Identifying Patterns of Vulnerability Incidence in Foundational Machine Learning Repositories on GitHub: An Unsupervised Graph Embedding Approach

Agrim Sachdeva
Kelley School of Business
Indiana University
Bloomington, Indiana, USA
agsach@iu.edu

Ben Lazarine
Kelley School of Business
Indiana University
Bloomington, Indiana, USA
belazar@iu.edu

Ruchik Dama
Kelley School of Business
Indiana University
Bloomington, Indiana, USA
rdama@iu.edu

Sagar Samtani
Kelley School of Business
Indiana University
Bloomington, Indiana, USA
ssamtani@iu.edu

Hongyi Zhu
Alvarez College of Business
University of Texas
San Antonio, Texas, USA
hongyi.zhu@utsa.edu

2022 IEEE International Conference on Data Mining Workshops (ICDMW) | 979-8-3503-4609-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDMW58026.2022.00084

***Abstract*—The rapid pace of the development of artificial intelligence (AI) solutions is enabled by leveraging foundational tools and frameworks that allow AI developers to focus on application logic and rapid prototyping. However, the security vulnerabilities present in foundation repositories might cause irreparable damage due to the AI solutions built using these libraries being deployed in production environments. Our research leverages source code hosted on the prevailing social coding platform GitHub to identify vulnerabilities in foundational repositories commonly used for modern AI development (Linux, BERT, PyTorch, and Transformers), as well as the AI repositories that utilize foundation repositories as dependencies. Using an unsupervised graph embedding approach, we generate graph embeddings that capture vulnerability information and the relationships between repositories. Based on these embeddings, we performed clustering as our downstream task to group similarly vulnerable repositories. Our research identifies patterns and similarities between repositories and will help develop effective mitigation of vulnerabilities present in groups of repositories based on foundational AI repositories. We also discuss the implications of identifying such clusters of vulnerable repositories.**

Keywords—Artificial Intelligence, machine learning, GitHub, vulnerability scanning, graph embedding, clustering, open source software security

I. Introduction

Artificial intelligence (AI) is "the ability of a machine to perform cognitive functions that we associate with human minds, such as perceiving, reasoning, learning, interacting with the environment, problem-solving, decision-making, and even demonstrating creativity" [1, p. iii]. Creating computer programs and machines capable of performing tasks that humans are naturally capable of imparts cognitive capabilities to machines such as sensing, comprehending, acting, and learning [2, 3]. Within AI, machine learning (ML) is an umbrella term for a set of techniques and tools that help computers learn and adapt on their own [4, 5]. A resurgence in AI use is attributed to the subclass of machine learning known as deep learning (DL), because of its advances and phenomenal successes in fields such as computer vision, boosting its application across multiple industries [6].

AI research finds itself in the third boom of its history, fueled by increased funding, scientific breakthroughs, and widespread public speculation about the scope and impact of these breakthroughs [7]. AI's use in production business systems is increasing rapidly, with estimates that it may increase the global GDP by $15.7 trillion by 2030 [8]. Applications of AI are becoming ubiquitous across financially critical business processes, safety-critical systems like transportation and medicine, as well as business applications, making the reliability and safety of its performance essential [9]. New machine learning methods have revolutionized many industries, including smart healthcare, financial technology, and surveillance [10].

The rapid development of applied AI-based solutions across many domains has been made possible due to foundational open-source libraries and DL frameworks available on GitHub, such as Torch, TensorFlow, and transformers [11]. These frameworks accelerate AI development since they have pre-implemented neural network layers and can allow developers to focus on the application of the method. Therefore, they can build models by training them on specific use cases without worrying about low-level implementation such as input parsing, matrix operations, or optimization [12]. However, one downside of the rapid and passionate implementation of AI applications is overlooking the software security vulnerabilities present in foundational libraries. These repositories are often referred to (i.e., depended on) when many AI researchers are developing their models. As such, vulnerabilities (e.g., insecure coding

2375-9259/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDMW58026.2022.00084

practices, secrets such as passwords, and API keys) within these foundational repositories could be propagated to applications utilizing them [12]. Exploiting these issues can lead to software supply chain cyber-attacks and significant information and financial losses [13-16].

In this research, we aim to identify vulnerable clusters of repositories that depend upon foundational repositories commonly used in the AI model development process (e.g., Linux, BERT, PyTorch, Transformers). We focus on GitHub, one of the most prominent host of publicly available AI and ML software, with 700 AI tools (i.e., frameworks and libraries) and 4,524 applied AI repositories hosted in 2020 [17]. Further, as determined by the number of stars, i.e. a measure of popularity on GitHub, foundational machine learning repositories comprise 6 of the 10 most popular repositories [18]. We use a sample of 207 repositories that had forked the four foundational repositories and conduct a vulnerability assessment on these repositories using an open-source static application security testing (SAST) tool. We represent the relationships between the repositories as a graph and utilize vulnerability assessment results as nodal features. A downstream clustering task identifies four distinct clusters of repositories.

