Suggesting Alternatives for Potentially Insecure Artificial Intelligence Repositories: An Unsupervised Graph Embedding Approach

Ben Lazarine
Indiana University
belazar@iu.edu

Sagar Samtani
Indiana University
ssamtani@iu.edu

Hongyi Zhu
University of Texas at San Antonio
hongyi.zhu@utsa.edu

Ramesh Venkataraman
Indiana University
venkat@indiana.edu

Abstract

Emerging Artificial Intelligence (AI) applications are bringing with them both the potential for significant societal benefit and harm. Additionally, vulnerabilities within AI source code can make them susceptible to attacks ranging from stealing private data to stealing trained model parameters. Recently, with the adoption of open-source software (OSS) practices, the AI development community has introduced the potential to worsen the number of vulnerabilities present in emerging AI applications, building new applications on top of previous applications, naturally inheriting any vulnerabilities. With the AI OSS community growing rapidly to a scale that requires automated means of analysis for vulnerability management, we compare three categories of unsupervised graph embedding methods capable of generating repository embeddings that can be used to rank existing applications based on their functional similarity for AI developers. The resulting embeddings can be used to suggest alternatives to AI developers for potentially insecure AI repositories.

Keywords: Artificial Intelligence, Open-source Software, Cybersecurity, Unsupervised Graph Embedding.

1. Introduction

In recent years, the rate of artificial intelligence (AI) development has increased rapidly and has led to its capabilities surpassing human performance in tasks ranging from language understanding (Vaswani et al., 2017) to image recognition (Dosovitskiy et al., 2020). While this has enabled the emergence of powerful tools such as ChatGPT and Dall-E, it has also drawn significant concern from leaders in AI development, pointing to the potential for unknown risks and vulnerabilities of these tools to cause societal damage (Bengio et al., 2022). Examples of the potential harm AI may pose have already been shown, including an AI-enabled facial recognition system, developed by Amazon, exhibiting racial bias and labeling members of Congress as criminals and an AI chatbot, developed by Microsoft, being trained by users to be racist and sexist within 24 hours of release. However, in addition to these examples of AI unintentionally going wrong, there are vulnerabilities in AI source code that may soon become new targets for intentional cyber-attacks. These attacks will range from adversaries exfiltrating private information used to train models to adversaries sending a flood of inputs engineered to require significant computational power to take down AI systems (MITRE, 2022).

The accelerated rate of development that AI has seen has largely been spurred by the adoption of open-source software (OSS) resources and practices by the AI development community. While accelerating the rate of development, this adoption has also brought the potential to increase the number of vulnerabilities present in emerging AI applications. This is an issue that is of particular concern to AI developers within the larger OSS development community, as currently much of AI development is done iteratively upon previously developed applications and open-source code (Langenkamp and Yue, 2022; Jiang et al., 2023a). Additionally, the nature of AI development lends itself to such iterative development more so than past open-source applications, particularly in areas such as natural language processing and computer vision, as progress in these application areas has been largely incremental by nature. GitHub is a popular software development platform that supports a significant amount

