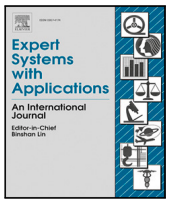
Expert Systems With Applications 288 (2025) 127888

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Comprehensive benchmarking of knowledge graph embeddings methods for Android malware detection

Jan Kincl [a,c,e,*], Tome Eftimov [b], Adam Viktorin [d], Roman Šenkeřík [d], Tanja Pavleska [a]

[a] Laboratory for Open Systems and Networks, Jozef Stefan Institute, Jamova Cesta 39, Ljubljana, 1000, Slovenia
[b] The Computer Systems Department, Jozef Stefan Institute, Jamova Cesta 39, Ljubljana, 1000, Slovenia
[c] Jožef Stefan International Postgraduate School, Jamova Cesta 39, Ljubljana, 1000, Slovenia
[d] Faculty of Applied Informatics, Tomas Bata University in Zlin, nam. T. G. Masaryka 5555, Zlin, 76001, Czech Republic
[e] School of Information and Physical Sciences, University of Newcastle, University Dr., Callaghan, Newcastle, 2308, NSW, Australia

ARTICLE INFO

Keywords:
Mobile android security
Knowledge graphs embeddings
Machine learning
Android malware detection

ABSTRACT

The rising popularity and open-source model of the Android operating system has made it a main target for attackers creating malware applications. With the mobile industry being an expanding device ecosystem, there is a critical need for developing effective methods to protect against mobile malware. Recognizing the latest approaches and their limitations, we have conducted a comprehensive empirical analysis on the applicability of knowledge graphs for malware detection in view of the influence of the scoring functions, the vector dimension, the stability of the obtained results, the performance of the individual classifiers, and other important time dependencies. In addition, we propose a knowledge-graph based method aimed at improving the quality of classification input data, while offering greater interfacing capabilities with external knowledge and lower computational complexity. The proposed method offers a new perspective on working with Android malware, demonstrating a unique data processing pipeline for malware sample identification and encouraging further innovation in the field. Our findings demonstrate that knowledge graph representation is not only feasible, but also provides well-performing results, remaining competitive with state-of-the-art approaches.

1. Introduction

The increasing availability of digital devices has greatly impacted our daily lives, but has also resulted in an increased risk of cyber-threats worldwide. The cybersecurity landscape faces numerous challenges, one of which is the rapid growth of the "malware industry". It is predicted that the financial losses inflicted by cybercriminal activity will reach $10.5 trillion USD in 2025 (Morgan, 2021), a figure that exceeds the financial damages caused by natural disasters in 2021 more than 40 times (OWID, 2024). Mobile devices are one of the largest digital platforms connecting people and businesses. Compared to the desktop industry, the mobile platform holds slightly over 55% of the global market share (StatCounter, 2024a), a trend that is expected to continue (Ceci, 2024).

Android OS is the most widely used mobile operating system, holding over 70% of the mobile devices market (StatCounter, 2024b, 2024c). This popularity has made it a primary target for attacks on the mobile platform. As early as 2012, the FBI reported that over 70% of attacks targeted Android (FBI, 2013). Despite continuous efforts, the security of the Android platform has remained a concern, with numerous instances of users being at risk (EOL, 2024; Statista, 2024; Thomas, Beresford, & Rice, 2015; Toulas, 2023). Additionally, almost 50% of free Android antivirus programs are ineffective in detecting malicious applications (Phillips, 2022). This is a worrying number, considering that malware infestation rate for Android is nearly 50 times higher than for iOS (Nokia, 2021). These problems highlight the escalating need for focused research on Android mobile malware.

Most recent malware detection research has focused on improving the classification performance with increasing model complexity, often relying on deep learning (DL), Graph Neural Networks (GNN) or transformer-based models. However, these approaches require extensive computational resources, are often black-box in nature, with limited interpretability, and demand large, balanced, and preprocessed datasets, which may not always be feasible.

Our approach aims to address the issue of increasingly complex analysis by proposing and evaluating Knowledge Graph (KG)-based malware representation method, supporting significantly less complex

* Corresponding author at: Laboratory for Open Systems and Networks, Jozef Stefan Institute, Jamova Cesta 39, Ljubljana, 1000, Slovenia.
E-mail addresses: jan.kincl@ijs.si (J. Kincl), tome.eftimov@ijs.si (T. Eftimov), aviktorin@utb.cz (A. Viktorin), senkerik@utb.cz (R. Šenkeřík), tanja.pavleska@ijs.si (T. Pavleska).

https://doi.org/10.1016/j.eswa.2025.127888
Received 13 November 2024; Received in revised form 13 April 2025; Accepted 24 April 2025
Available online 21 May 2025

malware detection. Drawing inspiration from ontology creation in domains such as health and nutrition, where KGs are used in disease detection, we introduce a new method that focuses on improving the quality of data input for the detection task, while employing simple and straightforward classification models for evaluation of the benefits. To do that, we develop a methodology for generating embeddings of malware application samples and comparing classification performance across multiple vector dimensions and multiple classification models.

While the idea of using standard machine learning models for malware detection is not new, the key innovation of this study lies in how these models are empowered through a novel data representation pipeline using knowledge graphs-based embeddings rather than in the ML models themselves. Traditional malware detection approaches relying on static analysis technique use fixed feature sets and can hardly keep pace with the rapid evolution of mobile threats and the increasing complexity of malware behaviors. Moreover, they struggle to generalize to new or obfuscated malware samples and perform poorly in malware detection due to the inability to capture semantic relationships between malware application features (Hei et al., 2024; Malhotra, Potika, & Stamp, 2024; Zhao, Sun, Huang, & Zhang, 2025).

Our study proposes that simpler models can achieve competitive performance when powered by richer, semantically aware representations derived from KGs representing the malware applications. The knowledge graph-based embeddings offer compact, dense representations of applications, while enabling inherent encoding of relationships between features and samples, leading to better generalization potential with fewer training parameters. This shifts the innovation from model complexity to data representation.

Additionally, there has been little prior comparative examination on how Knowledge Graph malware representation can affect malware detection performance or computational efficiency when paired with simpler models (e.g., SVM, Random Forest). Therefore, we systematically analyze the influence of different KG scoring functions (TransE, DistMult, ComplEx, HolE) on Android malware detection performance in terms of impact on classification accuracy, computational overhead, and detection stability across embedding dimensions and classifiers.

This comparative evaluation addresses the aforementioned ML limitations and a clear gap in the literature where KG methods were mostly introduced without empirical performance benchmarks, especially in the cybersecurity domain.

Our findings demonstrate that knowledge graph representation is not only feasible but also provides well-performing results. It outperforms simple bag-of-words model and remains competitive with state-of-the-art approaches. The approach aims to achieve performance comparable to the current state-of-the-art, but with significantly reduced computational requirements for feature selection and classification.

The main contributions of our work can be summarized as follows:

- A novel knowledge graph (KG)-based data representation pipeline is introduced for Android malware detection, which organizes static features extracted from APKs into an interpretable graph structure.
- Empirical evaluation of four KG embedding scoring functions (TransE, DistMult, ComplEx, HolE) is conducted across six embedding dimensions (5 to 100), offering a comparative insight into their effectiveness for malware classification.
- Comprehensive benchmarking against standard approaches, including bag-of-words (BOW) and PCA-reduced feature vectors, demonstrates that KG embeddings, particularly HolE, achieve comparable or superior accuracy with significantly reduced dimensionality.
- Demonstration of high detection accuracy using simple machine learning models, validating that complex classifiers are not a prerequisite for effective malware classification when semantically rich embeddings are used.
- Evaluation of computational efficiency and scalability, showing that the KG-based approach dramatically reduces storage and runtime requirements compared to traditional high-dimensional models, making it suitable for resource-constrained environments.
- Provision of a reusable and scalable KG-based feature representation, which offers potential for future integration with external threat intelligence sources and supports continued evolution of the malware detection pipeline.

The rest of the paper is organized as follows: Section 2 introduces the main concepts necessary to understand the presented work. It is followed by an overview of the relevant state-of-the-art approaches in Section 3, fitting the contributions of our work within the current malware detection landscape. In Section 4, the methodology of using a KG representation for mobile Android malware detection is outlined, introducing a new data processing pipeline. Section 5 presents the experimental design for the empirical analysis on KG applicability, followed by rich discussion on the results and their implication for malware detection in Sections 6 and 7. Finally, in Section 8 we conclude and point to future directions.

2. Theoretical background

2.1. Android malware detection

The mobile Android malware landscape is characterized by a diverse range of malicious software variants (Trojans, ransomware, spyware, adware, banking malware, etc.) designed to compromise the security and privacy of Android devices and users (Li et al., 2022). These malware variants exhibit various behaviors, propagation techniques, and attack vectors, posing threats to individuals, organizations, and the mobile ecosystem as a whole.

Malware detection is just one aspect of the overall malware analysis lifecycle, and is typically referred to as binary classification (Pandey & Mehtre, 2014). There are three main approaches to Android malware analysis: static, dynamic, and hybrid (Jusoh et al., 2021). Static analysis involves examining the application's resources, such as configuration files, source code, and other accessible assets. Dynamic analysis, on the other hand, is performed on an application executed within a controlled test environment to monitor the application behavior, network traffic, and system changes. Hybrid analysis combines elements of both static and dynamic analysis, aiming to leverage the strengths of both. Among these, static analysis is often the most preferred approach due to its significantly lower resource consumption, as it does not require the installation and execution of applications (Enck, 2011; Jusoh et al., 2021). It can also detect unknown malware based solely on the evaluated features (Yerima, Sezer, & McWilliams, 2014). Extracted and selected features from static analysis then serve as input for classification tasks, which nowadays increasingly rely on Machine learning (ML) and artificial intelligence (AI) methodologies.