The remainder of this paper is organized as follows. First, we review the literature on vulnerabilities in foundational GitHub repositories and graph embedding techniques. Second, we present our research gaps and questions. Third, we describe our methodology. Fourth, we summarize our results. Finally, we discuss the implications of our research and future work.

II. Prior Research

To set the foundation for this research, we review two areas of literature. First, we describe previous research that analyzes vulnerabilities in foundational GitHub repositories. Next, we summarize graph embedding techniques that can allow for identifying similarities between the repositories.

A. Vulnerabilities in Foundational GitHub Repositories

We summarize selected studies related to vulnerability analysis in foundational libraries and repositories on GitHub, specifically ML/ DL frameworks. Early research sought to identify vulnerabilities within foundational AI repositories through conventional software bug identification processes. For example, in an empirical study by Jia et al. [19] on implementation bugs of the foundational framework, the authors discovered that TensorFlow (used by over 36K projects) has bugs similar to traditional software systems. Di Franco et al. [20] studied bugs in foundational libraries such as NumPy and SciPy and found that a third of the bugs were related to Numeric. Focusing on a broader array of foundational repositories, Guo et al. [21] systematically study the impact of four DL frameworks, i.e., TensorFlow, PyTorch, CNTK, and MXNET, on software development and deployment processes.

Despite revealing essential insights, the studies above did not examine the contents within the vulnerable repositories to identify potential vulnerabilities. Recent research has utilized natural language processing (NLP) techniques to study the discourse concerning foundational ML/ DL repositories. Han et al. [22] applied Latent Dirichlet Allocation (LDA) to compare the discussion topics for TensorFlow, PyTorch, and Theano on GitHub and Stack Overflow. Harzevili et al. [23] study GitHub commits in foundational repositories, namely Pandas, NumPy, Scikit-Learn, PyTorch, and TensorFlow, that mention CVEs present in the National Vulnerability Database (NVD). Following a similar approach, some studies have also studied open-source operating systems and protocols. An exemplar study is Jimenez et al. [24], which analyzed git commit messages of the Linux kernel and OpenSSL that reported CVE numbers. Alfadel et al. [25] studied the time taken to fix vulnerabilities in PyPi projects. They discovered that Cross-Site-Scripting (XSS) was the most common vulnerability reported in PyPi projects.

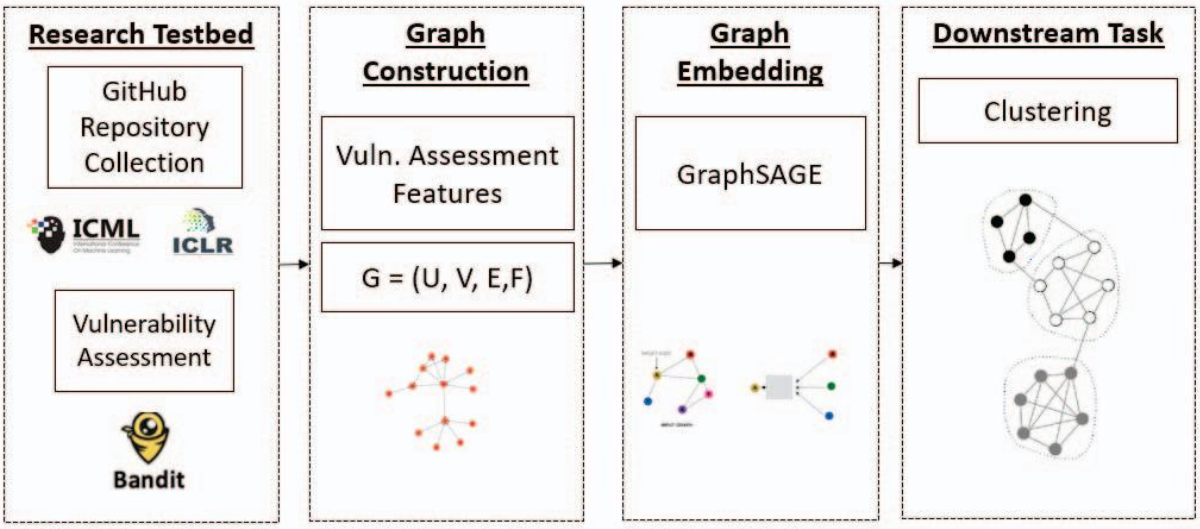
Dependency networks for DL libraries are crucial to identify given the rapid development of DL libraries. Recognizing this, Han et al. [26] explored dependency networks for foundational DL libraries, specifically TensorFlow, PyTorch, and Theano. However, this study did not capture the vulnerabilities of each repository when examining dependency networks. This, combined with the focus of prior studies examining foundational AI repositories, underscores the need for an approach that can examine the repositories based on foundational AI repositories to identify groups of repositories sharing similar sets of vulnerabilities that could be remediated in a targeted fashion.