URI: https://hdl.handle.net/10125/106441
978-0-9981331-7-1

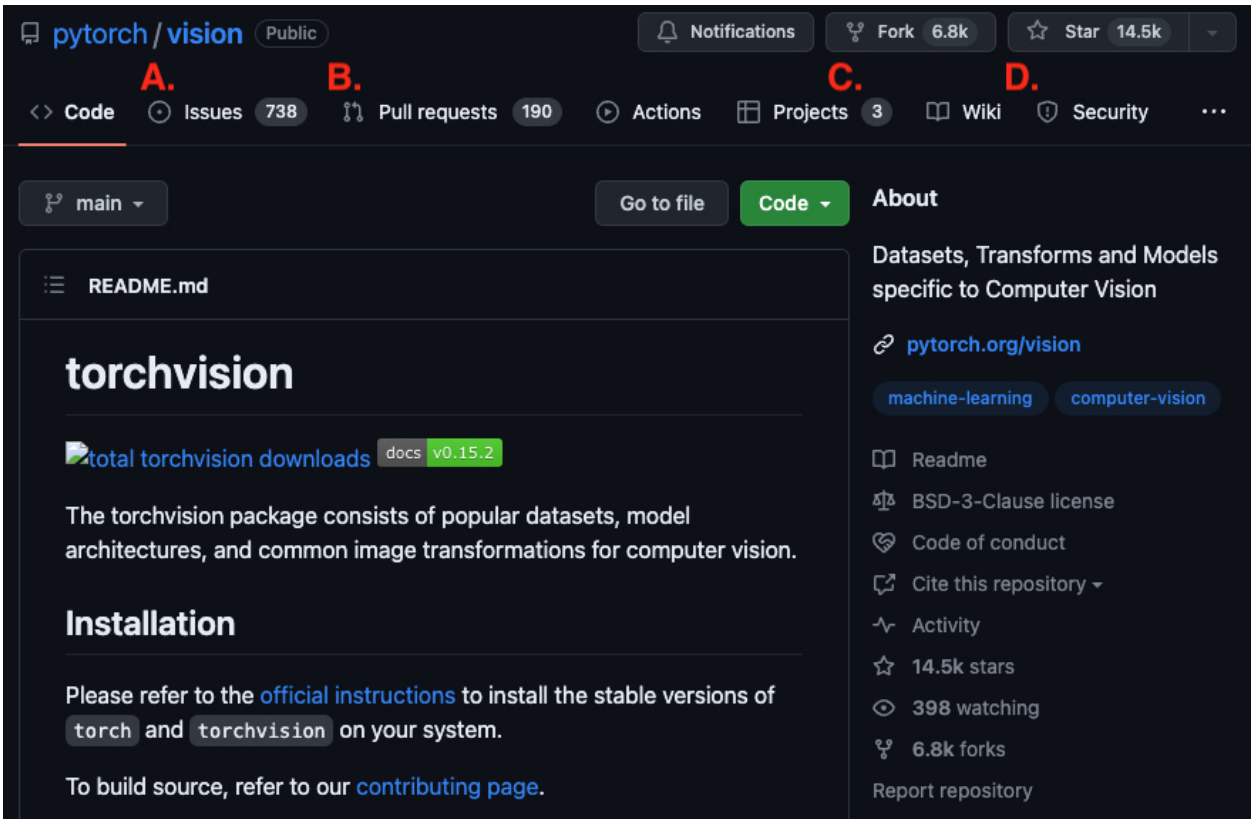

Figure 1. GitHub Mechanisms to Facilitate OSS Reuse. Each repository typically includes: (A) Open Issues (B) Pull Requests (C) Forking and (D) Starring/Watching.

of OSS development, seeing over 400 million OSS contributions in 2022 (GitHub, 2022). To exemplify how OSS resources enable iterative AI development, Figure 1 illustrates mechanisms offered by GitHub to facilitate code reuse.

To facilitate the reuse of OSS hosted on their platform, GitHub offers mechanisms for users to open issues and note areas for improvement, submit pull requests to contribute code or documentation, watch repositories that may be of interest for possible revisiting, and fork (i.e., copy) OSS repositories for their own development purposes. Developers may take on any or multiple of these user types within the AI development community. Regardless of what user type a developer may be taking on, each type signals that the user is in the process of or has completed searching for a repository to reuse. AI developers going through this process will aim to identify existing repositories that most closely align with the functionality they are searching for but also seek repositories with a low number of vulnerabilities. If an AI developer has found a repository with too many functionality or security issues, they may also search for alternatives. To demonstrate the process an AI developer may follow when selecting a repository to enhance, replicate, reuse, or adapt in their own development, we next detail three steps a typical AI reengineering workflow consists of.

- **Step 1:** A developer will survey existing repositories on platforms such as GitHub to identify potential candidates for reuse. A key limitation of this step is that it can be difficult and time-consuming to identify all relevant candidates through a manual survey consisting of methods such as keyword searches.
- **Step 2:** A developer will compare candidate repositories' key characteristics to assess which best aligns with their desired application. In this step, the developer will likely largely focus on assessing repository functionality while potentially assessing supplementary characteristics such as recent development activity, support, and possible security issues. AI-specific functionality characteristics that are often considered include types of data an application can operate on (e.g., image, text, or graph data), whether training data is already

provided, the applications underlying architecture (e.g., PyTorch or Tensorflow), and downstream tasks that the application may be suited for (e.g., image recognition, text generation, etc.). However, a key limitation of this comparison is that vulnerabilities within the repositories may not be considered, and even if the developer would like to, they may be prohibitively difficult to identify.

- **Step 3:** A developer will select one or more repositories for enhancement, replication, reuse, or adaptation. The key limitation of this step is that due to the difficulty of manually assessing repository characteristics, the best repository may not be selected, potentially leading to increased development time or selecting a repository with vulnerabilities to build a new application on top of, continuing the cycle of OSS development practices increasing vulnerability presence in emerging AI applications.