2.2. Knowledge graphs for malware detection

Knowledge graphs organize and connect information in a graph structure, allowing entities (nodes) to be linked by relationships (edges) to form a semantic network (Hei et al., 2024; Ma, Bai, Xing, Sun, & Li, 2020). In that context, entities represent various components and attributes related to malware, including malware samples, behaviors, indicators of compromise (IOCs), attack techniques, malware families, vulnerabilities, and mitigation strategies. Relationships, in addition, capture the semantic connections between entities, such as "belongs to", "causes", "targets", "exploits", "mitigates", and "is similar to". Attributes associated with entities provide additional descriptive information about malware characteristics, behaviors, and properties, such as: file hashes, file types, API calls, system calls, network traffic patterns, permissions, signatures, and metadata (Demirkıran, Çayır, Ünal,

& Dağ, 2022). The interconnected nature of knowledge graphs facilitates cross-referencing, contextualization, and the discovery of relevant information across various aspects of malware analysis. This approach allows malware samples to be represented as high-dimensional vector embeddings derived from the knowledge graph, where each dimension corresponds to a feature or characteristic of the sample, including semantic similarities. Consequently, entities sharing common attributes, behaviors, or relationships will have similar vector representations, enabling effective similarity-based comparisons and clustering.

Although the current body of research employs knowledge graphs to address different aspects of malware detection, there has been no comprehensive investigation into the specific contexts and conditions under which knowledge graphs are effective in detecting malware. Existing studies often focus on feasibility or ontology creation, but not on rigorous performance evaluation or practical benefits.

3. Related work

This section aims to introduce some of the approaches present in the domain of Android malware analysis and detection, placing our work among the state-of-the-art approaches while supporting the understanding on the malware analysis pipeline explained in this work.

3.1. ML and AI methods for malware detection

With recent advantages, both machine learning (ML) and artificial intelligence (AI) methodologies have been extensively employed as part of the android malware detection (or classification) pipeline (Arif et al., 2021). For instance, static approaches for malware detection typically include features like API calls (Gao, Cheng, & Zhang, 2021; Imtiaz et al., 2021; Karbab, Debbabi, Derhab, & Mouheb, 2018; Li et al., 2018; Shiqi, Shengwei, Long, Jiong, & Hua, 2018; Taheri et al., 2020), permissions (Adebayo & Aziz, 2019; Arslan, Doğru, & Bariş¸ci, 2019; Firdaus, Anuar, Karim, & Razak, 2018; Imtiaz et al., 2021; Li et al., 2018; Mathur, Podila, Kulkarni, Niyaz, & Javaid, 2021; Razak et al., 2018; Taheri et al., 2020), and intents (Feizollah, Anuar, Salleh, Suarez-Tangil, & Furnell, 2017; Imtiaz et al., 2021; Taheri et al., 2020). They directly employ machine learning models such as Random Forest, Naive Bayes, Logistic Regression, Support Vector Machine (Arslan et al., 2019; Feizollah et al., 2017; Mathur et al., 2021; Taheri et al., 2020), evolutionary algorithms in combination with ML methods (Adebayo & Aziz, 2019; Firdaus et al., 2018; Razak et al., 2018) (e.g. Genetic Search, Particle Swarm Optimization), as well as deep learning approaches utilizing more complex approaches like Graph Convolutional Networks and Bidirectional long short-term memory (Amin et al., 2020; Gao et al., 2021; Hasegawa & Iyatomi, 2018; Imtiaz et al., 2021; Karbab et al., 2018; Li et al., 2018; Liu, Lin, Li, & Zhang, 2020; Shiqi et al., 2018; Wang et al., 2020). Some studies have additionally employed byte-code (Amin et al., 2020), binary data converted to images (Liu et al., 2020), and combinations of code features with permissions and system features (Arslan et al., 2019; Firdaus et al., 2018; Hasegawa & Iyatomi, 2018). However, static analysis based approaches inherently assume that all behaviors reflected by features should be involved within the model training, thus lacking the capability of tackling unknown (out-of-sample) cases, resulting in high false positives (Li et al., 2018). Although obfuscation techniques have been used to make these models adaptable to new changes, they face the problem of not being able to retain compatibility with the datasets used for training (Tian, Yao, Ryder, Tan, & Peng, 2020). In addition, dynamic feature analysis detects mal-behavior in real time, which limits their scalability and increases the computational complexity (Dimjašević, Atzeni, Ugrina, & Rakamaric, 2016).

Recent advances in Android malware detection have shown a growing interest in graph-based machine learning, especially the use of Graph Neural Networks (GNNs), which model structural dependencies among entities like API calls, system events, permissions, and behaviors. These methods demonstrate strong performance by learning rich, task-specific representations from graph-structured data (Malhotra et al., 2024). Table 1 shows a comparison of our KG embedding-based approach with several state-of-the-art GNN-based techniques: Graph attention network (GAN) (Hei et al., 2024), Graph Convolutional Network GCN (Gao et al., 2021) and Federated GNN (Wu, Qiu, Wu, Jiang, & Jin, 2024). Although the comparison relies on reported results from various literature sources, rather than a direct benchmarking on identical datasets, it still offers useful insights into the relative effectiveness of these methods.

While GNN-based models demonstrate excellent detection capabilities, they often require careful graph schema engineering, graph construction per sample and specialized hardware for training and inference (e.g., GPUs). In contrast, our approach demonstrates comparable or superior accuracy, especially with HolE scoring. It is simple yet efficient, using only KG embeddings and standard ML classifiers. It possesses graph-agnostic classifier compatibility, with reusable and compact embeddings. Hence, the KG-embedding method represents a middle ground between expressiveness and deployability.

3.2. Federated learning approaches for malware detection

While traditional centralized machine learning methods have significantly advanced malware detection, they often face serious limitations in terms of data privacy, security, and scalability, especially in sensitive environments such as mobile networks, IoT systems, and satellite–terrestrial integrated networks. Federated Learning (FL) has emerged as a promising paradigm in cybersecurity for enabling collaborative model training across distributed devices or data silos without exposing raw data (Alazab et al., 2022; Ghimire & Rawat, 2022). This decentralized approach addresses the growing concern over data privacy, which is critical in sectors dealing with personal, governmental, or enterprise-level information. In malware detection, where access to diverse malware samples is crucial, FL can enable distributed participants (e.g., mobile devices, ISPs, security vendors) to collaboratively build models that generalize better to unseen threats, including zero-day malware and polymorphic attacks (Venkatasubramanian, Habibi Lashkari, & Hakak, 2023).

Several recent studies have highlighted the potential of FL in enhancing cybersecurity capabilities. In Wu et al. (2024), authors introduce a novel FL framework that incorporates attention-based graph neural networks (GNNs) to detect complex network attacks across decentralized nodes. Their approach achieves state-of-the-art detection accuracy while preserving data privacy, demonstrating the viability of combining graph-based models with federated architectures for intrusion detection. In addition, Jiang, Han, Zhang, Mu, and Shankar (2024) proposes an intrusion detection system leveraging federated learning combined with conditional GANs to simulate attack patterns across heterogeneous nodes. Their system shows resilience against data imbalance and adversarial attacks, offering a practical defense mechanism in space–ground integrated communication systems.

Despite this potential, the deployment of FL in malware detection comes with challenges (Alazab et al., 2022): the heterogeneous data distributions (non-IID) across nodes can affect model convergence and performance; noticeable communication overhead during training rounds may impact resource-constrained environments; and security risks in FL itself, such as poisoning or inference attacks, remain open problems even with perfectly optimized models.

By developing lightweight yet expressive representations for Android malware classification using knowledge graph embeddings, our work offers an alternative path to improving detection in privacy-aware and computationally constrained settings. Thus, while we do not directly implement federated learning, our focus on computational efficiency, reduced data requirements, and structural data representation aligns well with future integration into federated systems. For instance, KG embeddings could be generated locally at edge nodes and aggregated through federated learning to form privacy-preserving, collaboratively trained detection models.

Table 1
Comparison with SOTA GNN methods.

Aspect	Our KG-embedding approach	GAN	GCN	FL-GNN
Graph type	Simple undirected, single-edge KG	Heterogeneous, multi-relational	Feature-call graph	Multi-relational network graph
Model complexity	Low (embeddings + classic ML)	High (attention GNN)	Medium–High (GCN)	High (FL + attention GNN)
Training time	Fast (preprocessing + ML)	Slow (GNN + attention layers)	Moderate	Very slow (federated + GNN)
Computational requirements	CPU-friendly	Requires GPU	Requires GPU	Requires distributed setup
Accuracy	97%+ (HolE, D = 80–100)	98%+ (on curated sets)	96–97%	95–96%
Embedding reuse	Yes, fixed vectors reusable	No (model-specific)	No	No

3.3. Natural language processing for malware detection

In the efforts to address the above challenges, the integration of natural language processing (NLP) in the malware detection pipeline appears as an emerging trend, utilizing transformer-based language models like BERT (Rahali & Akhloufi, 2023) or custom LLMs for feature extraction and code analysis (Bitdefender, 2024). For instance, Fang et al. (2024) investigates 5 LLMs (GPT-3.5-turbo, GPT-4, LLaMA-2-13B, Code-LLaMA-2-13B-Instruct, StarChat-Beta) in terms of their performance on reading and understanding Android malware application code. The models demonstrate potential in analyzing the decompiled source code of the malware sample, and even overcoming the code obfuscation. Additionally, transformer-based language models have been used to directly process the extracted malware application features, creating embeddings that serve as input data for the classification process. Thus, Rahali and Akhloufi (2021) proposed a transformer-based architecture called MalBERT, designed for detecting malicious Android applications categories by utilizing pre-processed application features, which were further improved in MalBERTv2 (Rahali & Akhloufi, 2023). The potential of NLP-based classification was also confirmed in Demirkıran et al. (2022), by comparing different transformer models using API calls for multi-class malware classification. Some body of works joining NLP with image recognition models have also been adapted for use in the case of malware detection, through the use of deep CNNs (McLaughlin et al., 2017), nonlinear activation function for classification (Vinayakumar, Soman, & Poornachandran, 2017), and other semantic models (Xiao, Zhang, Mercaldo, Hu, & Sangaiah, 2019). However, as applications are constantly updated, explicit feature extraction from a constrained set of applications is ineffective in detecting unknown applications. Moreover, the time inefficiency imposed by the convolution operations becomes a potential bottleneck.