GitHub has several code collaboration and social network features, and those rich relationships must be captured [27]. Foundational repositories on GitHub are often forked or declared as dependencies, such that they can be built upon or used for an applied AI task [28]. These relationships can effectively be captured using a graph representation, where the repositories are represented as nodes, and relationships between them, such as forks and dependencies, are represented as edges. The generation of these networks can enable downstream tasks such as clustering [29]. Facilitating these tasks requires a mechanism to automatically represent each node as a low-dimensional feature vector (i.e., embedding). Therefore, we review the principles of graph embedding techniques in the following sub-section.

B. Graph Embedding Techniques

Graph embedding methods seek to use a set of functions to automatically produce a low-dimensional feature vector (i.e., embedding) for an aspect of a graph (e.g., node, edge, sub-graph, etc.). Since we need to capture information about the repositories, i.e., we need to incorporate features at the nodal level, we focus on methods that can account for that. Further, the information in the neighboring repositories is essential when considering the spread and incidence of vulnerabilities from foundational repositories. We focus on unsupervised graph representation learning techniques since we lack labels for the data and seek to transform graph components into a lower dimensional vector space while maximally preserving information about the repositories and the relationships between repositories [30].

One of the proposed taxonomies to classify graph embedding methods differentiates techniques into spatial and spectral categories [31]. Spatial graph embeddings place graphs in space such that nodes that are connected should be placed near each other, while nodes that are not connected are placed apart in the n-dimensional space [32]. On the other hand, spectral



embedding algorithms preserve network structure by generating vector representations of nodes from eigenvectors of an adjacency matrix or the Laplacian matrix [33].

In our review of GCN methods, we provide an overview of two spatial embedding techniques and one spectral embedding technique. The two spatial embedding techniques reviewed were GraphSAGE and DeepGraph InfoMax, and the spectral embedding technique reviewed was ConvGNN.

- maximizing the information that is common across the nodes [35].
- ConvGNNs are a representative spectral method that utilizes graph-based analogs of convolutional architectures, similar to the domain of signal processing [36, 37].

For our use case, we use a spatial model and use GraphSAGE to create graph embeddings. Spatial models are efficient and flexible, they are more scalable, and are not limited to operating on undirected graphs [38].

III. Research Gaps and Questions

We identified several research gaps in our literature review. First, prior research on identifying vulnerabilities in foundational AI repositories is mostly limited to utilizing empirical methods, and relationships between the repositories are not captured when analyzing the vulnerabilities. Second, most studies rely on developer-reported vulnerabilities, such as CVE information that is shared in commit messages, and do not scan the repositories for vulnerabilities. Third, vulnerability information is not used to identify groups of repositories that would be very beneficial to mitigate these vulnerabilities systematically. Based on these gaps, we propose the following research questions:

- How can we identify vulnerabilities in foundational AI development repositories on GitHub?
- GraphSAGE is a general inductive framework that uses aggregator functions to learn the aggregator feature information from the node's local neighborhood for node embeddings [34].
- Deep Graph InfoMax constructs a graph embedding by relying on mutual information maximization, i.e.,
- How can we identify groups of repositories that depend on foundational AI repositories based on their vulnerabilities?

IV. Methodology

We present our proposed research framework in Fig. 1. Research Testbed, Graph Construction, Graph Embedding, and Downstream Task. We describe each component of the proposed framework in further detail in the following sub-sections.

A. Research Testbed

Our initial data collection consisted of three phases to identify and collect all ML/DL repositories on GitHub from the AI community: (1) AI GitHub repository identification, (2) foundational repository identification, and (3) vulnerability assessment. For the AI GitHub repository identification, we identified top AI conferences to identify AI papers from the past two years to collect. We focus on AI models published in top AI conferences as these are important repositories that will be used for several applied AI tasks. Next, we extracted GitHub links from the identified AI papers and downloaded all associated repositories. Overall, we collected 9,422 paper PDFs from ICLR, KDD, NeurIPs, AAAI, CVPR, CIKM, ICML, and SigGraph. From the papers collected in phase 1, we identified and cloned 3,393 GitHub repositories. Out of the 3,393 repositories, 2,792 have been updated in 2021. This indicates that a significant portion of the repositories collected

have seen recent development and are used by a significantly large community.

