Golden Dataset for Software Supply Chain Security

Bailey Nathan
School of Computer Science
The University of Adelaide
a1870819@adelaide.edu.au

Nguyen Khoi Tran
School of Computer Science
The University of Adelaide
nguyen.tran@adelaide.edu.au

M. Ali Babar
School of Computer Science
The University of Adelaide
ali.babar@adelaide.edu.au

ABSTRACT

This project aims to develop a dataset of Software Supply Chain (SSC) metadata for the Generative Artificial Intelligence (GenAI) open-source software ecosystem, focusing on text-to-text GenAI. SSC metadata denotes the standardized, machine-understandable, and verifiable information regarding software artefacts, describing where, when and how they were produced. The dataset produced in this project would contain Software Bill of Materials (SBOM) and provenance statements of prominent open-source software tools, utilities, and frameworks making up the GenAI ecosystem. Such dataset would facilitate further investigation into the state of SSC security of GenAI. This dataset would also contribute to the on-going research in SSC security domain regarding the conceptualisation and measurement of SSC metadata quality.

CCS CONCEPTS

• **Security and privacy** → *Software and application security*;

KEYWORDS

Software Supply Chain, SBOM, Generative AI

ACM Reference Format:
Bailey Nathan, Nguyen Khoi Tran, and M. Ali Babar. 2024. Golden Dataset for Software Supply Chain Security. In *Proceedings of* . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION AND MOTIVATION

In recent times, developing software applications involves using open-source components and libraries; these open-source projects have incorporated dependencies from other open-source projects, creating a Software Supply Chain (SSC). [7, 9]. Generative Artificial Intelligence (AI) describes computational techniques for generating content such as text and images [3]. Generative AI has seen exponential growth in recent years, with the latest annual McKinsey Global Survey finding that sixty per cent of organisations with reported AI adoption are using generative AI [2]. However, generative AI is considered a part of an SSC and, similar to other software artefacts, SSCs must be secure from cyber attacks. An SSC attack exists when malicious code is injected into software to compromise dependent systems further down the supply chain [6]. Sonatype's 9th Annual State of the Software Supply Chain Report outlined that 2023 included twice as many SSC attacks as all the attacks combined from 2019 to 2022 [10]. SSC security is critical to ensure that the software developed does not contain vulnerabilities to help reduce the number of SSC attacks. SSC metadata is significant for SSC security, as it provides visibility into the SSC of a software artefact. SSC metadata can include a Software Bill of Material (SBOM), a list of ingredients that make up software components [1]. SSC metadata may also include provenance statements that describe when, where and who developed a software artifact to help provide context and traceability [4].

Unfortunately, there is a lack of real-world SSC metadata dataset for open source software ecosystem in general [12] and the GenAI ecosystem in particular. Such a dataset would be beneficial from both software development and research perspectives. It would enable developers to assess the quality of software artefacts they import into the supply chain of their own software solutions. For researchers, such a dataset could provide an empirical foundation to develop models and mechanisms to assess the quality of SSC metadata.

This research aims to generate a dataset containing SSC metadata for the generative AI open-source ecosystem. To achieve this goal, the project will be broken down into the following research questions

- **RQ1: What is the reference architecture of a Gen AI stack?** Such a reference architecture would identify and describe primary types of software components that form a GenAI solution. It lays a foundation for identifying and retrieving prominent repositories in GenAI ecosystem.
- **RQ2: What are major open source software repositories forming the GenAI ecosystem?** Finding such repositories would allow us to collect existing source code, which can lead into further analysis and the creation of SSC metadata.
- **RQ3: Which component within the generative AI ecosystem relies on the most number of dependencies?** Such a question allows us to analyse the results found from RQ2 and identify a list of dependencies for each component within the generative AI ecosystem.

The remains of this paper is structured as follows. Section 2 presents an overview of the existing literature regarding SSC security, SSC metadata and the quality of SBOMs. As well as providing insight into existing strategies and frameworks created to help increase the SSC security. Section 3 defines the methodology of our research. Section 4 outlines the experimental setup of the project that was required to generate the SBOMs. Section 5 describes the results found for each of the research questions. Finally, section 6 highlights the findings from this research and discusses further research that may be conducted.

2 LITERATURE REVIEW

The related work from these literature reviews covers Software Supply Chain (SSC) attacks and the importance of SSC metadata in protecting against such attacks.