3.4. Knowledge graphs for malware detection

An approach that makes the effort to address the efficiency of malware detection while accounting for out-of-sample applications was presented in Hei et al. (2024). It uses knowledge graphs to model the Android entities and their relationships, exploiting the semantic structure to infer higher order relationships. This results in a learning model capable of dynamically handling the applications, without the need for reconstructing the whole graph and the subsequent embedding model. Multiple studies have recently proved that knowledge graphs and ontologies represent a promising approach for Android malware analysis, bringing important improvements in the domain. Thus, the AndroVault framework introduces a system leveraging knowledge graphs for correlating application samples based on their attribute values (Meng et al., 2017). In addition to providing knowledge representation and extraction capabilities, the work also focuses on providing data suitable for further processing, including malware classification. Furthermore, an open-source malware ontology named MALOnt provided a structured framework for extracting information from malware threat reports and generating knowledge graphs for threat intelligence processing (Rastogi, Dutta, Zaki, Gittens, & Aggarwal, 2020), which updated in MalONT2.0 to include a broader range of classes and relation types (Christian, Dutta, Park, & Rastogi, 2021). A recent study extended the existing ontology framework AndMalOnt with additional classes and properties added into the process of creating a knowledge graph ontology of a malware dataset (Sabbah, Kharma, & Jarrar, 2023). All of the above approaches underscore the utility, high potential and feasibility of utilizing knowledge-graphs in the process of malware analysis for representing the complex malware data.

While the presented studies have shown the feasibility of supporting threat intelligence and classification of the samples into malware families, there is no comprehensive investigation of the applicability of knowledge graphs that would pinpoint concrete aspects of practical benefits for the problem of Android malware detection. Our paper represents a contribution in that direction, carrying out analysis of KG applicability in terms of the scoring functions' impact, the embedding vector dimension, the stability of the obtained results, the classifiers' performance, etc. In addition, we introduce a new knowledge graph-based method for representing the malware applications that aims at improving the quality of classification input data in a way that competes with the current state-of-the-art approaches while providing greater ability for interfacing with external knowledge and lower computational complexity. The following section defines the methodology followed to obtain the model.

4. Methodology

In order to understand the methodology for malware detection, it would be beneficial to first understand what it is in the Android applications that enables the malware analysis in the first place.

Android applications are compiled into an Android Application package (*.apk (APK)*) (Android Developers, 2023), which is the format used to install applications on Android devices. The APK file format is similar to a *.zip* or *.exe* file, containing all the elements of the application — typically, Java or Kotlin classes, resource files (such as images or texts used in the application), the manifest file, and compiled resources. This structure of the APK file allows for further processing and engineering, and by dissecting it, security analysts can better understand the components and behavior of an application, which aids in identifying and mitigating malware threats. Understanding the APK features is crucial for assessing the security and functionality of Android applications, as each feature can be exploited by malware.

Our approach relies on static feature analysis, more specifically on the: Permissions, Opcodes, API calls and packages, System commands, Activities, Services, and Receivers from the APK file. Permissions specify what sensitive data the app can access, for e.g., when reading contacts or accessing location data. Inappropriate permissions can be indicative of potential malware. Opcodes (operation codes) determine the specific operations the app performs, such as arithmetic calculations, data manipulation, and control flow. Analyzing them can help identify unusual behavior patterns in the app's execution. Furthermore, frequent or unusual API calls to sensitive functions can signal malicious behavior, whereas execution of system commands can be a red flag for unauthorized data access or system modification. Activities represent the user interface of the app, handling the user interactions. Malicious activities can, for e.g., mimic legitimate screens to phish for user information or mislead users. Services handle long-running operations or work that needs to continue even when the user is not interacting with the app, like playing music in the background, fetching data from the internet, or updating a database. It is highly relevant for the security analysis as malicious services can perform harmful actions in the

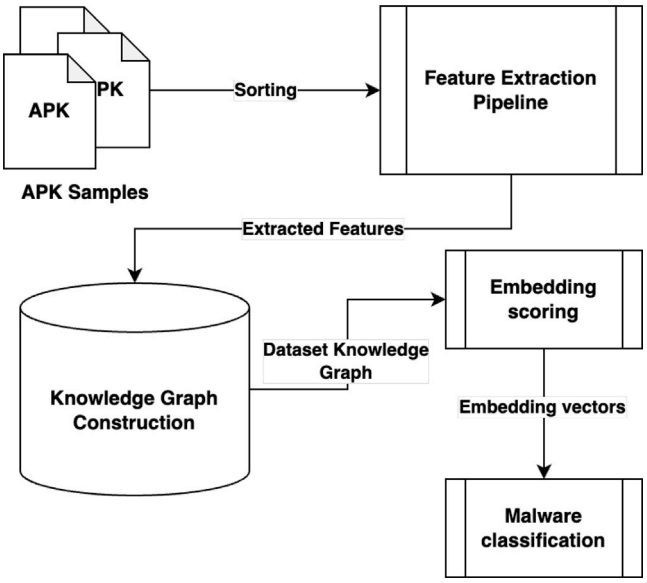


Fig. 1. Knowledge graph embedding generation pipeline.

background, such as spying on user activities or sending sensitive data to remote servers. Finally, receivers enable apps to respond to events, such as system boot completion or network connectivity changes and malware in that context can intercept broadcasts to trigger harmful actions or to gather sensitive information without user consent.

While the above is not the exhaustive set of features that can be analyzed for malware, it is clear that they all bear high relevance for from a security aspect and provide valuable data for the malware detection process. Our method aims to offer a new perspective on working with the different types of Android malware that can exploit the features. It relies on a unique data processing pipeline, which effectively goes through the following four stages:

1. Construction of the knowledge graph of an application sample directly from its extracted features;
2. Creation of an interconnected KG-representation for the entire dataset being processed;
3. Utilization of the embeddings generated from the KG of the malware dataset in creating the representation of individual malware samples; and
4. Performing malware detection using the generated embeddings and validating its accuracy through simple and straightforward ML models.

The main methodological steps of these stages are depicted in Fig. 1.

The following subsections offer a more detailed view on the processes and design choices involved in each of the stages.

4.1. Pre-processing and extraction of application features

The first step of the detection pipeline is the pre-processing of the application dataset and the extraction of application features. This encompasses sorting and decompiling of the APK packages[1] to extract individual source files and, subsequently, the application features. Feature extraction is followed by the process of feature selection, which is usually domain expert driven and consists of selecting a subset of the extracted features. This subset is then used to represent the analyzed applications, forming the base for the classification tasks.

[1] The decompiling can be done by using one of the many available tools (Jusoh et al., 2021).

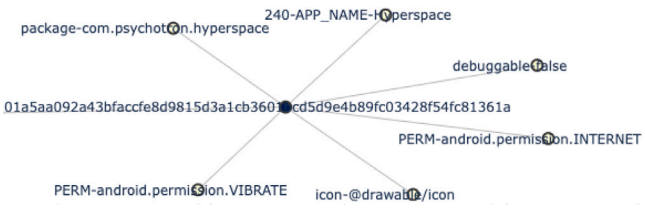


Fig. 2. Exemplary knowledge graph created from a single application sample, showcasing the relations between "root" and "leaf" nodes.

The features of the application can be represented in a tree structure, where the "root" node is the APK package HASH signature, and all other features (such as, *android.permission.SEND_SMS, android.intent.action.INSERT, etc.*) from the individual feature categories (*Strings, API calls, Permissions, etc.*) are connected as "leaf" nodes.

4.2. Creating a knowledge graph of the application dataset

Following the extraction of application features, the next step is to organize the features into a knowledge graph, consisting of one "root" node and N "leaf" nodes. The knowledge graph serves as a depiction of interconnected real-world entities, encompassing objects, events, situations, or concepts, showcasing their relational links (Chen, Jia, & Xiang, 2020). The KG representation employed in our methodology is based on an undirected graph, implying that the relation between two nodes has no direction and simply represents the presence of a given attribute in the application sample.

It is important to note here that the choice of edge-type can significantly impact both the effectiveness of malware detection and the nature of the analysis. For instance, directed edges can introduce direction relationships and add a behavioral component to the analysis. Furthermore, weighted edges can have associated numerical values, indicating relationship strength to quantify the relationship importance. Temporal edges can include timestamps and enable time-series analysis. As we wanted to limit the scope of our analysis to sole embeddings usability, the more complex edge types were not utilized in favor of undirected, single edge type graph structure. This eliminates potential increases in computational complexity at the expense of richer representation and analysis.

In parallel to the organization of features into KGs, a KG representing the whole application dataset is also created by adding both the "root" and the "leaf" nodes of individual applications into a single feature space.

The values of the individual "leaf" nodes then represent the values of the extracted feature itself. Fig. 2 shows the graph structure for a single, simplified application. The blue nodes are the application "root" nodes, while the remaining nodes correspond to the unique features that have been extracted for that application. In a similar manner, a knowledge graph for the application sample is also obtained.

Using a knowledge graph to process the application dataset allows for a flexible and dynamic expansion of the feature space without the need for cross-comparison of individual applications to establish feature vectors. The proposed approach ensures that feature nodes uniquely exist within the graph, with each application from the dataset being dynamically linked to them. This connection not only expands the feature space, but also enhances the understanding of the relations between the analyzed applications and the importance of specific features. These insights are then reflected in the subsequent knowledge graph embedding process.