The most commonly used language was Python across the collected GitHub repositories. We sought to identify the top foundational repositories that the collected GitHub repositories depended on. In this research, we focus on vulnerabilities in repositories that utilize widely used libraries and frameworks. e.g., PyTorch and Transformers [39, 40]. We also include BERT due to the applicability of large language models, and Linux, since many machine learning toolboxes are developed for Linux. We focused on the top four foundational repositories to help control the scope of our study and analysis. These repositories are as follows:

- **Linux:** The Linux kernel repository was created in 2005 and had over 136k stars (a measure of popularity) on GitHub. As a highly configurable open-source operating system, a large number of out-of-the-box ML toolboxes are developed for Linux [41].
- **BERT:** BERT is a pre-trained text encoder used for natural language processing (NLP) tasks [42, 43]. Created in October 2018, it has 32.2k stars on GitHub.
- **PyTorch:** PyTorch is a foundational ML Python package that provides tensor computation with GPU acceleration and deep neural networks [44]. Created in May 2016, it has 58.9k stars on GitHub.
- **Transformers:** The transformers library provides thousands of pre-trained models to perform deep learning tasks on different modalities such as text, vision, and audio. It was created in October 2018 and has 70.4k stars [45].

From our initial collection, we sampled 207 repositories that had forked the four foundational repositories. We executed a vulnerability assessment on these repositories using an open-source static application security testing (SAST) tool, Bandit [46]. Bandit is a security scanner from OpenStack [47]. It is specifically designed to find common security issues in Python code. Bandit scans Python files for secrets, insecure coding, and attack susceptibilities [48]. Potential security problems it scans for include SQL injections, the use of unsafe libraries, the existence of unreachable code, and bad file permissions [47]. The scanner returns a severity score for each scanned vulnerability based on the Common Vulnerabilities and Enumeration (CVE) score listed in the National Vulnerability Database (NVD). The results of the vulnerability scans, broken down by severity level, have been provided in Table 1.

TABLE I. VULNERABILITY SCANNER RESULTS

Repository Type	Severity	Count
Foundational Repositories	Low	46
	Medium	16
	High	0
Dependent Repositories	Low	856
	Medium	431
	High	5
TOTAL		1354

To help protect the privacy of the scanned repositories and reduce the possibility of them getting attacked, we do not report the specific vulnerabilities associated with each of the vulnerabilities. Instead, we present a summary of the vulnerability assessment results at an aggregate level. The foundational repositories had a total number of 46 low severity vulnerabilities, followed by 16 medium vulnerabilities, and no high severity vulnerabilities. The repositories that depend on these foundational repositories had a similar distribution, wherein 856 vulnerabilities were low, 431 fell into the medium category, and five vulnerabilities were rated as high.

B. Graph Construction

The relationships between the repositories that use the foundational AI repositories, and the foundational repositories themselves, can be represented using a bipartite graph. The motivation behind representing the data as a bipartite graph is to distinguish between the foundational repositories and the repositories that build upon the foundational repositories. We determine edges as the repositories either forking the foundational repositories or explicitly declaring a dependency on a foundational repository.

We denote the bipartite graph as G= (U, V, E, F) where G represents the bipartite graph, U is the node set of all the foundational repositories, V is the node set of all the other repositories that depend on the foundational repositories, E is the edge set of all the edges between the repositories, F is the feature set that contains the vulnerability scan information. The feature sets contain the vulnerability tests done on the repositories. The feature matrix F contains all the vulnerability information for each repository in our collection and is formatted as an NxF size matrix where N is the number of nodes and F is the number of features or vulnerabilities returned by Bandit (e.g., insecure method, insecure file permission, potential secret, etc.). Each row in the feature matrix contains the number of times each vulnerability in our assessment occurs in the corresponding repository. Metrics for the constructed graph are shown in Table 2.

TABLE II. GRAPH METRICS AND STATISTICS

Graph Level Metrics	Number of Nodes	211
	Number of Edges	213
	Graph Density	0.1
	Connected Components	2
Node Level Metrics	Maximum Degree	169
	Minimum Degree	1
	Average Degree	1.98
	Maximum Betweenness	0.9
	Average Betweenness	0.08

Overall, the constructed graph has 211 nodes and 213 edges. The graph density is 0.1. This relatively low graph density indicates that the repositories have very specific and limited connections with other repositories. The maximum degree represents a repository's maximum number of forks and 169. The average degree, or the number of repositories a repository is connected to, is 1.98. This indicates that the graph follows a power-law distribution.

C. Graph Embedding

For our graph, we used the GraphSAGE method for generating nodal embeddings [34]. GraphSAGE accounts for nodal features and operates on undirected graphs. Since it is an unsupervised method, it allows us to create embeddings that will facilitate the grouping of repositories based on their relationships and vulnerabilities without a priori knowledge. Given a graph, GraphSAGE learns embeddings of the nodes using only the graph structure and the node features. As such, it is ideal for our context since there is a lack of ground-truth datasets and a need to capture the graph structure (repository relationships) and nodal attributes (vulnerabilities for each repository).