In Marc Ohm's Open Source Software Supply Chain Attacks paper [6], he characterises software supply chain attacks as the injection of malicious code into a software package to compromise dependent systems. An attacker chooses a package to infect, and the malicious code may be injected into the sources, building the build or the package repository. Considering open source projects rely on community contributions, attackers take advantage of this and mimic project contributions where they may commit malicious code into the project's code. Ohm's research presents a dataset and a manual analysis of 174 malicious software packages used in real-world open-source SSC attacks. Such a dataset allows for supervised learning approaches to search for malicious packages. One limitation of the proposed dataset is that it can not obtain malicious packages for Java and PHP. Furthermore, thirty-four per cent of malicious packages are droppers with the goal of downloading a second-stage payload, and these may not be available. Overall, this academic literature uses an approach that utilises software packages distributed via package repositories such as npm and PyPi. The proposed dataset is beneficial for generic open-source software supply chain attacks. However, it does not provide a dataset specific to open-source generative AI attacks.

Nguyen Khoi Tran applies generated SSC metadata through a container-centric approach [12]. SSC metadata is a useful document for consumers to protect against SSC attacks, as it provides machine-readable and authenticated documents that describe an artifact's life cycle. Embedding the SSC metadata documents into container images provides benefits as it simplifies the resolution step because consumers no longer need to link SSC metadata with software artefacts.

Software Bill of Materials (SBOM) are often included in SSC metadata to help identify components with known vulnerabilities. This academic literature by Torres-Arias et al. [11] highlights the current challenge with SBOMs, which revolves around the quality of SBOMs being produced. It was found that only one per cent of generated SBOMs contained the minimum elements outlined by The U.S. National Telecommunications and Information Administration (NTIA) report. Currently, SBOM quality varies a lot and simply viewing an SBOM is not enough to determine the SBOM's quality. Software researchers currently have no standard SBOM datasets. Furthermore, the academic literature concludes that it is hard to research SBOMs since so few are made publicly available and suitable for research, thus highlighting the need for further research and more datasets containing SBOMs.

This paper [13] by D.Vu et al. proposes an approach that detects code injected into software packages by comparing the source code repository with their distributed artifacts. For instance, comparing the code in the artifacts distributed in the package repository, such as The Python Package Index (PyPI), to the source code repository like GitHub. The authors of this paper propose this approach due to current malware detection techniques being resource-demanding and requiring knowledge of previous releases. Despite the authors proposing a new approach, this method has some limitations. One limitation is that it is dependent on source code repositories such as GitHub and PyPI; however, if the source code repository is not available, then this approach can not be applied. Furthermore, this approach is also limited to SSC attacks that include code injections into packages, whilst there are many other ways attackers compromise an SSC. This highlights that this approach may be useful in specific situations; however, it is not a method that can be applied in general cases, such as generative AI, to improve software supply chain security.

Kapil Singi et al. [8] discuss software development as a supply chain and outlines several associated challenges, such as provenance, security and accountability. The paper proposes a governance framework for distributed software development that records, monitors and analyses various activities throughout the application development life cycle. The framework aims to create a transparent development process with provenance and traceability alongside adhering to regulations, improving the trust in the developed software. The framework utilises software telemetry and distributed ledgers to record data and blockchain-enabled technology to record various activities and data. One flaw of such a system is the complexity of implementation due to the reliance on blockchain. The proposed system is complex and requires a strong understanding of blockchain technology, which increases the difficulty for smaller developers to use such a system. This is an important consideration due to the exponential growth of open-source software, which includes diverse software developers and skill sets [14]. Additionally, it is not stated or tested if the proposed framework would work with open-source generative AI software development, emphasising that there is little research for securing generative AI software supply chains.

Shripad Nadgowda [5] presented a novel framework, `engram`, that is applied at different stages during an application life cycle to modernise the approach of securing applications. The paper emphasises the importance of SBOMs, and the engram framework proposes new applications of SBOM. Currently, SBOM documents are passed through different processes, such as vulnerability scanning and license scanning, which involves updating the original SBOM document with the appropriate information. This process can become a problem due to the document constantly being changed, making it hard to establish the provenance of the data inside. Additionally, a function such as license scanning has access to other properties like vulnerability status, which can lead to malicious code exploiting this access. The engram model addresses this challenge by using a new model called the SBOM control panel, which treats SBOM as a distributed process involving multiple parties. This approach offers benefits such as efficient SBOM management and enhanced provenance tracking. However, one limitation of the engram system is the increase in complexity. Treating SBOMs as a distributed process makes the SBOM management system harder to understand. Additionally, this framework may also face some challenges due to the current uncertainties revolving around the quality of SBOMs produced [11].