4.3. Application embeddings generation

Once the KG is created, it serves as an input for the next phase — generating the application embeddings (vector representations), which in turn feed the malware classification models. The embedding generation step involves transforming the graph data into lower-dimensional vector representations that capture the semantic relationships and structure of the graph. To obtain the embeddings, we use KG embedding learning methods (scoring functions) that utilize our representation of the application dataset in the form of a knowledge graph (Wang, Mao, Wang, & Guo, 2017). These embeddings encode semantic information, allowing the ML methods to encompass the entities (nodes) and their connections. Greater details regarding the representation learning are presented in Section 5 - Experimental design.

During the embedding generation stage, a choice should be made for a suitable scoring function. This function is a critical component in the embedding generation process, as it determines how the relationships and interactions within the knowledge graph will be quantified and represented in the embedding space. Moreover, it ensures that the embeddings accurately capture the semantic relationships between different features and guides this reduction process, making the embeddings more manageable and computationally efficient.

There are different types of scoring functions and their choice depends on the goal of the analysis and the choice of other elements in the malware detection pipeline. Different scoring functions lead to different embedding quality, affecting the overall performance of the malware detection. It is important to note, however, that the complexity of the scoring function affects the computational resources required for embedding generation, which may impact the feasibility of the approach for large datasets, as well as for resource-constrained environments.

4.4. Binary classification

The final step of the process is to use the obtained embeddings in the task of binary malware classification (to denote whether the sample is malware or not) and evaluate the accuracy and performance of the detection based on the KG embedding. The goal is to improve the quality of data representation and evaluate how well does the subsequent classification perform using straightforward algorithms. The output of this stage are the classification labels for each application (malware or benign).

The choice of classifier has impact on the detection accuracy, the efficiency of the pipeline, the scalability of the overall system, its robustness and complexity. For our instance (balanced testing dataset), high accuracy is important for minimizing false positives (benign applications wrongly classified as malware) and false negatives (malware not detected). Furthermore, efficiency is essential for handling large datasets and real-time detection scenarios, particularly on resource-constrained mobile devices. At this point of our research, we are not concerned with the real-time performance, as we focus on the case of static analysis. However, we are still aiming for scalable classifiers to ensure that the detection can handle growing amounts of data without degradation in performance. Therefore, we aim for methods that are less computationally heavy and less complex compared to some of the current state-of-the-art approaches, such as deep learning or transformer-based language models. Such simpler machine learning methods are able to process the numerical embeddings vectors, facilitating direct performance comparison between them. Therefore, during model selection, the priority was given to selecting simple machine learning models available in the scikit-learn toolkit (Pedregosa et al., 2011).

One important remark to be made here is that the performance of the classifier is also influenced by the choice of the scoring function, and should thus be done in parallel.

5. Experimental design

In this section, we provide detailed insights into the experimental study designed to demonstrate the practical viability of the proposed methodology. Each step of the experimental design follows the proposed malware detection pipeline. Thus, we start with the pre-processing and explaining the dataset used; we then proceed to the knowledge graph creation, which enables the embedding generation and the final (ML) classification of the analyzed dataset.

5.1. Pre-processing and extraction of application features

Dataset used for the purpose of the study is derived from the CICMalDroid 2020 dataset (Mahdavifar, Abdul kadir, Fatemi, Alhadidi, & Ghorbani, 2020; Mahdavifar, Alhadidi, & Ghorbani, 2022), a widely recognized and publicly available benchmark dataset for Android malware research. The original dataset contains more than 17,000 Android application samples, categorized into benign and various malware families. Samples selected for the experiment were selected based on following criteria: **(i) Successful feature extraction:** Only APK files for which the feature extraction process completed successfully (via AndroPyTool (Martín García, Lara-Cabrera, & Camacho, 2018; Martin Garcia, Lara-Cabrera, & Camacho, 2018) Docker image) were retained and considered for further analysis. This reduced the dataset size to 13,077 samples. **(ii) Balanced classes distribution:** All APK samples from the dataset were categorized into two subsets: *"Benign" (CLN)* and *"Malware" (MLW)*, with malware subtypes (e.g., ransomware, adware, spyware) not considered in this binary classification setting. Afterwards, 850 malware and 850 benign samples were randomly selected from the previously derived dataset. This ensured class balance for the experiment, which was crucial for evaluating the influence on classification without bias toward the majority class. **(iii) Random samples strategy:** Random sampling was applied 30 times to generate different train-test splits during the experiment (train: 1500, test: 200). This minimizes overfitting to a specific sample split and enables statistical stability and reproducibility in accuracy evaluations.

5.2. Creating a knowledge graph of the application dataset

The extracted features from the previous step are stored in JSON files, following the application attributes hierarchy. This facilitates the construction of the knowledge graph of the application sample, which follows the hierarchy of "root" and "leaf" nodes. First, a SHA-256 fingerprint of application package is calculated and its value is used as a value for the application "root" node. Next, all the extracted features are loaded from the features JSON files and one by one added as a "leaf" node into the application knowledge graph. Thus, a knowledge graph for the whole application dataset is created. The knowledge graphs of the individual application samples are then unified into one single feature space. By rule, every unique feature value can appear only once in the unified knowledge graph. Therefore, two identical "leaf" nodes cannot exist in the knowledge graph. In that manner, the unique "root" nodes of applications are added into the knowledge graph and, similar as before, the edges to the "leaf" nodes are created. If the same "leaf" node is shared among multiple applications, it will be connected to each of the applications "root" nodes. This way, a connection between the applications in the dataset is created and a relation between different applications is created. The above steps result in a heterogeneous network comprised of different types of entities that represent the nodes. All relations are of the same semantic type, which means that an edge can exist only between a pair of application and feature node and each feature "leaf" node can be connected to N application nodes.

Fig. 3 presents a simplified KG of four applications. Purple nodes represent malware applications, while light blue nodes denote benign applications.

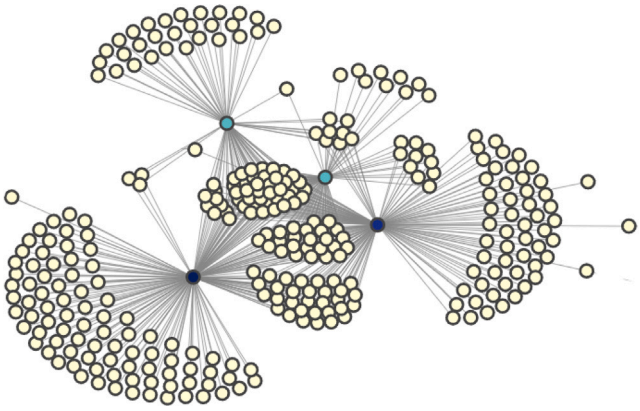


Fig. 3. Interconnected knowledge graph of an example dataset, showcasing both malware and benign applications interconnections through shared feature "leaf" nodes.

5.3. Application embeddings generation

As explained before, during the process of embedding generation, it is important to make a suitable choice for the scoring functions. In the particular context of mobile Android malware detection, it is important to understand how the particular detection model represents the relationships within the data and how effectively it can be used to detect anomalies or malicious patterns. Different scoring functions offer different advantages and impose particular limitations. For mobile Android malware detection, the choice of scoring function should balance expressiveness, computational efficiency, and the ability to model complex relationships.

To obtain the KG embeddings of all applications within the analyzed dataset, we use the AmpliGraph library (Costabello et al., 2019). At the time of the study, the following scoring functions were considered to learn the embeddings from the KG representation: *TransE, DistMult, ComplEx, and HolE*. The TransE (Translation Embedding) and DistMult (Distinguishing Multiple) functions are computationally efficient, and easy to understand and implement. They are suitable for simpler representations of relationships, but might miss complex interaction patterns in malware detection. DistMult, on the other hand, handles symmetric relationships well, which can be useful for identifying patterns where malware components have similar roles. However, it cannot effectively model asymmetric relationships, which are common in malware detection. Finally, the ComplEx (Complex Embeddings) and HolE (Holographic Embeddings) functions offer a good balance between expressiveness and efficiency, and are is suitable for capturing complex relationships, which is desirable for the task of malware detection. Nevertheless, they are less intuitive to use and might be more complex to implement and its operations might still be demanding for resource-constrained environments.

In our analysis, we test different embeddings dimensionality (D), specifically: 5, 10, 20, 40, 80, and 100. The number of batches in which the training set must be split (batch size) during the training loop has been set to 10,000. The number of negative examples that must be generated during the training for each positive sample (ETA) has been set to 5. The process of KG representation learning is run separately for each of the embedding dimensions. The dataset KG serves as an input for the scoring function. After the learning process is finished, every node of the KG is represented as a vector of embedding values. To represent the individual application samples, the embedding vector of the "root" application node is extracted. This vector of real numbers is then used in the process of sample classification.

5.4. Binary classification

A key objective of this work is to evaluate the efficacy of KG-based embedding when used as a classification input using simple, commonly available machine learning models. No hyperparameter tuning was applied to keep the evaluation focused on embedding quality rather than model optimization. To that end, we selected 12 diverse classifiers from the Scikit-learn library,[2] covering a range of learning paradigms, as shown in Table 2.

All models are available in standard Python ML toolkits, making the approach easily reproducible and deployable. Their varied nature helps validate the KG-based embeddings ability to generalize, and they were deployed in accordance to their default settings as defined within the scikit-learn library. The parameters are listed in Appendix.

To obtain robust evaluation results, the train and test split have been selected randomly 30 times. Each train split consists of 1500 applications, 750 per class (malware vs. benign). The test split consists of 200 applications (100 per class). To ensure fair comparison and reproducibility of the results, the splits have been saved for future use.[3] To minimize potential bias in comparison, during each of the 30 iterations where the train and test splits were created, we trained and tested all classifiers on identical splits. This ensures that the performance of the classifiers for given n-dimensional embedding vectors can be directly compared.