In this paper, we compare unsupervised graph embedding methods capable of representing repository relationships and characteristics as easily comparable embeddings to provide AI developers with a list of potential applications they can refer to when beginning their development. Our study offers a significant practical contribution to the AI development community by proposing an automated framework to help future AI developers better identify and select low-vulnerability repositories capable of performing their desired task in any AI reengineering process, enhancing their ability to select appropriate repositories for reuse, making the process faster, more accurate, and less likely to contain vulnerabilities.

The remainder of this paper is organized as follows. First, we review recent machine learning open-source software (MLOSS) and Unsupervised Graph Embedding literature. Second, we identify research gaps in the reviewed literature and pose our study's research questions. Third, we present our research design and explain each component in detail. Fourth, we present and discuss the results of our evaluation procedure. Fifth, we illustrate the potential value of the proposed method through a case study, and lastly, we conclude this research and summarize promising directions for future studies.

2. Literature Review

We review two key areas of literature to guide this research. First, we review MLOSS research to identify studies that have examined open-source and security practices of the MLOSS community. Second, we review unsupervised graph embedding methods to review approaches to modeling the relationship between machine learning applications and developers.

2.1. Machine Learning Open-Source Software Research

Recent MLOSS studies have focused on either pre-trained models or ML source code repositories. Additionally, MLOSS studies thus far have primarily conducted descriptive analysis or manual assessments. Descriptive studies have worked to cluster machine learning repositories with similar vulnerabilities (Sachdeva et al., 2022) and develop taxonomies of ML frameworks and libraries (Nguyen et al., 2019). Further studies have conducted manual model or repository assessments to identify attributes considered in pre-trained model reuse (Jiang et al., 2023b), identify bugs in reused pre-trained models (Pan et al., 2022), develop threat models (Jiang et al., 2022), quantify MLOSS GitHub contributions (Langenkamp and Yue, 2022), and identify code smells in ML repositories (Van Oort et al., 2021).

The primary methodologies used in past studies have been interviews (Jiang et al., 2023b), taxonomy development (Nguyen et al., 2019; Pan et al., 2022), and manual model/source code examination (Jiang et al., 2022; Langenkamp and Yue, 2022; Van Oort et al., 2021), with limited use of automated methods (Sachdeva et al., 2022). This has significantly restricted the number of models and repositories extant studies have examined, with the majority ranging between 10 and 100. Additionally, while studies have conducted vulnerability scans (Sachdeva et al., 2022; Van Oort et al., 2021), identified studies have yet to extend their analysis to ranking repositories most suitable for reuse from a functionality or security perspective. The lack of automated methods to examine repositories, as well as the lack of literature examining how to compare repositories from a functionality and security perspective, within extant MLOSS research leaves a significant need for the development of tools that can automatically provide AI developers with the ability to quickly and accurately assess repositories being considered for reengineering by the functionality they provide and the number of security issues they contain.

2.2. Unsupervised Graph Embedding

Identifying repositories that are close in functionality and do not have many vulnerabilities requires representing them in vector space to enable ML-based methods such as ranking (Goyal and Ferrara,

2018). GitHub follows a bipartite network structure, with relationships between repositories and developers (Lazarine et al., 2020). Past literature representing repositories and developers as a bipartite network has generated monopartite network projections of connected repositories and developers (Barabasi, 2013). Unsupervised graph embedding methods can operate on such a network, enabling practical analysis of a network of repositories by representing each repository based on its characteristics and network information. Traditional methods identified in our review can be grouped into three categories based on how they operate:

- **Autoencoder-Based Methods** such as Graph Convolution Autoencoder (GCAE) and Variational Graph Autoencoder (VGAE) (Kipf and Welling, 2016a) combine graph convolutional networks (GCNs) and autoencoders; using a GCN-based encoder to capture structural and feature information of a node and its neighbors via neighborhood aggregation, and a decoder to update nodal embeddings for accurate network reconstruction.
- **GCN-Based Deep Representation Methods** such as Graph Convolutional Networks (GCN) (Kipf and Welling, 2016b), Simplified Graph Convolutions (SGC) (Wu et al., 2019), and Graph Attention Networks (GAT) (Velickovic et al., 2017) use graph convolutional networks to capture nodal structure and feature information via message passing in which nodes exchange information with their neighbors to update their embeddings.
- **Matrix Factorization-Based Deep Representation Methods** such as Text-Associated Deepwalk (TADW) (Yang et al., 2015) and Text-Enhanced Network Embedding (TENE) (Yang and Yang, 2018) learn nodal embeddings by decomposing the adjacency and feature matrices into lower dimensional matrices that represent nodes and their embeddings.