Overall, these literature reviews outlined a gap in current datasets for generative AI. While there are datasets for general open-source software, there is a lack of datasets specifically created for generative AI software. This research aims to generate a golden dataset of

SSC metadata for generative AI to help users easily gain visibility into a software artefacts supply chain.

3 RESEARCH METHODOLOGY

This section outlines the research methodology followed to propose solutions for the three defined research questions. The research methodology applied a mixed method approach and was organised into three phases.

3.1 Reference Architecture Construction

A preliminary reference architecture (RA) for a generative artificial intelligence (GenAI) stack was constructed by adapting a systematic literature review process. This process aimed to support the identification of prominent open-source projects.

In particular we

(1) Identified various articles from both research and grey literature that described the architecture of GenAI. Such a process was completed using the Google search database with the following two search strings `Generative AI Architecture` and `Generative AI Stack`.
(2) Extracted recurring components (and connectors) from the identified architectures.
(3) Synthesised a preliminary RA from the list of common components. Furthermore, the RA created was refined by undergoing a peer-review process.

3.2 Gen AI Repository Identification

The components from the preliminary reference architecture (RA) established in RQ1 were used to discover existing types of generative AI projects. These identified types were translated into keywords, which allowed them to be searched on GitHub.

Using the identified keywords, the following steps were followed:

(1) The keywords were used in the GitHub search bar feature. This allowed us to discover numerous GitHub repositories for various components outlined in the preliminary reference architecture (established in RQ1).
(2) For each search conducted, the GitHub search results were filtered by best match and the relevant GitHub repositories were listed. This list of repositories was stored in a table.
(3) The source code from the identified repositories was collected. Additionally, if available, the source code from multiple releases was also collected to help understand the evolution of the software. This resulted in source code being collected for the first, middle and latest releases of such software. Additionally, the table was updated to display the number of versions collected for the corresponding repository.
(4) Generated Software Supply Chain (SSC) metadata using the source code collected from the identified repositories. This process was completed by utilising this existing tool [12].

3.3 SSC Metadata Analysis

An iterative exploratory data analysis was conducted using the SSC metadata that was generated from the results in RQ2. The following steps were followed to achieve this:

(1) A Python script was created to list the dependencies contained in each SBOM for every component type.
(2) The dependencies were displayed in a table alongside other information, such as the version of the software artefact and the total number of dependencies discovered for such version.
(3) The table was grouped together by software artefacts and sorted by version. This provided a visual analysis of the change in the number of dependencies over version releases.
(4) A Python script was created to read the tables and produce a graph that displays the average number of packages for each component type in the generative AI stack.
(5) An analysis on the results was conducted using the produced tables and graph

4 EXPERIMENTAL SETUP

The experiment's initial setup included creating folders for the corresponding component type within the generative AI stack. The SBOM files generated from the source code extracted from the GitHub repositories were then placed into their relative folders. The experimental setup also consisted of two Python scripts that accessed the folder directory to read through each of the SBOM files and generate results in the form of a CSV file.

4.1 GitHub Repository

For project reproduction purposes, a private GitHub repository was created containing the Python source code and the SBOM files used. The GitHub repository can be downloaded, and the Python script can be run with the same SBOM files generated in this project. It allows for an analysis of the dependencies different component types in the Generative AI ecosystem contain. **Link to GitHub Repository: click here**

5 RESULTS

The research methodology was followed to produce results to help aid in answering the three research questions for this project. The findings from this research are demonstrated in the next 3 sections.

5.1 Reference Architecture for Generative AI Stack

A preliminary reference architecture (RA) for a generative AI stack was constructed (Figure 1). The RA outlines the required components for a user's workflow when interacting with a text-to-text generative AI system, such as a chatbot. The RA can be broken down into five main layers: the user interface layer, the data preprocessing and embedding layer, the orchestration layer, the model layer and finally, the infrastructure layer. Each layer contains certain components that are essential for a generative AI model to run on a machine.