The working process of GraphSAGE is mainly divided into two steps. The first performs neighborhood sampling of an input graph. The second one is learning aggregation functions at each search depth [34]. The GraphSAGE embedding process is described in the subsequent paragraphs in a stepwise manner:

- **Context construction:** According to GraphSAGE, nodes that live in the same neighborhood should have similar embeddings. So, the algorithm has a parameter k that controls the neighborhood depth. If k is 1, only the immediately adjacent nodes are considered, if k is 2, the nodes with a distance of 2 are considered, and so on. Therefore, selecting k is very important in the GraphSAGE algorithm as increasing the k value can introduce irrelevant or undesired information sharing between nodes that can cause a reduction in quality for the embeddings generated for all the nodes [34].
- **Information aggregation:** Once the neighborhood is defined, aggregators are used to accept the information from the neighborhood as input, and then the embeddings are generated as the output. The information aggregation process works as follows: For each neighborhood until k, a neighborhood embedding is created based on the aggregator function for every node and combined with the exiting node embedding. There can be several aggregators but the pooling aggregator offers the highest efficiency and performance. In the pooling aggregator, the embedding is passed through the neural network layer to update the node embedding. Each neighbor's vector is independently fed through a fully-connected neural network; following this transformation, an elementwise max-pooling operation is applied to aggregate information across the neighbor set [34]. The equation for the aggregation function is given below (Equation 1), wherein max denotes the element-wise max operator and σ is a nonlinear activation function.

$$\text{AGGREGATE}_k^{\text{pool}} = max\left(\left\{\sigma\left(\mathbf{W}_{\text{pool}}\,\mathbf{h}_{u_i}^{k} + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\right\}\right) \quad (1)$$

- **Loss function:** Finally, to know the weights of the aggregators and the embeddings we need a differentiable loss function as we want neighboring nodes to have similar embeddings and independent nodes to have distant embeddings.

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^{\top}\mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma\left(-\mathbf{z}_u^{\top}\mathbf{z}_{v_n}\right)\right) \quad (2)$$

Equation 2 is the loss function for the GraphSAGE algorithm. In the equation node, u and node v are two neighbors. The Q in the equation represents a negative sample. A negative sample means a non-neighbor node. Q is used to set apart embeddings of the nodes u and v. Lastly, the σ denotes the sigmoid function [34].

V. Results and Evaluation

To evaluate the quality of our graph embeddings generated via GraphSAGE, we conduct a subsequent downstream task of clustering. The quality of clusters generated from a set of graph embeddings can be used to identify the utility the embeddings provide in downstream tasks. In our context, the aim of conducting clustering as our downstream task is to identify groups of ML/ DL repositories on GitHub that contain similar sets of vulnerabilities and identify repositories that are most similar to foundational ML repositories based on network characteristics and vulnerability scan information.

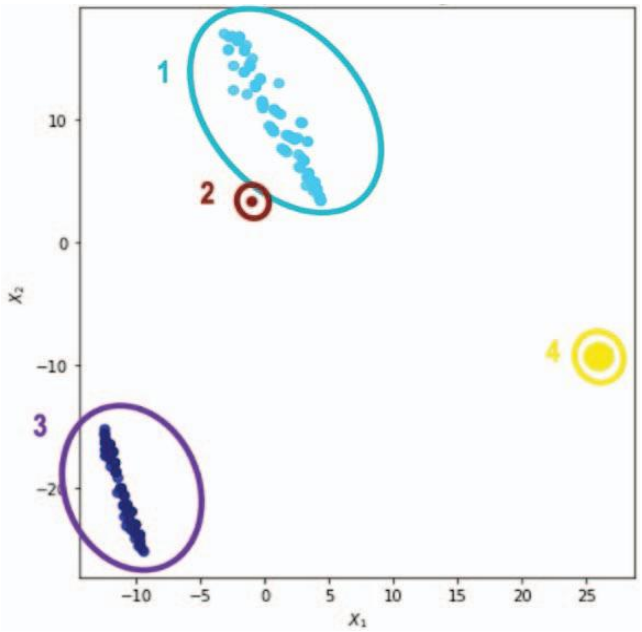
Fig. 2. Clustering Results

We use k-means clustering to identify our graph embedding clusters, and the Silhouette score and Dunn index were calculated to evaluate the cluster quality [49]. The Silhouette score and the Dunn Index are used to measure the goodness of the split between clusters and are calculated as the ratio of the smallest inter-cluster distance to the largest intra-cluster distance [49]. It can range from 0 to infinity, with a larger value indicating a better clustering split. A silhouette score ranges between (−1, +1), where a higher value indicates better matching of a member to its cluster [50]. Using the elbow method, we identified that the optimal number of clusters is four (as generated by k-means, a prevailing unsupervised clustering technique), with a Silhouette score of 0.920 and a Dunn index of 0.271.