In general, unsupervised graph embedding methods can capture repository characteristics and network information in an embedding, encoding local neighborhood and feature information for each repository into embeddings suitable for downstream tasks such as similarity ranking.

2.3. Research Gaps and Questions

We identified two key research gaps from our literature review. First, recent MLOSS research has primarily consisted of manual and descriptive investigations of machine learning repositories. As a result, MLOSS research has yet to develop automated solutions to reduce vulnerabilities in emerging repositories. Second, extant research has not leveraged unsupervised graph embedding techniques to generate repository representations that can be used to identify repository alternatives based on their functionality characteristics; however, such deep learning methods are a promising solution for this task due to their ability to fully leverage relationship and characteristic information of repositories. Based on these research gaps, we propose the following questions:

- How can we address key limitations AI developers face when going through the process of searching, comparing, and selecting repositories for reengineering?
- How can unsupervised graph embedding methods represent repositories with similar functionality to facilitate repository alternative ranking?

3. Research Design

We present our proposed research framework in Figure 3. The framework has four major components: Research Testbed, Graph Formulation, Graph Embedding, and Experiments and Evaluation. We describe each in the following sub-sections.

3.1. Research Testbed: MLOSS Repositories on GitHub and Vulnerability Assessment

The first component of our research testbed is a collection of 990 MLOSS repositories on GitHub. To identify a related community of MLOSS repositories in which multiple options will be present for various tasks, we used the GitHub Search API with the term ”Hugging Face.” Hugging Face is a platform that supports AI development, providing foundational repositories and computational resources needed to develop new AI models and host them in an easy-to-use environment. To leverage issues, pull requests, watches, and forks as a mechanism for identifying related repositories, we identified 27,167 issues, 3,071 pull requests, 27,786 watches, and 4,620 forks across our collection of 990 repositories. This number of issues, pull requests, watches, and forks indicates that we successfully identified an ML community with a high level of development activity and interest.

The second component of our research testbed is a vulnerability assessment conducted with three prevailing open-source static application security testing (SAST) tools popular within the information security

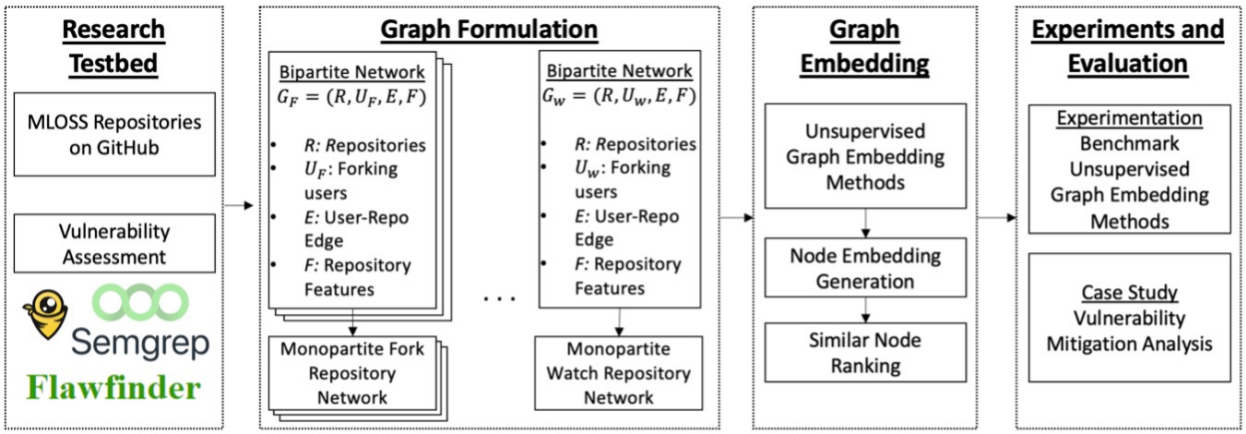

Figure 2. Proposed Research Framework

community: Semgrep (Kluban et al., 2022), Bandit (Torkura and Meinel, 2016), and Flawfinder (Kaur and Nayyar, 2020). We selected these tools based on their usability and popularity to maximize coverage of vulnerability types and programming languages. Semgrep leverages community-published rulesets to scan all types of files for 15 vulnerability types (potential passwords/keys, weak cryptography, dangerous file permissions, dangerous handling of user input, etc.) across four categories (secrets, insecure permissions and functions, web attack susceptibilities, and AI-specific vulnerabilities). Semgrep is the only SAST tool capable of scanning for AI-specific vulnerabilities that we have identified. Bandit scans Python files for 11 vulnerability types across three categories (secrets, insecure permissions and functions, and web attack susceptibilities), and Flawfinder scans C/C++ files for four vulnerability types across the same three categories. All three scanners provide severity and confidence scores for identified vulnerabilities. A summary of our vulnerability assessment results is shown in Table 1.