The user interface layer: This layer's primary responsibility is to construct a user interface (UI) to display the results produced by the generative AI model back to the user. For a text-to-text generative AI model, the user interacts with this UI to enter their text prompt, which takes such input and passes it through to the large language model to generate a response. This response is then passed back up to the UI tool, which then displays the response

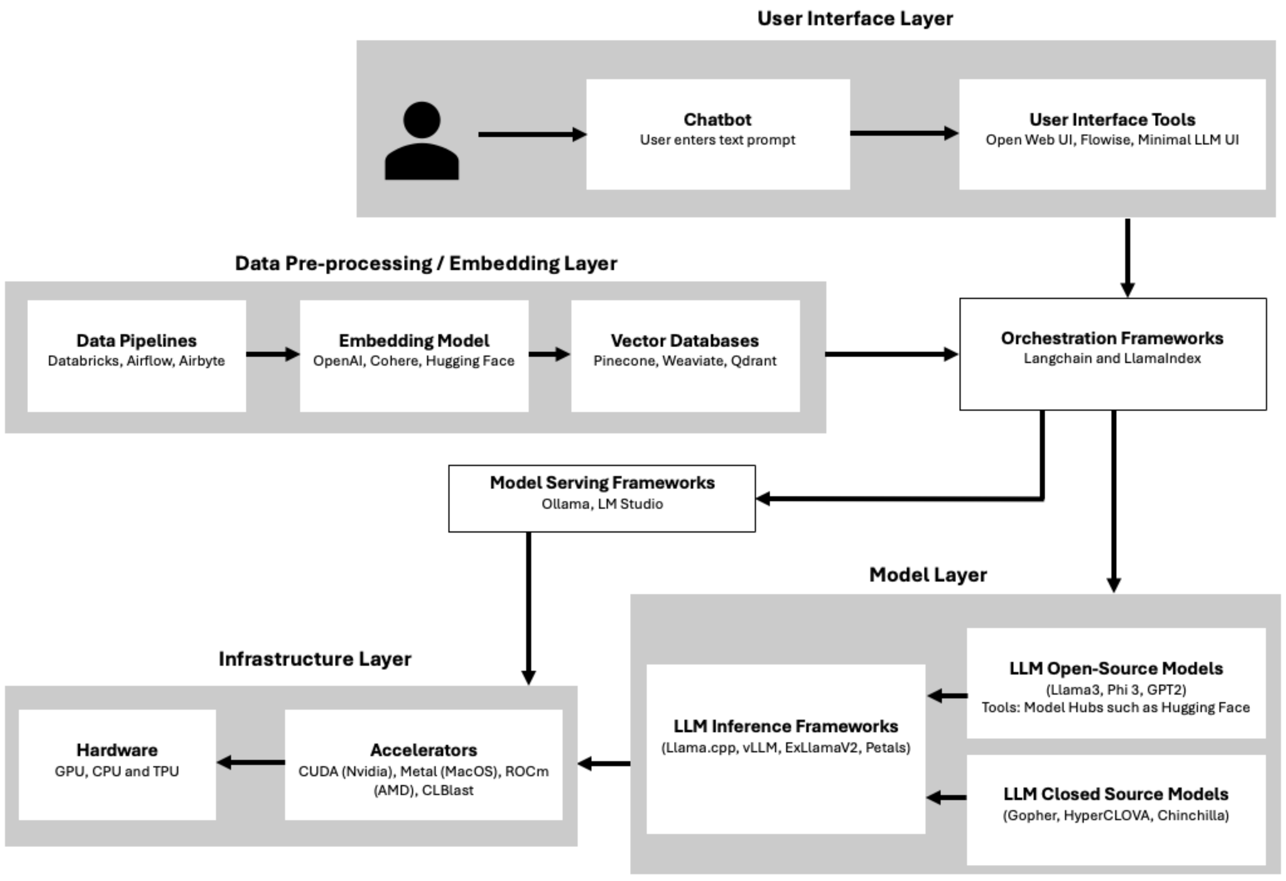


Figure 1: Preliminary Reference Architecture of Generative AI Stack

to the user. The user interface layer is an essential component within the generative AI ecosystem as it allows the user to easily interact with the generative AI model and provides the user with a pleasurable experience. Hence, there are a variety of open-source projects that have been developed and made available, each having a slightly different UI layout.

Data pre-processing and embedding layer: This layer involves the process of storing private data that can be retrieved by the generative AI model when required. In the pre-processing and embedding layer, data is typically passed through an embedding model and then stored within a vector database. In the generative AI ecosystem, large language models can only read and understand numeric data. Hence, the embedding model is responsible for transforming the data from non-numeric data, such as sentences/phrases, into a series of values that the large language model can understand. This numeric data then gets stored in a vector database. Vector databases are an important part of the pre-processing pipeline and are responsible for efficiently storing all the embeddings produced so they can be retrieved later. There are a variety of different vector databases that a developer may choose from when creating a generative AI model. As a result, there are many open-source projects that are available to use as a vector database.