5.5. Benchmarking

To compare the results from the embedding-based techniques, we utilized a bag-of-words (BOW) model. This model breaks down each application in the dataset into a vector containing over 50,000 features, representing various aspects of the application. From these features, a BOW feature vector was generated for each application sample. The resulting vector for each application sample has the same length as the entire dataset's BOW vector and each element in the vector denotes whether a given feature is present in an application sample (indicated by '1') or not (indicated by '0').

To compare the embedding vectors with those generated by KG representation learning, the dimensionality needs to be reduced. To achieve this, we applied principal component analysis (PCA), aiming to condense the feature space while preserving as much information as possible. We produced BOW-based embeddings with the same dimensions as the KG-based embeddings (D = 5, 10, 20, 40, 80, 100). The obtained explained variances from the reductions are: 0.59, 0.65, 0.69, 0.75, 0.80, 0.82.

To determine the accuracy loss due to the PCA reduction and to provide further comparison, an unreduced BOW-based embedding vector was also used in the process of malware detection.[4]

6. Results

This section presents the results of the empirical analysis on the applicability of knowledge graphs for malware detection. We investigate how each aspect of malware detection is affected by the elements of the KG-based pipeline. Moreover, we compare the effectiveness of KG-generated embeddings against BOW approach and feature vectors reduced using PCA, as described previously in the experiment section.

[2] Selected models are available within the Sklearn toolkit: https://scikit-learn.org/stable/.

[3] According to the principle of open-science, the data utilized for obtaining the results, individual classification splits, non-displayed accuracy figures were saved and are available at https://git.e5.ijs.si/JK_IJS/Data_for_KG_malware_analysis_study.

[4] We made a request to the authors of Rahali and Akhloufi (2021, 2023) to obtain their transformers-based malware detection model for inclusion in our analysis, but no response has been received so far.

Table 2
Overview of models selected for the experiment.

Model	Type	Justification
Support Vector Machine (SVM)	Linear/non-linear	Known for high accuracy and margin-based generalization.
Random Forest (RF)	Ensemble (Bagging)	Handles high variance well, robust to overfitting, and interpretable.
K-Nearest Neighbors (KNN)	Instance-based	Simple, non-parametric model. Good for testing embedding similarity utility.
Multi-Layer Perceptron (MLP)	Neural network	Tests non-linear relationships in KG embeddings using a shallow NN.
Logistic Regression (LR)	Linear	Baseline model. Easy to interpret and computationally efficient.
Gaussian Naive Bayes (GNB)	Probabilistic	Assumes feature independence. Serves as a baseline for feature decoupling performance.
Decision Tree (DT)	Tree-based	Tests how well features partition the input space. Fast and interpretable.
AdaBoost (AB)	Ensemble (Boosting)	Evaluates performance improvements via adaptive weight updates.
Gradient Boosting (GB)	Ensemble (Boosting)	Provides strong generalization by optimizing differentiable loss functions.
Gaussian Process (GP)	Bayesian	Tests smooth function approximation and confidence estimates.
Ridge Regression (RG)	Linear (Regularized)	Adds L2 regularization. Useful for controlling overfitting in lower-dimensional embeddings.
Passive Aggressive (PA)	Online Learning	Efficient for large-scale learning and adapts quickly to new data.

Table 3
Accuracy averaged across all dimensions and scoring functions.

	SVM	RF	KNN	MLP	LR	GNB	DT	AB	GB	GP	RG	PA
KG	0.960	0.958	**0.944**	0.959	0.939	0.855	0.897	0.937	0.953	**0.947**	0.935	0.919
PCA	0.886	0.949	0.923	0.937	0.895	0.770	0.917	0.925	0.937	0.931	0.875	0.842
BOW	**0.964**	**0.976**	0.897	**0.979**	**0.984**	**0.931**	**0.949**	**0.973**	**0.979**	0.935	**0.980**	**0.983**

Table 4
Accuracy averaged across all dimensions with separate results for the individual scoring functions.

Model	ComplEx	DistMult	HolE	TransE	PCA	BOW
SVM	0.970	0.960	**0.974**	0.939	0.886	0.964
RF	0.961	0.962	**0.967**	0.940	0.949	0.976
KNN	0.946	0.947	**0.959**	0.925	0.923	0.897
MLP	0.970	0.956	**0.973**	0.936	0.937	0.979
LR	0.957	0.933	**0.962**	0.904	0.895	0.984
GNB	0.870	0.848	**0.864**	0.838	0.770	0.931
DT	0.890	0.905	**0.917**	0.874	0.917	0.949
AB	0.942	0.936	**0.954**	0.917	0.925	0.973
GB	0.959	0.954	**0.966**	0.936	0.937	0.979
GP	0.961	0.946	**0.965**	0.915	0.931	0.935
RG	0.953	0.926	**0.956**	0.906	0.875	0.980
PA	0.942	0.907	**0.940**	0.887	0.842	0.983

6.1. Knowledge graph embeddings for malware detection

Table 3 displays the results of the malware detection accuracy, comparing three approaches: *KG embeddings, PCA, and BOW feature vectors*. The accuracies are averaged across the analyzed dimensions (5–100) for both KG and PCA, with the individual scoring functions for establishing KG embeddings averaged as well. The left column denotes the classification model used for malware detection. In majority of the cases, the BOW model outperformed both PCA and KG embeddings. However, when comparing KG embeddings to PCA, both of which rely on dimension reduction, KG embeddings demonstrated better performance in the majority of cases.

6.2. Influence of the embedding scoring function on malware detection

Table 3 does not differentiate the detection results depending on the scoring function used to generate the knowledge graph embedding. To examine the impact of scoring function on the quality of the embedding and its suitability for malware detection, Table 4 presents the same set of results, with additional separation by individual scoring functions.

When focusing solely on the embedding scoring functions, it is clear that the HolE method consistently provides the best results for malware detection. Furthermore, the performance of PCA dimension-reduced feature vectors-based detection is outperformed by KG embeddings in majority of the cases. Moreover, a comparison between HolE-based KG embeddings and BOW feature vectors reveals that HolE-based embeddings provide detection accuracy much closer to the BOW approach, and even outperform it in certain cases, despite being subject to dimension reduction and the results being averaged across all the analyzed dimensions.

The analysis above is in line with HolE's trait of offering a good balance between expressiveness and efficiency, as it captures the complex interactions characteristic for modeling malware behavior patterns. Moreover, compared to the other functions, HolE generalizes better to unseen data, which is an important property in the context of rapidly evolving malware instances.

6.3. Vector dimension influence on malware detection

To further investigate the influence of dimension on the accuracy of the malware detection, Fig. 4 displays a comparison of detection accuracy related to input vector dimension. The figure shows the averaged detection accuracy of the classification models used for HolE-based embeddings, PCA reduced vectors and BOW feature vectors.

The results indicate that detection accuracy tends to improve with increasing dimensions. As shown in Table 4, HolE-based embeddings provide the best detection accuracy. Furthermore, when compared across different dimensions, the HolE KG-based embedding vectors significantly outperform malware detection based on PCA-based feature vectors. Additionally, it can be observed that as the dimension increases, the detection accuracy based on HolE generated embeddings converges to values comparable to those obtained from BOW feature vectors, both exceeding an accuracy of 95%. For some of the low-dimensional settings, such as 5 or 10, the ComplEx scoring function-based detection slightly outperformed the HolE approach. However, the differences in the averaged accuracy in these instances are marginal, and HolE approach generally provides better results. Its ability to create low-dimensional embeddings of high-dimensional data ensures that the detection system remains lightweight and efficient.

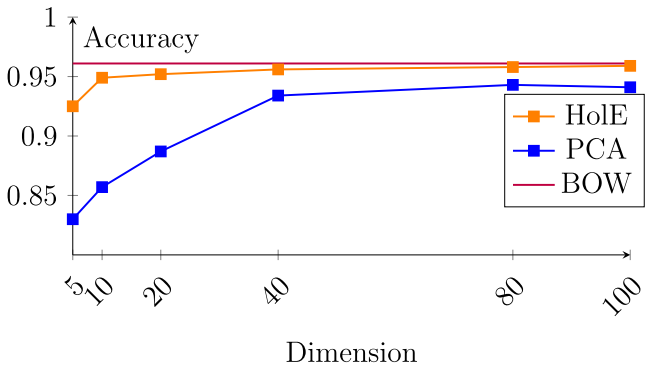


Fig. 4. Comparison of the influence of input vector dimension on malware detection accuracy for used approaches, averaged across used ML models.

6.4. Detection results stability

To analyze the stability of the detection results, we compare the relationship between input vector dimension and detection accuracy for individual classification models through a series of boxplots. Given that higher dimensions tend to result in better accuracy (as demonstrated in the previous subsections), we isolated the case of dimension = 100 for comparing HolE-based detection accuracy with PCA and BOW-based accuracies.

Fig. 5 displays the comparison between HolE and PCA accuracy results in orange and blue color for individual models used. The results are compared to the averaged accuracy of methods mentioned in Section 3, represented by the red threshold line at 95%. Overall, the median accuracy values obtained from HolE embeddings outperform those from the PCA-based approach. Additionally, the stability of the detection results based on HolE appears to be superior to those based on PCA, as indicated by the boxplots showing lower variation in the results. The comparison between HolE and BOW-based detection results in orange and purple reveals mostly similar performance and stability in the results. Although some of the results based on BOW feature vectors seem to show slightly higher stability than the HolE-based results (RF, GNB, DT, GB), the HolE shows better performance for the widely used algorithms, such as, SVM, KNN, MLP. However, despite the dimension reduction of the HolE based feature vectors, the detection results demonstrate comparable levels of performance and stability.

6.5. Classifier performance

In order to validate the quality and effectiveness of the KG-based embeddings, as well as to analyze the performance differences of individual models, we conducted a direct comparison between the analyzed state-of-the-art models.