Table 1. Summary of GitHub Vulnerability Scan Results

		Severity			Total	Affected Repositories
		High	Medium	Low		
Vulnerability Category	Secret	47	124	1,488	1,659	164
	Insecure	802	4,268	57,483	62,553	450
	Attack	28	1,131	3,286	4,445	175
	AI-Specific	3,113	81	3,645	6,839	235
Total		3,990	5,604	65,902	75,496	537

Overall, our vulnerability assessment returned 75,496 vulnerabilities across 537 repositories in our GitHub collection, with 3,990 high severity, 10,141 medium severity, and 65,902 low severity vulnerabilities. Our assessment returned 1,659 secrets, 62,553 insecure, 4,445 attack, and 6,839 AI-specific vulnerabilities by category. We observe that the identified vulnerabilities follow a typical severity distribution, with most of the vulnerabilities being low severity. Additionally, the majority of vulnerabilities identified were insecure permissions and functions. Lastly, each vulnerability category was observed to affect at least 175 repositories, with insecure permissions and functions and AI-specific vulnerabilities affecting the most at 450 and 235, respectively.

3.2. Graph Formulation

As identified previously, the relationship between repositories and developers on GitHub can be represented as a bipartite projection that can subsequently be projected into monopartite networks for repositories and users (Lazarine et al., 2020). Therefore, we denote four bipartite networks as $G_T = R,\ D_T,\ E,\ F; T \in (f, i, w,\ p)$, where G is an undirected bipartite graph, R is the node set, r1, r2, r3, … rn, of all repositories. D is the node set, d1, d2, d3, … dn, of all developers with a T relationship to a repository, E is the edge set, e1, e2, e3, … en, of edges between repositories and developers, F is the feature matrix of each repository node (repository functions and number of vulnerabilities), and $T \in (f, i, w,\ p)$ is the set of developer types, denoting developers that have forked, opened an issue, watched, or submitted a pull request to a repository, respectively. We project our four bipartite networks into four monopartite networks of related repositories defined as $G_T = R_T,\ E,\ F; T \in (f, i, w,\ p)$, where G is an undirected graph, R is the node set, r1, r2, r3, … rn, of all repositories with a T relationship, E is the edge set, e1, e2, e3, … en, of edges

between repositories, F is the feature matrix of each repository node (repository functions and number of vulnerabilities), and $T \in (f, i, w, p)$ is the set of repository relationships, denoting repositories that share common forking, issue opening, watching, or pull requesting developers, respectively.

3.3. Graph Embedding

Successful ranking of repositories by functionality and vulnerability level in a network requires an unsupervised graph embedding method that can consider all repository alternatives and account for nodal features. Therefore, we select methods from three categories of unsupervised graph representation learning to identify which method is best able to capture repository network and feature information. The graph embedding process is as follows:

- **Step 1 (Graph Representation):** The input network of repositories is represented as an adjacency matrix representing linked repositories and a feature matrix representing the functionality and vulnerability information of each repository. Each repository's functionality information is encoded from its name, description, and readme file as a vector leveraging Doc2Vec (Rokon et al., 2021; Le and Mikolov, 2014) and its vulnerability information is represented as the number of vulnerabilities within the repository for each type of vulnerability identified by Semgrep, Bandit and Flawfinder (Lazarine et al., 2020).
- **Step 2 (Embedding Generation):** Each unsupervised graph embedding method performs its primary operation (autoencoding, message passing, or matrix factorization) generating and updating embeddings for each repository, capturing structural and feature information.
- **Step 3 (Repository Representations):** Repository representations are obtained for each repository from the final layer of the graph embedding method.

This procedure results in repository embeddings that can be compared by cosine similarity to identify a ranked list of potential repository alternatives by functionality.