Orchestration Layer: This layer consists of a singular component defined as orchestration frameworks or sometimes referred to as application frameworks. Orchestration frameworks are responsible for retrieving contextual data from vector databases and maintaining the memory of previous large language memory calls. As such, this process is beneficial for keeping track of the conversation history between the user and the generative AI model. The most widespread and common orchestration frameworks are Langchain and LlamaIndex. Additionally, there are other open-source projects that attempt to provide an alternative to these two main frameworks.

Model Layer: The model layer consists of the large language model (LLM) itself, as well as an LLM inference framework. One of the most important parts of developing a generative AI model is deciding which LLM to use. The LLM may be an open-sourced model or a closed-source model. There are existing tools known as Model Hubs, such as Hugging Face, which includes a variety of open-source pre-trained LLMs. LLM inference and serving frameworks are essential due to the large computational load and memory requirements that come with running an LLM. These inference frameworks aim to efficiently manage the LLM and ensure the model can be run on the available hardware. There are a variety of open-sourced inference frameworks available; however, certain frameworks may only be designed to be compatible with a specific LLM. Despite this, there are some inference frameworks, such as vLLM, that aim to be compatible with the majority of open-source LLMs available on Model Hubs, such as Hugging Face.

Infrastructure Layer: At the base of the generative AI stack is the infrastructure layer, which consists of the computer hardware and accelerator required to run the generative AI model. This layer within the generative AI ecosystem becomes highly important when developing and training large language models. It is essential that the necessary computational power and data processing capabilities are provided. The hardware can include a central processing unit (CPU), specific graphic processing units (GPUs), as well as a tensor processing unit (TPU). Furthermore, there are hardware accelerators that accelerate the AI computations and allow the LLM to run on the hardware. There are a variety of different accelerators depending on the specific type of hardware that is being used.

Summary: Different component types within the generative AI stack can be identified from this preliminary reference architecture. The four key components identified and used for the remaining parts of this project are LLM inference frameworks, orchestration frameworks, vector databases and user interface tools.

5.2 Generative AI Repositories and SSC Metadata Generation

The following four search strings were found to provide the best GitHub repositories that contained open-source projects for each of the component types in the reference architecture. These were the four search strings used,

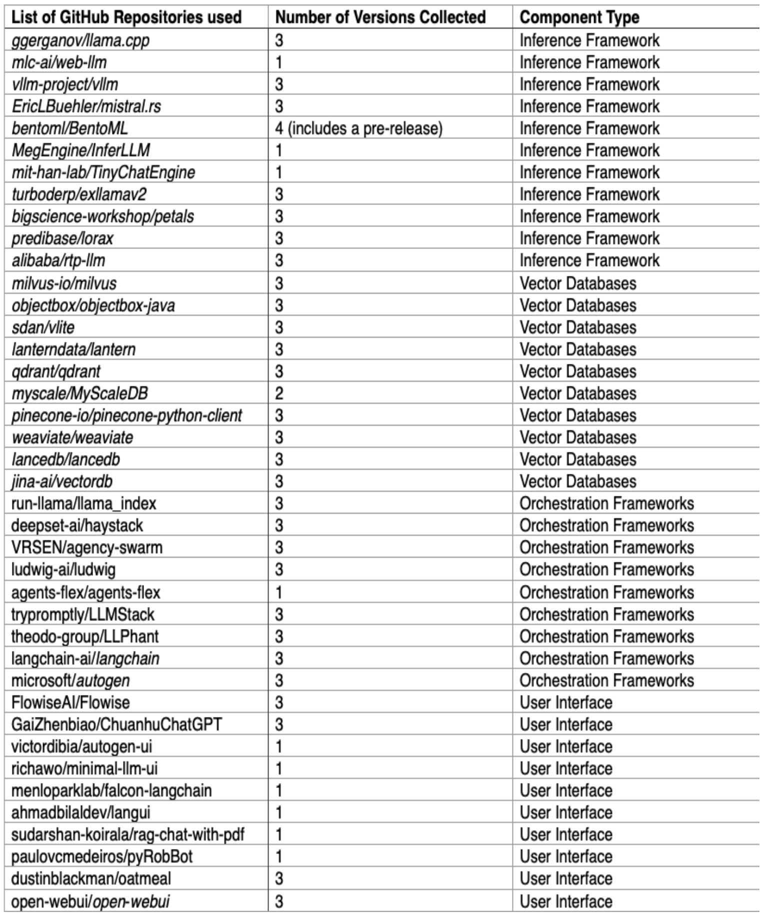
Inference Framework Search String: LLM Inference in: description, **Vector Database Search String:** Vector Database in: description **Orchestration Framework:** Langchain OR LangChain OR LLM orchestration OR application framework in:description and **UI Tools:** UI LLMs in:description

