In the Shadows of Dependency: Can AI Unmask the Hidden Threats Lurking in the Software Supply Chain?

Anonymous submission

Abstract

In the interconnected landscape of modern software development, the increasing reliance on third-party dependencies has heightened the risk of supply chain vulnerabilities. Manual analysis of such vulnerabilities is often time-consuming and resource-intensive. Recent advancements in the use of Large Language Models (LLMs) have shown promise in automating this process, potentially reducing costs and improving response times. This paper presents a comparative analysis of traditional manual methods and LLM-based approaches in identifying and analyzing software supply chain failures. Building on previous studies by Singla et al. (2023) and Anandayuvaraj et al. (2024), we demonstrate how LLMs, particularly with the use of advanced prompt engineering techniques, can enhance the accuracy and efficiency of software failure analysis. Our findings indicate that LLMs can effectively replicate manual analysis and with further improvements replace human expertise. This study provides a foundation for leveraging LLMs in proactive software supply chain risk management, with potential applications beyond the software industry.

Code — https://anonymous.4open.science/r/ECE-570-project-4C6B

Introduction

In today's interconnected world, software plays an essential role in nearly every facet of daily life, supporting everything from personal communication to critical infrastructure (Jones and Bonsignour 2011). To accelerate development timelines and reduce costs, modern software applications increasingly rely on extensive external dependencies, both directly (e.g., through libraries) and indirectly (e.g., the dependencies of those libraries). This reliance on third-party components has a significant impact on an application's risk profile: studies estimate that in a typical web application, approximately 80% of the codebase comes from dependencies, leaving only 20% as custom business logic (Pashchenko et al. 2018; Synopsys).

While this approach offers efficiency, it also introduces substantial risk, particularly through the *software supply chain*—a complex web of dependencies maintained by external entities (Ellison et al. 2010a; Gokkaya, Aniello, and Halak). Supply chain vulnerabilities and attacks allow malicious actors to infiltrate software indirectly, inserting or exploiting insecure code that becomes embedded in applications and, ultimately, exploitable at deployment (Okafor et al. 2022).

A failure-aware engineering approach, which examines previous failures to avoid repeating them, is increasingly valuable for software development (Petroski et al. 1994; Anandayuvaraj et al. 2023). Despite organizations' reluctance to disclose internal security incidents, insights into past failures can often be gathered from publicly available sources like news articles and technical reports. These provide useful data for understanding vulnerabilities (Anandayuvaraj and Davis 2023). Such information, categorized as "Open-Source Intelligence" (Team 2022), is frequently employed by government and defence organizations to strengthen security frameworks and develop effective countermeasures (Gill 2023).

This paper explores how to improve the process of analyzing open-source intelligence and historical incidents. This can help us form a resilient approach to managing software supply chain risks, supporting the secure and reliable development of software in an increasingly interconnected and interdependent digital ecosystem.

We use a holistic metric (cost, accuracy, time, etc.) to analyze and compare the current approaches to use open-source intelligence to help with future software supply chain failures. We suggest further improvements to these methods. An overview of the approach is given in Fig. 1. We observed that emerging automated approaches leveraging Large Language Models for classifying and learning from failures are advancing rapidly, offering substantial benefits, including high-speed, cost-effectiveness, and scalability. Additionally, our proposed enhancements aim to further improve these methods, boosting their accuracy and reliability. Our contributions include:

- A comparative analysis of current methods to solve software supply chain failures
- An evaluation of suggested improvements upon these methods to solve software supply chain failures

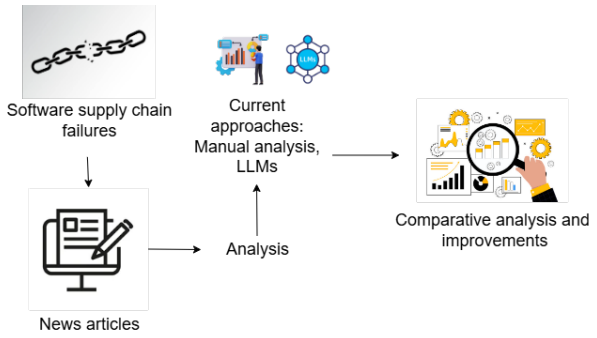

Figure 1: Performing comparative analysis of current approaches (manual analysis, LLMs, etc) to analyze open-source software failures (news articles, blogs). Finding which method work the best and suggesting improvements.