3.4. Evaluation Procedure

In our experimentation, we compare 11 baseline deep learning methods to evaluate three major categories of baseline embedding methods:

- **Autoencoder-Based Deep Representation Learning,** including Graph Convolutional Autoencoder (GCAE) (Kipf and Welling, 2016a), Graph Attention Autoencoder (GATE) (Salehi and Davulcu, 2019), and Variational Graph Autoencoder (VGAE) (Kipf and Welling, 2016a).
- **GCN-Based Deep Representation Learning,** including Graph Convolutional Network (GCN) (Kipf and Welling, 2016b), Simplified Graph Convolutions (SGC) (Wu et al., 20129), Graph Attention Network (GAT) (Velickovic et al., 2017), and Approximate Personalized Propagation of Neural Predictions (APPNP) (Gasteiger et al., 2018).
- **Matrix Factorization-Based Deep Representation Learning,** including Text-Associated Deep Walk (TADW) (Yang et al., 2015), Text-Enhanced Network Embedding (TENE) (Yang and Yang, 2018), Binarized Attributed Network Embedding (BANE) (Yang et al., 2018), and Fusing Structure and Content via Non-negative Matrix Factorization (FSCNMF) (Bandyopadhyay et al., 2018).

In Experiment 1, repository embeddings are generated by each benchmark method, and the embedding performances are evaluated with Mean Average Precision (MAP). MAP evaluates the quality of embeddings produced by an unsupervised graph embedding method by attempting to reconstruct the network from the embeddings and comparing the reconstructed network to the original network, calculating the average precision of each node (Ciu et al., 2018). This evaluates how well the embeddings produced by the graph embedding method encoded the network's structure.

We performed Experiment 1 on all four identified repository networks and across two feature set combinations (repository functionality features alone and repository functionality and vulnerability features) to observe model performances across various datasets and evaluate each model's ability to encode functionality and vulnerability features together.

4. Results and Discussion

We present the results for Experiment 1 are presented in Table 2. The best performances are highlighted in boldface.

In Experiment 1, GCAE performed with a MAP of 0.347, 0.336, 0.452, and 0.487 for each network with the repository functionality feature set. This outperforms all other methods in three of the four repository networks,

Table 2. Experiment 1 Results

Category	Model	Pull Request		Issue		Fork		Watch	
		Function	Function+ Vulnerability	Function	Function+ Vulnerability	Function	Function+ Vulnerability	Function	Function+ Vulnerability
Autoencoder	GCAE	**0.347**	**0.382**	0.336	0.220	**0.452**	0.263	**0.487**	0.309
	GATE	0.331	0.300	0.327	**0.341**	0.411	**0.367**	0.210	0.217
	VGAE	0.293	0.360	**0.337**	0.215	0.392	0.363	0.274	**0.335**
GCN	GCN	0.097	0.027	0.127	0.063	0.205	0.085	0.153	0.161
	GAT	0.066	0.030	0.049	0.020	0.036	0.195	0.151	0.008
	SGC	0.060	0.034	0.107	0.071	0.202	0.070	0.000	0.165
	APPNP	0.064	0.080	0.109	0.093	0.229	0.183	0.179	0.165
Matrix Factorization	TADW	0.075	0.088	0.092	0.094	0.261	0.204	0.198	0.183
	TENE	0.089	0.083	0.123	0.110	0.252	0.223	0.177	0.168
	BANE	0.102	0.108	0.091	0.137	0.210	0.236	0.172	0.184
	FSCNMF	0.082	0.080	0.104	0.112	0.221	0.219	0.163	0.164

with GATE performing second best in the pull request repository network (0.331) and the fork repository network (0.411) and VGAE performing best in the issue repository network (0.337) and second best in the watch repository network (0.274). Interestingly, we see that while GCAE is the best-performing method overall, its performance drops significantly for the issue, fork, and watch networks, with drops of 11.6%, 18.9%, and 19.8%, respectively, when vulnerability features are included. This drop in performance is seen across many of the methods, indicating that these benchmark methods struggle to encode both functionality and vulnerability information of a repository while retaining information about the network structure. Additionally, the autoencoder-based methods performed the best across all four networks, with the matrix factorization methods performing second best, and the GCN-based methods performing the worst, indicating that the autoencoder-based methods can best capture both the local repository relationships and repository features.

5. Case Study

To demonstrate the practical value of our proposed research framework for AI developers, we conduct a case study illustrating how it addresses the key limitations of the current AI reengineering process that AI developers go through. The aim of our case study is to identify potential alternative repositories for a key vulnerable repository identified from our GitHub collection. We conduct our case study on the 990 Hugging Face related repositories collected as our research testbed. The overall process for conducting our case study is comprised of four steps:

- **Step 1 (GitHub Collection):** In this step, we identify and collect 990 related AI repositories on GitHub, using the key words of ”Hugging Face”.
- **Step 2 (Vulnerability Scanning):** In this step, we scan every repository collected in Step 1 for vulnerabilities using the aforementioned vulnerability scanners.
- **Step 3 (Embedding Generation):** Here, we leverage the best-performing graph embedding method from Experiment 1 (GCAE) to generate embeddings for every repository, embedding each repository’s network and feature information.
- **Step 4 (Alternative Identification):** For a selected key vulnerable repository, we perform cosine similarity analysis to identify repositories with similar embeddings, signaling similar functionality and analyze their potential effectiveness as alternatives for reengineering.