These four search strings on GitHub allowed for the identification of GitHub repositories. Source code was extracted from the GitHub repositories found, and a table was created with the relevant information. This included how many versions of the open-source project were collected, the name of the repository, and the corresponding component type from the RA. The table produced from this process can be seen in Figure 2. This aims to provide detail and insight into the exact open-source projects used to generate SBOMs, allowing for the reproduction of this project.

The source code collected was used to generate SBOMs for each of the versions. The existing SBOM generator tool produced a total of three SBOM files for every version; each SBOM file aimed to display the same results, just in different formats. This included a CycloneDX format, an SPDX format and a Syft format [12]. Overall, this resulted in a total of 300 SBOMs being produced over the 100 different releases collected. These SBOM files produced are available in the GitHub repository 4.1. For the remainder of this project and analysis, only the CycloneDX and SPDX formats were used.

5.3 SSC Metadata Analysis

We wrote a Python script to read through each of the SBOM files generated and extract the list of dependencies that each version



List of GitHub Repositories used	Number of Versions Collected	Component Type
ggerganov/llama.cpp	3	Inference Framework
mlc-ai/web-llm	1	Inference Framework
vllm-project/vllm	3	Inference Framework
EricLBuehler/mistral.rs	3	Inference Framework
bentoml/BentoML	4 (includes a pre-release)	Inference Framework
MegEngine/InferLLM	1	Inference Framework
mit-han-lab/TinyChatEngine	1	Inference Framework
turboderp/exllamav2	3	Inference Framework
bigscience-workshop/petals	3	Inference Framework
predibase/lorax	3	Inference Framework
alibaba/rtp-llm	3	Inference Framework
milvus-io/milvus	3	Vector Databases
objectbox/objectbox-java	3	Vector Databases
sdan/vlite	3	Vector Databases
lanterndata/lantern	3	Vector Databases
qdrant/qdrant	3	Vector Databases
myscale/MyScaleDB	2	Vector Databases
pinecone-io/pinecone-python-client	3	Vector Databases
weaviate/weaviate	3	Vector Databases
lancedb/lancedb	3	Vector Databases
jina-ai/vectordb	3	Vector Databases
run-llama/llama_index	3	Orchestration Frameworks
deepset-ai/haystack	3	Orchestration Frameworks
VRSEN/agency-swarm	3	Orchestration Frameworks
ludwig-ai/ludwig	3	Orchestration Frameworks
agents-flex/agents-flex	1	Orchestration Frameworks
trypromptly/LLMStack	3	Orchestration Frameworks
theodo-group/LLPhant	3	Orchestration Frameworks
langchain-ai/langchain	3	Orchestration Frameworks
microsoft/autogen	3	Orchestration Frameworks
FlowiseAI/Flowise	3	User Interface
GaiZhenbiao/ChuanhuChatGPT	3	User Interface
victordibia/autogen-ui	1	User Interface
richawo/minimal-llm-ui	1	User Interface
menloparklab/falcon-langchain	1	User Interface
ahmadbilaldev/langui	1	User Interface
sudarshan-koirala/rag-chat-with-pdf	1	User Interface
paulovcmedeiros/pyRobBot	1	User Interface
dustinblackman/oatmeal	3	User Interface
open-webui/open-webui	3	User Interface

Figure 2: List of Repositories used to Extract Source Code

incorporated. The Python script was run for each component type from the RA, which then generated a table for each. This table stores the name of the open-source project, how many dependencies were listed in the SBOM file, the SBOM format type, and a list of all the dependency names. This table is saved as CSV file, these tables can be seen through the GitHub repository 4.1.

The tables produced from this Python Script contain beneficial metadata surrounding each open-source project within a generative AI ecosystem. Specifically, the results seen from this metadata highlight the list of dependencies as well as the number of dependencies each software artefact and version use. This table is extremely beneficial as it highlights the differences between the number of dependencies each version contains. This allows developers that are considering using a particular software artefact in their generative AI model to see the change between different versions and the dependencies/packages they rely on.