A. Clustering Results

Our clustering results are presented in Fig. 2. Each cluster is color coded, circled, and labeled. Cluster 1 contains 28 repositories, cluster 2 contains 34 repositories, cluster 3 contains 61 repositories, and cluster 4 contains 88 repositories. In the following paragraphs, we discuss each cluster and provide the most common vulnerabilities in the cluster.

Cluster 1 contains two of the four foundational repositories in our graph, i.e., Linux and BERT. The two most prevalent vulnerabilities in Cluster 1 are *try_except_pass* and *subprocess_without_shell_equals_true*. *try_except_pass* tests for a pass in the except block, i.e. the exception is not handled



[51]. *subprocess_without_shell_equals_true* tests for the spawning of a subprocess without using a command shell. This presents a security issue if appropriate care is not taken to sanitize any user-provided or variable input. Hackers could exploit this vulnerability to insert malicious input into shell commands. Both prevalent vulnerabilities in Cluster 1 have been classified as low severity by Bandit [51]. Cluster 1 does not have any medium or high severity vulnerabilities.

Cluster 2 has a total of 34 repositories, and contains the other two foundational repositories, i.e., PyTorch and Transformers. The most prevalent vulnerability was *assert_used*, or CWE-703, when assert is used to enforce interface constraints. This is problematic due to these protections being removed when compiled to byte code [51]. The second-most prevalent vulnerability is *blacklist*, or imports that make code susceptible to injection attacks [51]. For example, being susceptible to untrusted data means hackers could execute data poisoning attacks. These vulnerabilities are respectively classified as low severity and medium severity.

Clusters 3 and 4 do not contain any foundational repositories and have a total of 61 and 88 repositories, respectively. Bandit discovered that *subprocess_without_shell_equals_true* was the most prevalent vulnerability in Cluster 3, while it was *start_process_with_partial_path* in Cluster 4.

VI. Conclusion and Future Work

In this research, we identify vulnerabilities present in foundational repositories that represent the tools used for AI development. These include specifically PyTorch, BERT, Linux, and Transformers. We analyze the foundational repositories and scientific AI papers for vulnerabilities and create a graph representation. Using an unsupervised graph embedding method, GraphSAGE, we create graph embeddings, to perform the downstream task of clustering repositories with similar vulnerabilities. We identify four different clusters of repositories and characterize the kinds of vulnerabilities present in the repositories.

We believe our findings can be used to improve the security of foundational AI development repositories and provide a targeted way to fix the vulnerabilities. As we characterize the types of vulnerabilities present in each cluster, strategies to improve them can be developed for efficiently remediation of software vulnerabilities. Such remediation strategies can play an essential role in effectively mitigating software supply chain vulnerabilities, particularly for AI. Overall, our research contributes to using AI techniques for security analysis [52-56].

Some directions for future work include using different graph embedding methods, taking into account a more significant number of nodal features in the form of the characteristics of the repositories, and undertaking more extensive data collection. We also only use a static analysis tool, Bandit. Given the rapid development of tools that can potentially dynamically scan for AI-specific vulnerabilities, we can further explore how software vulnerabilities can be linked to exploits in AI models. Ultimately, such advances can help to further shed light on how to improve the security of AI software development processes.

VII. References

[1] A. Rai, P. Constantinides, and S. Sarker, "Next-Generation Digital Platforms: Toward Human–AI Hybrids," *MIS Quarterly,* vol. 43, no. 1, pp. iii-ix, 2019/03// 2019. [Online].

[2] R. Bawack, S. F. Wamba, and K. Carillo, "Where Information Systems Research Meets Artificial Intelligence Practice: Towards the Development of an AI Capability Framework," *Technology,* vol. 12, pp. 15-2019, 2019.

[3] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the Dartmouth summer research project on artificial intelligence, august 31, 1955," *AI Magazine,* vol. 27, no. 4, pp. 12-12, 2006.

[4] K. Leavitt, K. Schabram, P. Hariharan, and C. M. Barnes, "Ghost in the Machine: On Organizational Theory in the Age of Machine Learning," *Academy of Management Review,* 2020.

[5] C. M. Bishop, "Pattern recognition," *Machine Learning,* vol. 128, no. 9, 2006.

[6] B. Allen, A. S. McGough, and M. Devlin, "Toward a Framework for Teaching Artificial Intelligence to a Higher Education Audience," *ACM Transactions on Computing Education (TOCE),* vol. 22, no. 2, pp. 1-29, 2021.