The repository we select for Step 4 in our case study is HugNLP. HugNLP is a popular NLP library based on Hugging Face Transformers with 218 stars and 9 forks. However, this repository contains 248 vulnerabilities, 102 of which are AI-specific vulnerabilities including potential code injection and insecure deserialization. Using the embeddings generated by GCAE in Step 3, and comparing them for similarity to HugNLP’s embedding, we can identify three NLP repositories with significantly fewer vulnerabilities that AI developers may consider as alternatives to HugNLP:

- **KNN-Transformers:** This repository trains sequence-to-sequence language models, and our vulnerability assessment only identified 9 vulnerabilities within it, the majority of which being low severity. This would be a promising alternative for any AI developer looking to train or develop a language model capable of performing sequence-to-sequence tasks.
- **T2t-tuner:** This repository can be used to train text-to-text language models with Transformers, and our vulnerability assessment only identified one low-severity vulnerability within it. This

repository would be a promising alternative for text generation-based AI tasks. Additionally, identifying this repository highlights the potential of our proposed framework to address the first key limitation of the AI reengineering workflow, alternative identification. T2t-tuner only has 18 stars and three forks, indicating that developers may find it difficult to find via simple repository search.

- **T5-flax-gcp:** This repository provides developers with a tutorial to pre-train and fine-tune an NLP model for sequence-to-sequence tasks. While this repository uses JAX as its AI library, it may still be a promising alternative to HugNLP as our vulnerability assessment found it to have only two vulnerabilities.

6. Conclusion and Future Work

We focus the context of our study on AI repositories and development due to AI's continuing increase in its potential impacts, aiming to address the problem of AI reengineering leading to increased vulnerabilities in emerging AI applications. Identifying that extant research has yet to leverage graph embedding methods as a technique to compare repositories within an AI development community, we compare 11 graph embedding methods to generate embeddings of repositories for downstream ranking. We evaluate each method on four repository networks constructed from a collection of AI repositories related to Hugging Face. Our results demonstrate that GCAE is a promising method to capture the functionality and vulnerability information of repositories while retaining key network information. Our study's practical contribution is to AI developers, making finding all potential AI task alternatives, considering functionality and security characteristics of given alternatives, and selecting an alternative for reuse faster and more likely to result in the best alternative being selected.

As with any research, our study has limitations. First, we analyze each repository network individually, which may lead to incomplete repository relationships, potentially failing to identify the best alternative recommendations. This limitation may be overcome by designing a framework that examines all four repository networks together when generating each repository embedding. Such a framework would better leverage the full crowd wisdom available on GitHub to identify repository alternatives. Second, our research is limited by using static application security testing tools to conduct our vulnerability assessment. Dynamic scanning tools may identify a more expansive set of vulnerabilities. Finally, our framework does not account for the evolution of a repository. Considering temporal dynamics may help pinpoint alternative repositories' viability at the appropriate time points.

There are several promising directions for future research. Our study primarily focuses on suggesting less vulnerable AI repositories as alternatives for an AI developer to consider; however, AI developers may have specific needs that do not allow for using alternatives. Here, AI developers may benefit from a tool that automatically maps vulnerabilities within a repository they are using to a threat model and remediation strategies such that they can understand whether the vulnerabilities have a potential for impact in their reuse and how to address them if they do. Further research may extend this work to map all emerging AI repositories to use cases and threat models to understand whether vulnerability remediation is important for a given repository. Conducting a dynamic vulnerability assessment of AI applications in a deployed environment can significantly elevate our understanding of their vulnerabilities and their implications.

7. References

Bandyopadhyay, S., Kara, H., Kannan, A. and Murty, M.N., 2018. Fscnmf: Fusing structure and content via non-negative matrix factorization for embedding information networks. arXiv preprint arXiv:1804.05313.

Barabási, A.L., 2013. Network science. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 371(1987), p.20120375.

Bengio Y., Russell S., and Selman, B., 2023. Pause Giant AI Experiments: An Open Letter. FutureofLife.org.

Ciu, P., Wang, X., Pei, J., and Zhu, W., "A Survey on Network Embedding," IEEE TKDE, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. and Uszkoreit, J., 2020. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

Gasteiger, J., Bojchevski, A. and Günnemann, S., 2018, Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In International Conference on Learning Representations.