The list of dependencies a software artefact contains can be checked against known vulnerabilities. Hence, the results tables produced from this project, can lead to further research surrounding the vulnerabilities contained within software artefacts for the generative AI ecosystem.

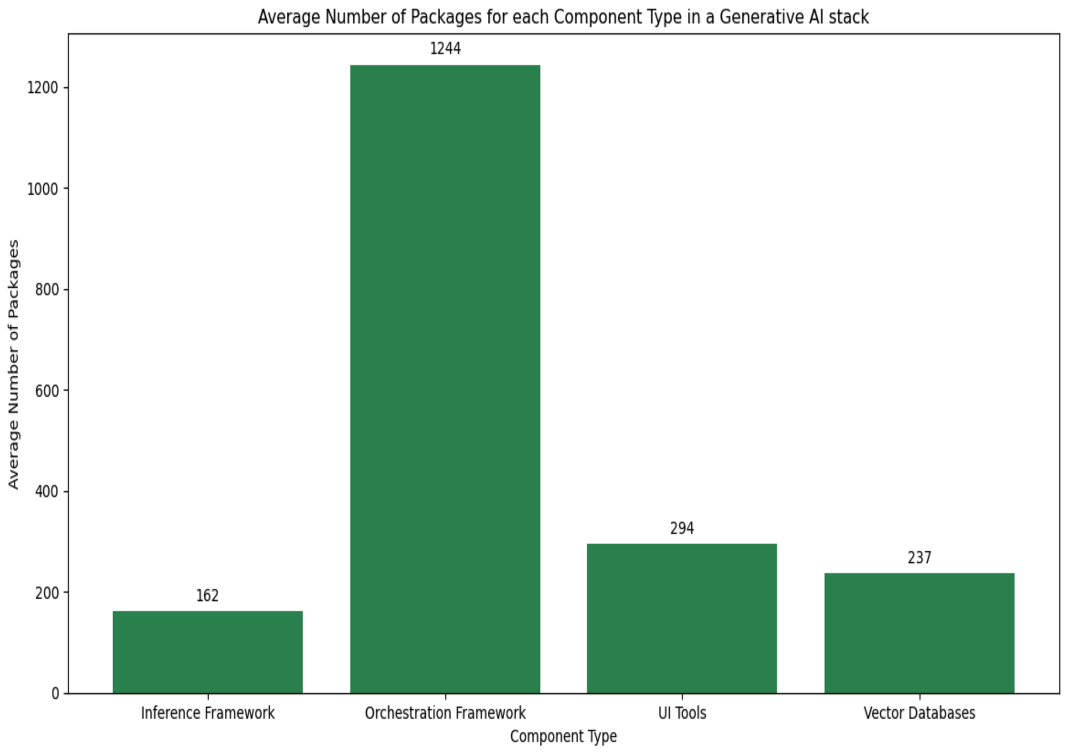


Figure 3: Graph Displaying Average Number of Packages for each Component Type

Additionally, considering SBOM files were collected for four different component types within the RA of the generative AI stack, we can compare the number of packages each component type typically requires. The results of this can be displayed in a column graph, as it allows for visual representation on the differences between the number of dependencies/packages each component type requires. Figure 3 shows the graph produced from the Python script created to read through each of the results tables produced earlier in the project. The graph displays the average number of packages each component type uses.

From the graph produced (Figure 3) it is clear that orchestration frameworks on average require and depend on the most amount of packages. If these packages are open-source projects then orchestration frameworks are at a higher risk for SSC attacks. This is because there are more packages it depends on, therefore, a higher probability of malicious code being injected into one of these open-source projects further down the SSC with the intent comprise the orchestration framework. Where as, the inference framework on average only used 162 packages compared to the 1244 average number of packages the orchestration framework contained.

To summarise the result tables produced and saved as CSV files, are beneficial to identify the change in number of packages between version releases within a specific component type of the generative AI stack. Whilst combing the result tables for each of the component types allows for an analysis on comparing the number of packages used across the different component types. Overall, this dataset of metadata and information produced is beneficial to researchers and developers working in the current field of open-source generative AI ecosystem, as it lists the dependencies/packages that are commonly used by different software artefacts for text-to-text generative AI models.

6 CONCLUSION

This paper presented a reference architecture (RA) for a text-to-text generative AI model (Figure 1). Such RA allowed for the identification of open-sourced projects within the generative AI ecosystem, in which software supply chain (SSC) metadata was generated from the source code. The SSC metadata produced for different components within the generative AI ecosystem can be stored in a dataset for further research and analysis purposes. Such a dataset is useful for the efficient identification of further dependencies each open-source project incorporates. This proves beneficial for developers working on text-to-text generative AI models as they are able to use the list of dependencies for various components and check against known vulnerabilities before using such open-source projects in their own software.