In terms of accuracy results obtained by the detection models, the SVM, RF, and MLP algorithms consistently showed highest performance, while the GNB algorithm exhibited lower accuracy, particularly as the dimension increased.

When comparing the detection accuracy of MLP directly across the three approaches, as depicted in Fig. 6, it becomes evident that HolE-based results outperform the other approaches from D = 40 onwards, with a slight decrease in accuracy observed at D = 100. Furthermore, even for lower dimensions, HolE-based results demonstrate notably higher accuracy, especially compared to the PCA reduced feature vectors.

Table 5
Computation time (in seconds) for BOW-based detection.

	Average	Median	σ
Fitting	29.697	14.512	34.949
Detection	3.024	0.564	6.714

6.6. Model performance comparison

Additionally, both detection recall and time performance were measured during the experiment. The recall results showed similar behavior to the one of detection accuracy.

In terms of detection time performance both times of fitting and utilizing the models were measured. Fig. 7 shows the comparison between KG embedding-based and PCA-based approach.

The results have been averaged across all utilized detection models and separated based on differing dimensions of input vectors. It can be observed that the time constraints of both fitting and detection are increasing with increasing vector dimension. However, the averaged performance of both methods remains basically identical, with the embedding-based results minimally outperforming the PCA approach.

Furthermore, the BOW-based detection approach introduces significantly higher requirements in terms of fitting and detection performance compared to approaches that focus on reducing dimensions. Table 5 shows the averaged results for fitting and detection times for the BOW vectors.

7. Discussion and future work

The experimental analysis demonstrates the potential that knowledge graphs hold for the purpose of malware detection, showing that the KG-based results are comparable to the state-of-the-art approaches. This reaffirms and further refines the theoretical comparative analysis presented in Section 3.1, in Table 1. Employing KG embeddings to transform the processed data into smaller dimensional spaces leads to a notable decrease of computational and storage requirements. Moreover, the high levels of detection performance are retained, with less information loss compared to the PCA dimension reduction approach.

While the BOW approach has proven effective in malware detection and can sometimes yield better performance, it comes at the expense of exponentially higher computational and storage requirements and a significantly larger feature space (100 vs 50,000 dimensions). Therefore, the BOW approach may become useless for datasets with higher number of applications and extracted features. In contrast, the more scalable KG dataset representation allows for more efficient data manipulation, processing and storing. Finally, our findings from the detection accuracy analysis indicate that KG-based embeddings, regardless of the used scoring function and dimension, outperform PCA-based vector reduction in majority of the cases. Considering the comparable performance to GNN-based approaches from Table 1, our work is a strong testament for the general usability of the KG-based approach for the purpose of malware detection.

One of the major benefits for Android malware detection lies in the simplification of the feature space of the malware dataset. KG-based representation not only introduces scalability in working with datasets and selected features, but it also allows for capturing the individual relations between different applications and application features in the input data, without explicit feature processing. This leads to increased information density in the generated embeddings, regardless of vector dimension.

Simplification of the feature space, along with improved information density, enable better performance of simpler classification models in the malware detection pipeline, while allowing the sample processing to account for the semantic relationships in the dataset. Despite the simplicity and lack of fine-tuning of the used models, detection performance remained stable. Additionally, the simpler nature of the models

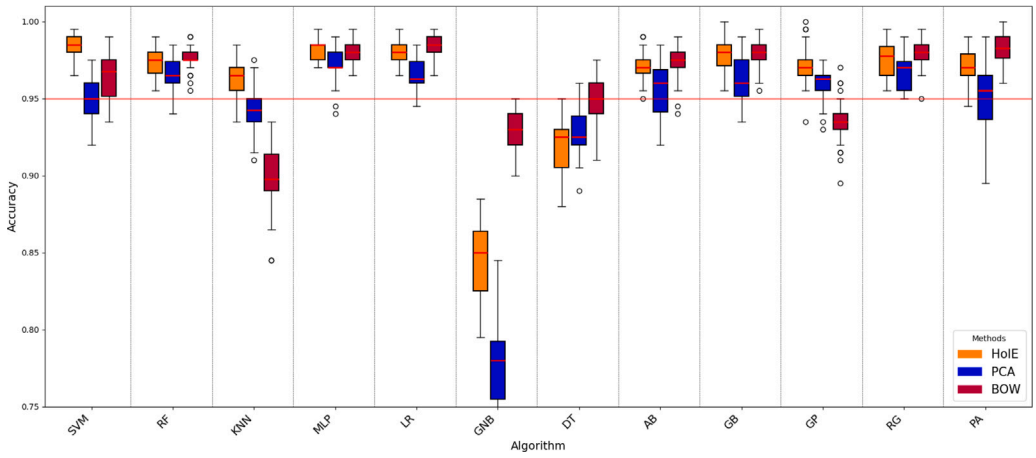


Fig. 5. Comparison of detection accuracy of individual models utilizing input data based on D = 100 HolE and PCA, and BOW feature vectors.

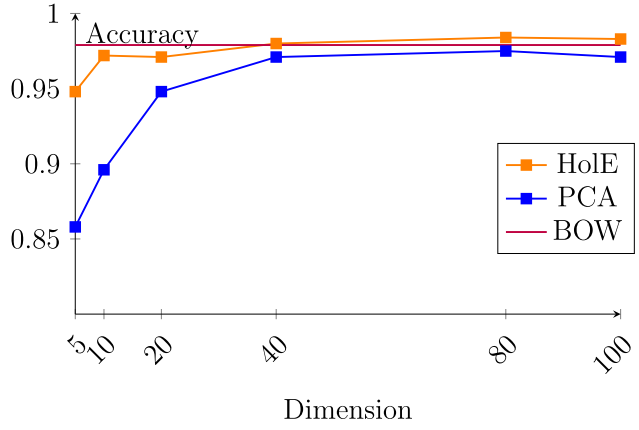


Fig. 6. Comparison of the influence of input vector dimension on malware detection accuracy for used approaches, comparing results of MLP algorithm.

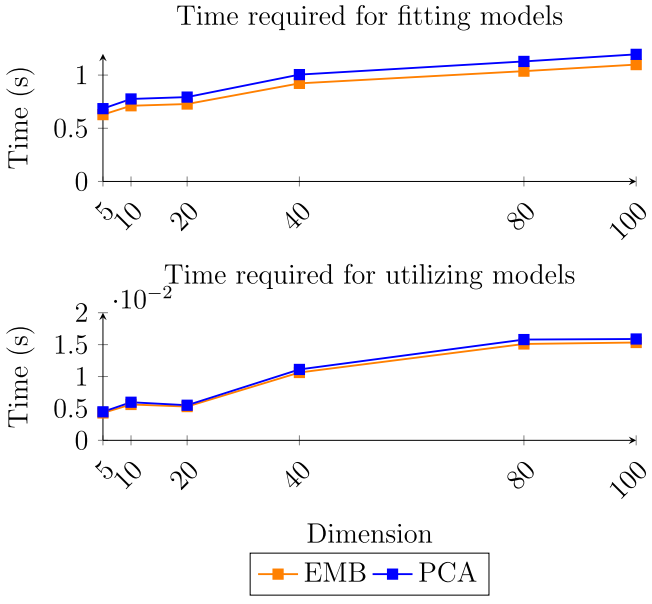


Fig. 7. Comparison of the influence of input vector dimension on both fitting and model utilizing computational requirements for dimension reducing approaches.

did not compromise the accuracy, which remained high while reducing the computational complexity of the detection pipeline for both model fitting and utilization tasks. This presents another important benefit for the domain of malware detection, highlighting the importance of focusing on refining the data rather than solely refining the models. It is important to note here that the "simplicity" of models in our work is not a compromise, but a deliberate design decision guided by the strive for data-centricity. By prioritizing semantic richness and data compactness, we demonstrate that classification performance can remain competitive with much more complex architectures.

One limitation of the approach proposed in this work stems from the inherent nature of static analysis that are ineffective in the discovery of out-of-sample malware. Thus, the current form of the KG representation only considers the static snapshot of the application in the graph. However, this choice was done by design, as the study aimed to carry out comprehensive empirical analysis of the applicability of knowledge graphs, validating the current model design choices and setting a baseline for subsequent model extensions. Hence, this limitation will be addressed in our future work, in which we plan to interface the present model with external knowledge bases, as well as with other methods for knowledge representation and discovery.

Another limitation is the use of undirected edges for the knowledge graph creation, which may limit the representation of the nodes' relationships in terms of directionality and informativeness. Our future work will explore the benefits of using other types of edges or a combination thereof and analyze their impact on performance and resource usage for the resource-constrained environment.

Finally, the current method is transductive, i.e., if a new data point is added to the testing dataset, the learning model has to be retrained. Adding inductive capability to the model, i.e. enabling prediction of new data based on an observed training set or by including unseen samples by similarity comparison may allow updating of the KG with new applications and make use of the embeddings stored in the vector database without the need to retrain the entire model. We intend to train the model exclusively using applications from specific malware families and assess its performance on new malware families in order to evaluate its generalization capabilities to unforeseen families that may emerge in the future.

8. Conclusion

This study investigates the potential of knowledge graphs for mobile Android malware detection, offering evidence-based support for the various benefits of their application. The empirical analysis carried out in this work offer insights on the influence of the scoring functions, the vector dimension, the stability of the obtained results, the performance

of the individual classifiers, and other time dependencies that are part of the KG-based malware detection pipeline.