Goyal, P., and Ferrara, E., "Graph Embedding Techniques, Applications, and Performance: A Survey." Knowledge-Based Systems, 151, pages 78–94, 2018.

https://atlas.mitre.org
https://octoverse.github.com
Jiang, W., Banna, V., Vivek, N., Goel, A., Synovic, N., Thiruvathukal, G.K. and Davis, J.C., 2023. Challenges and Practices of Deep Learning Model Reengineering: A Case Study on Computer Vision. arXiv preprint arXiv:2303.07476.
Jiang, W., Synovic, N., Hyatt, M., Schorlemmer, T.R., Sethi, R., Lu, Y.H., Thiruvathukal, G.K. and Davis, J.C., 2023. An empirical study of pre-trained model reuse in the hugging face deep learning model registry. arXiv preprint arXiv:2303.02552.
Jiang, W., Synovic, N., Sethi, R., Indarapu, A., Hyatt, M., Schorlemmer, T.R., Thiruvathukal, G.K. and Davis, J.C., 2022, November. An Empirical Study of Artifacts and Security Risks in the Pre-trained Model Supply Chain. In Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (pp. 105-114).
Kaur, A., and Nayyar, R., "A Comparative Study of Static Code Analysis Tools for Vulnerability Detection in C/C++ and JAVA Source Code," Procedia Computer Science, 171, pages 2023–2029, 2020.
Kipf, T.N. and Welling, M., 2016. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308.
Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. In The International Conference on Learning Representations.
Kluban, M., Mannan, M. and Youssef, A., 2022, May. On measuring vulnerable JavaScript functions in the wild. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (pp. 917-930).
Langenkamp, M. and Yue, D.N., 2022, July. How Open Source Machine Learning Software Shapes AI. In Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society (pp. 385-395).
Lazarine, B., Samtani, S., Patton, M., Zhu, H., Ullman, S., Ampel, B. and Chen, H., 2020, November. 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) (pp. 1-6). IEEE.
Le, Q. and Mikolov, T., 2014, June. Distributed representations of sentences and documents. In International conference on machine learning (pp. 1188-1196). PMLR.
Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., López García, Á., Heredia, I., Malík, P. and Hluchý, L., 2019. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. Artificial Intelligence Review, 52, pp.77-124.
Pan, R., Biswas, S., Chakraborty, M., Cruz, B.D. and Rajan, H., 2022. An Empirical Study on the Bugs Found while Reusing Pre-trained Natural Language Processing Models. arXiv preprint arXiv:2212.00105.
Rokon, M.O.F., Yan, P., Islam, R. and Faloutsos, M., 2021, September. Repo2vec: A comprehensive embedding approach for determining repository similarity. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 355-365). IEEE.
Sachdeva, A., Lazarine, B., Dama, R., Samtani, S. and Zhu, H., 2022, November. Identifying Patterns of Vulnerability Incidence in Foundational Machine Learning Repositories on GitHub: An Unsupervised Graph Embedding Approach. In 2022 IEEE International Conference on Data Mining Workshops (ICDMW) (pp. 1-8). IEEE.
Salehi, A. and Davulcu, H., 2019. Graph attention auto-encoders. arXiv preprint arXiv:1905.10715.
Torkura, K. A., and Meinel, C., "Towards Vulnerability Assessment as a Service in OpenStack Clouds," Proceedings - Conference on Local Computer Networks, LCN, pages 1–8, 2016.
Van Oort, B., Cruz, L., Aniche, M. and Van Deursen, A., 2021, May. The prevalence of code smells in machine learning projects. In 2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN) (pp. 1-8). IEEE.
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems, 30.
Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y., 2017, Graph Attention Networks. In International Conference on Learning Representations.
Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T. and Weinberger, K., 2019, May. Simplifying graph convolutional networks. In International conference on machine learning (pp. 6861-6871). PMLR.
Yang, C., Liu, Z., Zhao, D., Sun M., and Chang, E., "Network Representation Learning with Rich Text Information," In Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015.
Yang, H., Pan, S., Zhang, P., Chen, L., Lian, D. and Zhang, C., 2018, November. Binarized attributed network embedding. In 2018 IEEE International Conference on Data Mining (ICDM) (pp. 1476-1481). IEEE.
Yang, S. and Yang, B., 2018, August. Enhanced network embedding with text information. In 2018 24th International Conference on Pattern Recognition (ICPR) (pp. 326-331). IEEE.