Further research can be conducted to explore the entire generative AI ecosystem. This would include expanding the current RA to include the fine-tuning process of large language models. In addition, other types of generative AI models can be explored to ensure the entire generative AI ecosystem is being inspected. This may include text-to-speech or image-to-image generative AI models. In future research, the current dataset can be expanded to include additional SSC metadata, such as software provenance statements. This would provide further information on the open-source projects for each component type within the generative AI ecosystem.

Certain limitations need to be further explored in this research project. The existing SBOM generation tool used for this project is still continually developing and needs to be fine-tuned. This is evident considering the number of packages found for a software artefact varied between the two SBOM formats.

REFERENCES

[1] Cybersecurity & Infrastructure Security Agency. [n. d.]. Software Bill of Materials (SBOM). ([n. d.]). Retrieved March 17, 2024 from https://www.cisa.gov/sbom#:~:text=A%20%E2%80%9Csoftware%20bill%20of%20materials,that%20make%20up%20software%20components.

[2] McKinsey & Company. 2023. The state of AI in 2023: Generative AI's breakout year. (2023). Retrieved March 18, 2024 from https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year#/

[3] Stefan Feuerriegel, Jochen Hartmann, Christian Janiesch, and Patrick Zschech. 2024. Generative AI. *Bus Inf Syst Eng 66(1):111–126 (2024)* (2024). https://doi.org/10.1007/s12599-023-00834-7

[4] Nick Miethe. 2023. Securing Your Software Supply Chain: Tracing it Back to the Start. (2023). Retrieved March 17, 2024 from https://meatybytes.io/posts/openshift/ocp-features/security/sscs/ssc-foundations/

[5] Shripad Nadgowda. 2022. Engram: The One Security Platform for modern Software Supply Chain Risks. *WoC '22: Proceedings of the Eighth International Workshop on Container Technologies and Container Clouds* (Nov. 2022), 2. https://doi.org/10.1145/3565384.3565889

[6] Marc Ohml, Henrik Plate, Arnold Skyosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. *17th International Conference, DIMVA 2020 Lisbon, Portugal, June 24–26, 2020 Proceedings* (2020). https://doi.org/10.1007/978-3-030-52683-2_2

[7] Jacob Schmitt. 2023. Software supply chain: What it is and how to keep it secure. (2023). Retrieved March 17, 2024 from https://circleci.com/blog/secure-software-supply-chain/

[8] Kapil Singi, R.P. Jagadeesh Chandra Bose, Sanjay Podder, and Adam P. Burden. 2019. 'Trusted Software Supply Chain'. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2019). https://doi.org/10.1109/ASE.2019.00141

[9] Sonatype. 2020. 2020 State of the Software Supply Chain. (2020). Retrieved March 17, 2024 from https://www.sonatype.com/hubfs/Corporate/Software%20Supply%20Chain/2020/SON_SSSC-Report-2020_final_aug11.pdf

[10] Sonatype. 2023. 9th Annual State of the Software Supply Chain. (2023). Retrieved March 17, 2024 from https://www.sonatype.com/hubfs/2023%20Sonatype-%209th%20Annual%20State%20of%20the%20Software%20Supply%20Chain-%20Update.pdf

[11] Santiago Torres-Arias, Dan Geer, and John Speed Meyers. 2023. A Viewpoint on Knowing Software: Bill of Materials Quality When You See It. *IEEE Security Privacy* (2023). https://doi.org/10.1109/MSEC.2023.3315887

[12] Nguyen Khoi Tran, Samodha Pallewatta, and M. Ali Babar. 2023. 'Toward a Reference Architecture for Software Supply Chain Metadata Management'. (2023). https://arxiv.org/pdf/2310.06300.pdf

[13] Duc-Ly Vu, Ivan Pashchenko, Fabio Massacci, Henrik Plate, and Antonino Sabetta. 2020. Towards Using Source Code Repositories to Identify Software Supply Chain Attacks. *CCS '20: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020). https://doi.org/10.1145/3372297.3420015

[14] Yunwen Ye and Kouichi Kishida. 2003. Toward an Understanding of the Motivation of Open Source Software Developers. *25th International Conference on Software Engineering, 2003. Proceedings* (2003), 1–4. https://doi.org/10.1109/ICSE.2003.1201220