Leveraging KG embeddings to represent the Android applications, the paper introduced an alternative approach for malware detection that improves the quality of classification input data, competing with the current state-of-the-art methods. In our comprehensive benchmarking experiment, we evaluated the effectiveness of four KG embedding scoring functions across six dimensions (5, 10, 20, 40, 80, 100). These embeddings were utilized to generate input feature vectors for a malware classifier employing 12 state-of-the-art machine-learning algorithms. For comparison, we employed PCA dimension reduction to provide feature vectors of identical dimensions, while a BOW model was utilized to create full feature vectors based on the analyzed application dataset. The obtained results are highly promising, with the highest recorded KG embedding-based accuracy exceeding 97%. Additionally, the performance of KG embeddings surpassed that of the comparative PCA-reduced vectors and, in certain cases, even outperformed the BOW-based approach. Furthermore, KG-based classification demonstrated strong performance even with lower-dimensional input vectors, offering higher information density and reducing both computational and data storage requirements, especially when compared to the BOW approach.

Finally, the paper presented a rich discussion on the results, pointing out some of the limitations of the proposed method and potential directions to address them as part of our future work. The results and contributions of our work show the significant potential of using KG as a source for generating embeddings to represent analyzed datasets in the process of detecting malware applications. Moreover, they provide deeper insights into the inter-dependencies between the separate components of the malware detection pipeline, validating the practical utility of the approach.

CRediT authorship contribution statement

Jan Kincl: Data curation, Formal analysis, Funding acquisition, Investigation, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Tome Eftimov:** Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Writing – review & editing. **Adam Viktorin:** Conceptualization, Investigation, Writing – review & editing. **Roman Šenkeřík:** Conceptualization, Funding acquisition, Writing – review & editing. **Tanja Pavleska:** Conceptualization, Formal analysis, Investigation, Resources, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Funding in direct support of this work: The Internal Grant Agency of Tomas Bata University under the project no. IGA/CebiaTech/2023/004, the resources of A.I.Lab at the Faculty of Applied Informatics, Tomas Bata University in Zlin (ailab.fai.utb.cz), the financial support of the Tomas Bata University's Erasmus+ program for traineeship mobility supported by the European Union, and the Slovenian Research and Innovation Agency through program grants No. P2-0098 and No. P2-0037, and project No. GC-0001.

Appendix. Default parameters of classifiers

SVM: *C=1.0; break_ties=False; cache_size=200; class_weight=None; coef0=0.0; decision_function_shape='ovr'; degree=3; gamma='scale'; kernel='rbf'; max_iter=−1; probability=False; random_state=None; shrinking=True; tol=0.001; verbose=False*| **RF**: *bootstrap=True; ccp_alpha=0.0; class_weight=None; criterion='gini'; max_depth=None; max_features='sqrt'; max_leaf_nodes=None; max_samples=None; min_impurity_decrease=0.0; min_samples_leaf=1; min_samples_split=2; min_weight_fraction_leaf=0.0; monotonic_cst=None; n_estimators=100; n_jobs=None; oob_score=False; random_state=None; verbose=0; warm_start=False*| **KNN**: *algorithm='auto'; leaf_size=30; metric='minkowski'; metric_params=None; n_jobs=None; n_neighbors=5;p = 2; weights='uniform*| **MLP**: *activation='relu'; alpha=0.0001; batch_size='auto'; beta_1=0.9; beta_2=0.999; early_stopping=False; epsilon=1e-08; hidden_layer_sizes=(100,); learning_rate='constant'; learning_rate_init=0.001; max_fun=15000; max_iter=200; momentum=0.9; n_iter_no_change=10; nesterovs_momentum=True; power_t=0.5; random_state=None; shuffle=True; solver='adam'; tol=0.0001; validation_fraction=0.1; verbose=False; warm_start=False*| **LR**: *C=1.0; class_weight=None; dual=False; fit_intercept=True; intercept_scaling=1; l1_ratio=None; max_iter=100; multi_class='auto'; n_jobs=None; penalty='l2'; random_state=None; solver='lbfgs'; tol=0.0001; verbose=0; warm_start=False*| **GaussianNB**: *priors=None; var_smoothing=1e-09*| **DT**: *ccp_alpha=0.0; class_weight=None; criterion='gini'; max_depth=None; max_features=None; max_leaf_nodes=None; min_impurity_decrease=0.0; min_samples_leaf=1; min_samples_split=2; min_weight_fraction_leaf=0.0; monotonic_cst=None; random_state=None; splitter='best'*| **AB**: *algorithm='SAMME.R'; base_estimator=None; learning_rate=1.0; n_estimators=50; random_state=None*| **GB**: *ccp_alpha=0.0; criterion='friedman_mse'; init=None; learning_rate=0.1; loss='deviance'; max_depth=3; max_features=None; max_leaf_nodes=None; min_impurity_decrease=0.0; min_samples_leaf=1; min_samples_split=2; min_weight_fraction_leaf=0.0; n_estimators=100; n_iter_no_change=None; random_state=None; subsample=1.0; tol=0.0001; validation_fraction=0.1; verbose=0; warm_start=False*| **GP**: *copy_X_train=True; kernel=None; max_iter_predict=100; multi_class='one_vs_rest'; n_jobs=None; n_restarts_optimizer=0; optimizer='fmin_l_bfgs_b'; random_state=None; warm_start=False*| **RG**: *alpha=1.0; class_weight=None; copy_X=True; fit_intercept=True; max_iter=None; positive=False; random_state=None; solver='auto'; tol=0.0001*| **PA**: *C=1.0; average=False; class_weight=None; early_stopping=False; fit_intercept=True; loss='hinge'; max_iter=1000; n_iter_no_change=5; n_jobs=None; random_state=None; shuffle=True; tol=0.001; validation_fraction=0.1; verbose=0; warm_start=False*.

Data availability

Data will be made available on request.

References

Adebayo, O. S., & Aziz, N. A. (2019). Improved malware detection model with apriori association rule and particle swarm optimization. *Security and Communication Networks*, http://dx.doi.org/10.1155/2019/2850932.

Alazab, M., Priya, R. M. S., Parimala, M., Maddikunta, P. K. R., Gadekallu, T. R., & Pham, Q.-V. (2022). Federated learning for cybersecurity: concepts, challenges, and future directions. *IEEE Transactions on Industrial Informatics*, http://dx.doi.org/10.1109/TII.2021.3119038.

Amin, M., Tanveer, T. A., Tehseen, M., Khan, M., Khan, F. A., & Anwar, S. (2020). Static malware detection and attribution in android byte-code through an end-to-end deep system. *Future Generation Computer Systems*, http://dx.doi.org/10.1016/j.future.2019.07.070.

Android Developers (2023). Application fundamentals. https://developer.android.com/guide/components/fundamentals.

Arif, J. M., Razak, M. F. A., Awang, S., Tuan Mat, S. R., Ismail, N. S. N., & Firdaus, A. (2021). A review: static analysis of android malware and detection technique. In *2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM)* (pp. 580–585). http://dx.doi.org/10.1109/ICSECS52883.2021.00112.

Arslan, R. S., Doğru, İ. A., & Bariş¸ci, N. (2019). Permission-based malware detection system for android using machine learning techniques. *Int J Softw Eng Knowl Eng*, http://dx.doi.org/10.1142/S0218194019500037.

Bitdefender (2024). Large language models for malware analysis. https://bit-ml.github.io/blog/post/large-language-models-for-malware-analysis/.

Ceci, L. (2024). Mobile internet usage worldwide - statistics & facts. https://www.statista.com/topics/779/mobile-internet/#topicOverview (Accessed: 16.5.2024).

Chen, X., Jia, S., & Xiang, Y. (2020). A review: knowledge reasoning over knowledge graph. *Expert Systems with Applications*.

Christian, R., Dutta, S., Park, Y., & Rastogi, N. (2021). Ontology-driven knowledge graph for android malware. https://doi.org/10.48550/arXiv.2109.01544.

Costabello, L., Bernardi, A., Janik, A., Pai, S., Van, C. L., McGrath, R., et al. (2019). Ampligraph: a library for representation learning on knowledge graphs. http://dx.doi.org/10.5281/zenodo.2595043.

Demirkıran, F., Çayır, A., Ünal, U., & Dağ, H. (2022). An ensemble of pre-trained transformer models for imbalanced multiclass malware classification. arXiv:2112.13236 (cs.CR).

Dimjašević, M., Atzeni, S., Ugrina, I., & Rakamaric, Z. (2016). Evaluation of android malware detection based on system calls. http://dx.doi.org/10.1145/2875475.2875487.

Enck, W. (2011). Defending users against smartphone apps: techniques and future directions. In *Information Systems Security: 7th International Conference, ICISS 2011, Kolkata, India, December 15-19, 2011, Procedings 7* (pp. 49–70). Springer.

EOL (2024). Android end-of-life. https://endoflife.date/android (Accessed: 16.5.2024).

Fang, C., Miao, N., Srivastav, S., Liu, J., Zhang, R., Fang, R., et al. (2024). Large language models for code analysis: do llms really do their job?.

FBI (2013). Roll call release - threats to mobile devices using the android operating system. https://info.publicintelligence.net/DHS-FBI-AndroidThreats.pdf (Accessed: 16.5.2024).

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). Androdialysis: analysis of android intent effectiveness in malware detection. *Computers & Security*, http://dx.doi.org/10.1016/j.cose.2016.11.007.

Firdaus, A., Anuar, N. B., Karim, A., & Razak, M. F. A. (2018). Discovering optimal features using static analysis and a genetic search-based method for android malware detection. *Frontiers of Information Technology & Electronic Engineering*, http://dx.doi.org/10.1631/FITEE.1601491.

Gao, H., Cheng, S., & Zhang, W. (2021). Gdroid: android malware detection and classification with graph convolutional network. *Computers & Security*, *106*, 102264. http://dx.doi.org/10.1016/j.cose.2021.102264, https://www.sciencedirect.com/science/article/pii/S0167404821000882.

Ghimire, B., & Rawat, D. B. (2022). Recent advances on federated learning for cybersecurity and cybersecurity for federated learning for internet of things. *IEEE Internet of Things Journal*, http://dx.doi.org/10.1109/JIOT.2022.3150363.

