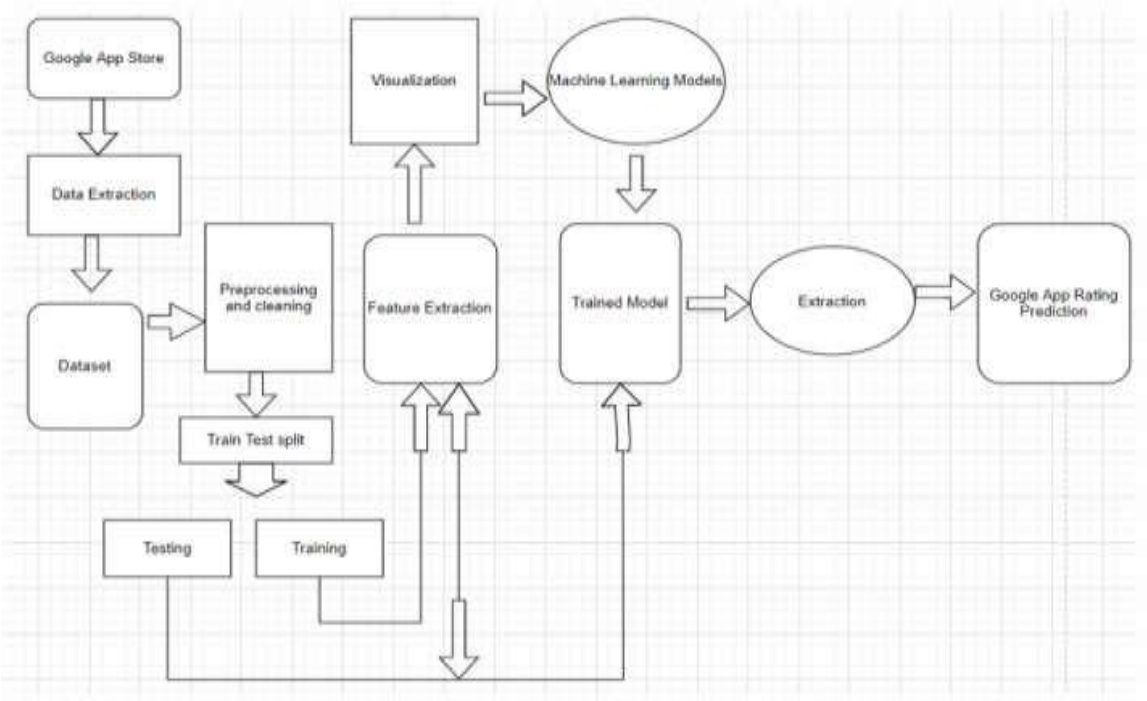


FIGURE 1. Architecture Diagram

5. ARCHITECTURE DIAGRAM

Subjective information can be identified and extracted through techniques like opinion mining, which merges natural language processing, text analysis, and computational linguistics. This text mining specialty develops algorithms to identify and extract written expressions of people's opinions and perspectives. Individual beliefs and values significantly influence everyday decision-making. Mobile devices, particularly smartphones, increasingly affect both our personal and work lives. These devices now offer numerous applications providing services ranging from healthcare and fitness to beauty, monitoring, and sports activities. According to Figure 1A, approximately 2.6 million applications were available on Google Play as of March 2013. Users of these applications can provide feedback through written reviews and star ratings. Figure 1B indicates that annual app downloads currently reach 205.4 billion and are expected to grow to 352.9 billion by 2023. For potential users, reviews and ratings are crucial factors. Research demonstrates that user feedback and ratings significantly influence mobile application adoption rates. Studies show that consumers will pay 20–99 percent more for five-star rated products compared to four-star alternatives. These shared ratings, issue reports, and reviews provide value to both application users and developers.

6. IMPLEMENTATION

Software testing is an investigative process that informs stakeholders about the quality of a software product or service being examined. It offers an objective and unbiased assessment of the software, helping businesses recognize and

comprehend the potential risks associated with software deployment. Testing methods involve running a program or application to identify software defects (bugs or other flaws) and to confirm that the software is suitable for its intended purpose.