[7] C. Rosca, B. Covrig, C. Goanta, G. van Dijck, and G. Spanakis, "Return of the AI: An Analysis of Legal Research on Artificial Intelligence Using Topic Modeling," in *NLLP@ KDD*, 2020, pp. 3-10.

[8] PwC, "Sizing the prize: exploiting the AI Revolution, What's the real value of AI for your business and how can you capitalise?," *PwC's Global Artificial Intelligence Study,* 2019.

[9] S. Samtani, M. Kantarcioglu, and H. Chen, "Trailblazing the Artificial Intelligence for Cybersecurity discipline: a multi-disciplinary research roadmap," vol. 11, ed: ACM New York, NY, USA, 2020, pp. 1-19.

[10] B. Liu, M. Ding, S. Shaham, W. Rahayu, F. Farokhi, and Z. Lin, "When machine learning meets privacy: A survey and outlook," *ACM Computing Surveys (CSUR),* vol. 54, no. 2, pp. 1-36, 2021.

[11] M. Mandal and S. K. Vipparthi, "An Empirical Review of Deep Learning Frameworks for Change Detection: Model Design, Experimental Frameworks, Challenges, and Research Needs," *IEEE Transactions on Intelligent Transportation Systems,* vol. 23, no. 7, pp. 6101-6122, 2022, doi: 10.1109/TITS.2021.3077883.

[12] Q. Xiao, K. Li, D. Zhang, and W. Xu, "Security risks in deep learning implementations," in *2018 IEEE Security and privacy workshops (SPW)*, 2018: IEEE, pp. 123-128.

[13] B. Lazarine *et al.*, "Identifying vulnerable GitHub repositories and users in scientific cyberinfrastructure: An unsupervised graph embedding approach," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2020: IEEE, pp. 1-6.

[14] M. Ebrahimi, N. Zhang, J. Hu, M. T. Raza, and H. Chen, "Binary black-box evasion attacks against deep learning-based static malware detectors with adversarial byte-level language model," *arXiv preprint arXiv:2012.07994,* 2020.

[15] Y. Chai, Y. Zhou, W. Li, and Y. Jiang, "An Explainable Multi-Modal Hierarchical Attention Model for Developing Phishing Threat Intelligence," *IEEE Transactions on Dependable and Secure Computing,* vol. 19, no. 2, pp. 790-803, 2022, doi: 10.1109/TDSC.2021.3119323.

[16] S. Samtani, M. Abate, V. Benjamin, and W. Li, "Cybersecurity as an industry: A cyber threat intelligence perspective," *The Palgrave Handbook of International Cybercrime and Cyberdeviance,* pp. 135-154, 2020.

[17] D. Gonzalez, T. Zimmermann, and N. Nagappan, "The State of the ML-universe: 10 Years of Artificial Intelligence; Machine Learning Software Development on GitHub," presented at the Proceedings of the 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 2020. [Online]. Available: https://doi.org/10.1145/3379597.3387473.

[18] A. Al-Rubaye and G. Sukthankar, "Scoring Popularity in GitHub," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2020: IEEE, pp. 217-223.

[19] L. Jia, H. Zhong, X. Wang, L. Huang, and X. Lu, "The symptoms, causes, and repairs of bugs inside a deep learning library," *Journal of Systems and Software,* vol. 177, p. 110935, 2021.

[20] A. Di Franco, H. Guo, and C. Rubio-González, "A comprehensive study of real-world numerical bug characteristics," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017: IEEE, pp. 509-519.

[21] Q. Guo *et al.*, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019: IEEE, pp. 810-822.

[22] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia, "What do programmers discuss about deep learning frameworks," *Empirical Software Engineering,* vol. 25, no. 4, pp. 2694-2747, 2020.

[23] N. S. Harzevili, J. Shin, J. Wang, and S. Wang, "Characterizing and Understanding Software Security Vulnerabilities in Machine Learning Libraries," *arXiv preprint arXiv:2203.06502,* 2022.

[24] M. Jimenez, M. Papadakis, and Y. Le Traon, "An empirical analysis of vulnerabilities in OpenSSL and the Linux kernel," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016: IEEE, pp. 105-112.

[25] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical analysis of security vulnerabilities in python packages," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021: IEEE, pp. 446-457.

[26] J. Han, S. Deng, D. Lo, C. Zhi, J. Yin, and X. Xia, "An Empirical Study of the Dependency Networks of Deep Learning Libraries," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 28 Sept.-2 Oct. 2020 2020, pp. 868-878, doi: 10.1109/ICSME46990.2020.00116.