Hasegawa, C., & Iyatomi, H. (2018). One-dimensional convolutional neural networks for android malware detection. In *2018 IEEE 14th International Colloquium on Signal Processing & its Applications (CSPA)*. http://dx.doi.org/10.1109/CSPA.2018.8368693.

Hei, Y., Yang, R., Peng, H., Wang, L., Xu, X., Liu, J., et al. (2024). Hawk: rapid android malware detection through heterogeneous graph attention networks. *IEEE Transactions on Neural Networks and Learning Systems*, http://dx.doi.org/10.1109/TNNLS.2021.3105617.

Imtiaz, S. I., ur Rehman, S., Javed, A. R., Jalil, Z., Liu, X., & Alnumay, W. S. (2021). Deepamd: detection and identification of android malware using high-efficient deep artificial neural network. *Future Generation Computer Systems*, *115*, 844–856. http://dx.doi.org/10.1016/j.future.2020.10.008, https://www.sciencedirect.com/science/article/pii/S0167739X2032985X.

Jiang, W., Han, H., Zhang, Y., Mu, J., & Shankar, A. (2024). Intrusion detection with federated learning and conditional generative adversarial network in satellite-terrestrial integrated networks. *Mobile Networks and Applications*, http://dx.doi.org/10.1007/s11036-024-02435-4.

Jusoh, R., Firdaus, A., Anwar, S., Osman, M. Z., Darmawan, M. F., & Razak, M. F. A. (2021). Malware detection using static analysis in android: a review of feco (features, classification, and obfuscation). *PeerJ Computer Science*, *7*, https://api.semanticscholar.org/CorpusID:236150110.

Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). Maldozer: automatic framework for android malware detection using deep learning. *Digital Investigation*, *24*, S48–S59.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, http://dx.doi.org/10.1109/TII.2017.2789219.

Li, W., Wang, Z., Cai, J., & Cheng, S. (2018). An android malware detection approach using weight-adjusted deep learning. In *2018 international conference on computing, networking and communications (ICNC)* (pp. 437–441). IEEE.

Liu, X., Lin, Y., Li, H., & Zhang, J. (2020). A novel method for malware detection on ml-based visualization technique. *Computers & Security*, http://dx.doi.org/10.1016/j.cose.2019.101682.

Ma, D., Bai, Y., Xing, Z., Sun, L., & Li, X. (2020). A knowledge graph-based sensitive feature selection for android malware classification. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 188–197). http://dx.doi.org/10.1109/APSEC51365.2020.00027.

Mahdavifar, S., Abdul kadir, A. F., Fatemi, R., Alhadidi, D., & Ghorbani, A. (2020). Dynamic android malware category classification using semi-supervised deep learning. http://dx.doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.

Mahdavifar, S., Alhadidi, D., & Ghorbani, A. A. (2022). Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of Network and Systems Management*, http://dx.doi.org/10.1007/s10922-021-09634-4.

Malhotra, V., Potika, K., & Stamp, M. (2024). A comparison of graph neural networks for malware classification. *Journal of Computer Virology and Hacking Techniques*, http://dx.doi.org/10.1007/s11416-023-00493-y.

Martín García, A., Lara-Cabrera, R., & Camacho, D. (2018). Android malware detection through hybrid features fusion and ensemble classifiers: the andropytool framework and the omnidroid dataset. *Information Fusion*, http://dx.doi.org/10.1016/j.inffus.2018.12.006.

Martin Garcia, A., Lara-Cabrera, R., & Camacho, D. (2018). A new tool for static and dynamic android malware analysis. http://dx.doi.org/10.1142/9789813273238_0066.

Mathur, A., Podila, L. M., Kulkarni, K., Niyaz, Q., & Javaid, A. Y. (2021). Naticusdroid: a malware detection framework for android using native and custom permissions. *Journal of Information Security and Applications*, http://dx.doi.org/10.1016/j.jisa.2020.102696.

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., et al. (2017). Deep android malware detection. In *Proceedings of the seventh ACM on conference on data and application security and privacy*. New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3029806.3029823.

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., et al. (2022). A systematic overview of android malware detection. *Applied Artificial Intelligence*, http://dx.doi.org/10.1080/08839514.2021.2007327.

Meng, G., Xue, Y., Siow, J. K., Su, T., Narayanan, A., & Liu, Y. (2017). Androvault: constructing knowledge graph from millions of android apps for automated analysis. https://doi.org/10.48550/arXiv.1711.07451.

Morgan, S. (2021). Cybercrime to cost the world $10.5 trillion annually by 2025. https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/ (Accessed: 16.5.2024).

Nokia (2021). Threat intelligence report 2021. https://vpnoverview.com/wp-content/uploads/nokia_threat_intelligence_report_2021_report_en.pdf (Accessed: 16.5.2024).

OWID (2024). Global damage costs from natural disasters, 1980 to 2022. https://ourworldindata.org/grapher/damage-costs-from-natural-disasters (Accessed: 16.5.2024).

Pandey, S. K., & Mehtre, B. (2014). A lifecycle based approach for malware analysis. In *2014 Fourth International Conference on Communication Systems and Network Technologies* (pp. 767–771). http://dx.doi.org/10.1109/CSNT.2014.161.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: machine learning in python. *Journal of Machine Learning Research*.

Phillips, A. (2022). Android antivirus vulnerabilities. https://www.comparitech.com/antivirus/android-antivirus-vulnerabilities/ (Accessed: 16.5.2024).

Rahali, A., & Akhloufi, M. A. (2021). Malbert: malware detection using bidirectional encoder representations from transformers. In *2021 IEEE international conference on systems, man, and cybernetics*. http://dx.doi.org/10.1109/SMC52423.2021.9659287.

Rahali, A., & Akhloufi, M. A. (2023). Malbertv2: code aware bert-based model for malware identification. *Big Data and Cognitive Computing*, http://dx.doi.org/10.3390/bdcc7020060.

Rastogi, N., Dutta, S., Zaki, M. J., Gittens, A., & Aggarwal, C. C. (2020). Malont: an ontology for malware threat intelligence. https://doi.org/10.48550/arXiv.2006.11446, CoRR.

Razak, M. F. A., Anuar, N. B., Othman, F., Firdaus, A., Afifi, F., & Salleh, R. (2018). Bio-inspired for features optimization and malware detection. *Arab J Sci Eng*.

Sabbah, A., Kharma, M., & Jarrar, M. (2023). Creating android malware knowledge graph based on a malware ontology.

Shiqi, L., Shengwei, T., Long, Y., Jiong, Y., & Hua, S. (2018). Android malicious code classification using deep belief network. *KSII Transactions on Internet & Information Systems*, *12*(1).

StatCounter (2024a). Desktop vs mobile vs tablet market share worldwide. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet (Accessed: 16.5.2024).

StatCounter (2024b). Mobile operating system market share worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide (Accessed: 16.5.2024).

StatCounter (2024c). Operating system market share worldwide. https://gs.statcounter.com/os-market-share (Accessed: 16.5.2024).

Statista (2024). Mobile android operating system market share worldwide from january 2018 to july 2023, by version. https://www.statista.com/statistics/921152/mobile-android-version-share-worldwide/ (Accessed: 16.5.2024).

Taheri, R., Ghahramani, M., Javidan, R., Shojafar, M., Pooranian, Z., & Conti, M. (2020). Similarity-based android malware detection using hamming distance of static binary features. *Future Generation Computer Systems*, *105*, 230–247. http://dx.doi.org/10.1016/j.future.2019.11.034, https://www.sciencedirect.com/science/article/pii/S0167739X19315122.

Thomas, D., Beresford, A., & Rice, A. (2015). *Security metrics for the Android ecosystem*. http://dx.doi.org/10.17863/CAM.27064.

Tian, K., Yao, D., Ryder, B. G., Tan, G., & Peng, G. (2020). Detection of repackaged android malware with code-heterogeneity features. *IEEE Transactions on Dependable and Secure Computing*, http://dx.doi.org/10.1109/TDSC.2017.2745575.

Toulas, B. (2023). Android malware infiltrates 60 google play apps with 100m installs. https://www.bleepingcomputer.com/news/security/android-malware-infiltrates-60-google-play-apps-with-100m-installs/ (Accessed: 16.5.2024).

Venkatasubramanian, M., Habibi Lashkari, A., & Hakak, S. (2023). Federated learning assisted iot malware detection using static analysis. In *Proceedings of the 2022 12th international conference on communication and network security*. New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3586102.3586131.

Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2017). Deep android malware detection and classification. In *2017 international conference on advances in computing, communications and informatics*. http://dx.doi.org/10.1109/ICACCI.2017.8126084.

Wang, S., Chen, Z., Yan, Q., Ji, K., Peng, L., Yang, B., et al. (2020). Deep and broad url feature mining for android malware detection. *Information Sciences*, http://dx.doi.org/10.1016/j.ins.2019.11.008.

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). *Knowledge graph embedding: A survey of approaches and applications*. IEEE.

Wu, J., Qiu, G., Wu, C., Jiang, W., & Jin, J. (2024). Federated learning for network attack detection using attention-based graph neural networks. *Scientific Reports*, http://dx.doi.org/10.1038/s41598-024-70032-2.

Xiao, X., Zhang, S., Mercaldo, F., Hu, G., & Sangaiah, A. K. (2019). Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications*, http://dx.doi.org/10.1007/s11042-017-5104-0.

Yerima, S. Y., Sezer, S., & McWilliams, G. (2014). Analysis of bayesian classification-based approaches for android malware detection. *IET Information Security*, *8*(1), 25–36. http://dx.doi.org/10.1049/iet-ifs.2013.0095, https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ifs.2013.0095.

Zhao, Y., Sun, S., Huang, X., & Zhang, J. (2025). An android malware detection method using frequent graph convolutional neural networks. *Electronics*, http://dx.doi.org/10.3390/electronics14061151.