The testing process involves running software components or system elements to assess various characteristics of interest.
Generally, these characteristics demonstrate how well the component or system being tested:

- Fulfills the requirements that directed its design and creation
- Handles all types of inputs appropriately
- Completes its functions within a reasonable timeframe
- Provides adequate usability
- Can be successfully installed and operated in its target environments
- Delivers the overall outcomes that stakeholders expect

Experimental Result

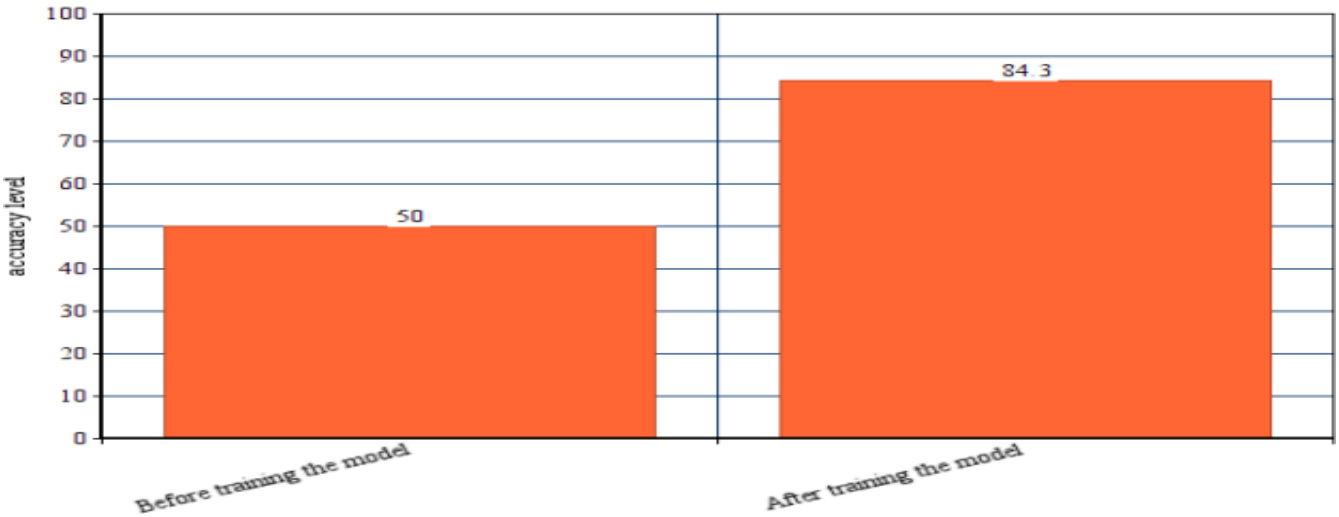


FIGURE 2. Experimental Result

7. CONCLUSION

The Google Play Store represents the largest application marketplace globally. While it surpasses the Apple App Store in download volume, it generates less revenue. We collected data from the Play Store to conduct analytical research. This project employs Big Data technologies, specifically Hive, to examine various characteristics of the Google Play Store application dataset, including leading free applications, top paid applications, most-reviewed applications, and apps featured under editor's choice, utilizing Hive QL to query and present the findings as demonstrated previously. Additionally, the project attempts to classify user reviews as positive or negative based on the provided dataset through the implementation of a deep neural network. The process began by importing the text-based dataset and transforming it into numerical format before feeding it into the deep neural network. The network calculates outputs, compares them with actual values, and then trains the model by adjusting weights to reduce error through backpropagation—a process repeated multiple times. Ultimately, the trained model's outputs are compared against established thresholds to classify reviews as positive or negative. The model's accuracy improved significantly from 50% to 84.3% following the training process, representing a substantial enhancement over the initial model.

Future Enhancement: This project implements Random Forest and Support Vector Machine algorithms to achieve superior accuracy and performance compared to existing app rating methods. Random Forest is among the most widely utilized machine learning methods, especially for categorization and prediction tasks. Likewise, Support Vector Machine represents a supervised machine learning technique applicable to both categorization and prediction challenges.

REFERENCES

[1] Alexandar, “Ranking Fraud Detection in Google Play Store”,2021.
[2] Cristiano Pegoraro Chenet and Alessandro Savino, “A Survey On Hardware-Based Malware Detection Approaches”,2021
[3] Carlos Cilleruelo, Enrique-Larriba, Luis De-Marcos And Jose-Javier,Martinez-Herráiz “Malware Detection Inside App Stores Based On Life Span Measurements”,2021.
[4] Hiroyakato, Takahiro Sasakii wao Sasase and Hiroyakato, “Android Malware Detection Based On Composition Ratio of Permission Pairs”,2021,
[5] Ikram Ul Haq, Tamim Ahmed Khan,Adnan Akhunzada And Ikram Ul Haq,“A Dynamic Robust Dl-Based Model For Android Malware Detection”,2021.
[6] Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar and Duen Horng Chau. “Search Rank Fraud and Malware Detection in Google Play”,2017
[7] Sri Sunngkusuma Wardan, “Hybrid Android Malware Detection: A Review of Heuristic-Based Approach “, 2024.
[8] Sohn, S. Y., Krasnoff, L., Rees, P., Kalk, N. J. & Carter, B. The association between smartphone addiction and sleep: A UK cross-sectional study of young adults. Front. Psych. 12, 629407 (2021).
[9] Wilkerson, G. B. et al. Wellness survey responses and smartphone app response efficiency: Associations with remote history of sport-related concussion. Percept. Mot. Skills 128, 714–730 (2021).