[27] B. Lazarine, Z. Zhang, A. Sachdeva, S. Samtani, and H. Zhu, "Exploring the Propagation of Vulnerabilities from GitHub Repositories Hosted by Major Technology Organizations," in *Proceedings of the 15th Workshop on Cyber Security Experimentation and Test*, 2022, pp. 145-150.

[28] Y. Fan, X. Xia, D. Lo, A. E. Hassan, and S. Li, "What makes a popular academic AI repository?," *Empirical Software Engineering,* vol. 26, no. 1, pp. 1-35, 2021.

[29] S. E. Schaeffer, "Graph clustering," *Computer science review,* vol. 1, no. 1, pp. 27-64, 2007.

[30] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering,* vol. 30, no. 9, pp. 1616-1637, 2018.

[31] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems,* vol. 151, pp. 78-94, 2018.

[32] P. Oostema and F. Franchetti, "Leveraging High Dimensional Spatial Graph Embedding as a Heuristic for Graph Algorithms," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021: IEEE, pp. 539-547.

[33] A. Modell and P. Rubin-Delanchy, "Spectral clustering under degree heterogeneity: a case for the random walk Laplacian," *arXiv preprint arXiv:2105.00987,* 2021.

[34] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems,* vol. 30, 2017.

[35] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep Graph Infomax," *ICLR (Poster),* vol. 2, no. 3, p. 4, 2019.

[36] M. Balcilar, G. Renton, P. Héroux, B. Gaüzère, S. Adam, and P. Honeine, "Spectral-designed depthwise separable graph neural networks," in *Proceedings of Thirty-seventh International Conference on Machine Learning (ICML 2020)-Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[37] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203,* 2013.

[38] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems,* vol. 32, no. 1, pp. 4-24, 2020.

[39] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771,* 2019.

[40] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 510-520.

[41] S. Sonnenburg *et al.*, "The SHOGUN machine learning toolbox," *The Journal of Machine Learning Research,* vol. 11, pp. 1799-1802, 2010.

[42] I. Tenney, D. Das, and E. Pavlick, "BERT rediscovers the classical NLP pipeline," *arXiv preprint arXiv:1905.05950,* 2019.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805,* 2018.

[44] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems,* vol. 32, 2019.

[45] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38-45.

[46] S. Peng, P. Liu, and J. Han, "A Python security analysis framework in integrity verification and vulnerability detection," *Wuhan University Journal of Natural Sciences,* vol. 24, no. 2, pp. 141-148, 2019.

[47] D. K. Konoor, R. Marathu, and P. Reddy, "Secure Openstack cloud with Bandit," in *2016 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, 2016: IEEE, pp. 178-181.

[48] B. Lazarine *et al.*, "Identifying Vulnerable Github Repositories And Users In Scientific Cyberinfrastructure: An Unsupervised Graph Embedding Approach," *2020 IEEE International Conference on Intelligence and Security Informatics (ISI),* pp. 1-6, 2020.

[49] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, 2020: IEEE, pp. 747-748.

[50] M. K. Pakhira, S. Bandyopadhyay, and U. Maulik, "Validity index for crisp and fuzzy clusters," *Pattern recognition,* vol. 37, no. 3, pp. 487-501, 2004.

[51] "Bandit," https://bandit.readthedocs.io/en/latest/.

[52] F. Lin *et al.*, "Linking Personally Identifiable Information from the Dark Web to the Surface Web: A Deep Entity Resolution Approach," in *2020 International Conference on Data Mining Workshops (ICDMW)*, 17-20 Nov. 2020 2020, pp. 488-495, doi: 10.1109/ICDMW51313.2020.00072.

[53] S. H. A. Mahmood and A. Abbasi, "Using Deep Generative Models to Boost Forecasting: A Phishing Prediction Case Study," in *2020 International Conference on Data Mining Workshops (ICDMW)*, 2020: IEEE, pp. 496-505.

[54] Y. Liu et al., "Identifying, Collecting, and Monitoring Personally Identifiable Information: From the Dark Web to the Surface Web," 2020 IEEE International Conference on Intelligence and Security Informatics (ISI), 2020, pp. 1-6, doi: 10.1109/ISI49825.2020.9280540.

[55] Zhang, N., Ebrahimi, M., Li, W., & Chen, H. (2022). Counteracting dark Web text-based CAPTCHA with generative adversarial learning for proactive cyber threat intelligence. ACM Transactions on Management Information Systems (TMIS), 13(2), 1-21.

[56] M. Ebrahimi, Y. Chai, H. H. Zhang and H. Chen, "Heterogeneous Domain Adaptation with Adversarial Neural Representation Learning: Experiments on E-Commerce and Cybersecurity," in IEEE Transactions on Pattern Analysis and Machine Intelligence, doi: 10.1109/TPAMI.2022.3163338.