Background and Related works

Studying prior failures

Examining past failures across various fields has proven invaluable for enhancing safety, refining processes, and advancing knowledge. In aerospace engineering, analyzing incidents like the Challenger and Columbia disasters prompted NASA to strengthen safety protocols and risk assessment practices (Vaughan 1996). Similarly, medical research into errors and complications has led to standardized procedures and tools, such as electronic health records and checklists, inspired by studies from both medical and non-medical domains (Gawande 2010; Leape 2002).

Civil engineering has likewise benefited from failure case studies, with events like the Tacoma Narrows Bridge collapse prompting stricter building codes and structural design improvements (Petroski 1994). Studying prior failures is becoming increasing popular in software engineering as well. Learning from security breaches, including the Equifax incident, has led to the development of comprehensive cybersecurity frameworks like the NIST standards, which guide risk management and incident response (Cybersecurity 2018). Other works (Singla et al. 2023; Anandayuvaraj et al. 2024a) have also highlighted the importance of studying past failures in the field of software to aid secure systems.

Studying failures has, therefore, driven significant improvements across fields, fostering resilience, safety, and better design standards.

Software Supply Chain and Failures

The approach to software development has shifted significantly over the years. Initially, developers wrote code from scratch, which was costly and time-intensive (Maxam 2023). With the rise of reusable libraries and open-source components, developers increasingly rely on *dependencies*—external modules like libraries and packages—which streamline production but introduce new security risks (Vasilakis et al. 2021). This evolution has led to the concept of the *software supply chain*, encompassing the code, dependencies, tools, and processes involved in creating and maintaining software products (Ellison et al. 2010b).

Reliance on third-party code has rapidly increased, with a 2023 Synopsys report finding that nearly 80% of software codebases contain open-source components (Pashchenko et al. 2018). While beneficial, this dependency-based model exposes applications to risks; vulnerabilities in a single component can compromise numerous systems. Sonatype reports a stark increase in supply chain attacks, from 216 incidents between 2015 and 2019 to over 88,000 by 2022 (Sonatype 2022). High-profile breaches, such as SolarWinds and ShadowHammer, underscore the potential impact of compromised dependencies (Huddleston et al. 2021). Researchers have increasingly focused on understanding software supply chain attacks. Studies by Ohm (Ohm et al. 2020), Ladisa (Ladisa et al. 2022), Zimmerman (Zimmermann et al. 2019), and Zahan (Zahan et al. 2022) have examined and categorized various types of attacks within the software supply chain.

LLMs and NLPs in Software

Large Language Models (LLMs) and Natural Language Processing (NLP) have gained considerable attention for their applications in software development, especially in automating tasks and enhancing productivity. LLMs, such as GPT and BERT, are designed to process and generate human-like language, enabling capabilities in code generation, error detection, and documentation (Brown 2020). By training on vast amounts of text data, these models can predict, interpret, and even refactor code, making them highly effective for tasks like auto-completion and code review (Chen et al. 2021).

In the context of software supply chains, LLMs can assist in identifying vulnerabilities, automating security tasks, and detecting potential risks across dependencies by analyzing patterns within vast codebases and dependencies (Navarro 2024). NLP models are also used in software documentation, providing real-time, context-aware assistance that simplifies understanding complex code structures (Feng et al. 2020). Furthermore, integrating LLMs into development environments supports collaborative coding and enhances productivity by reducing the manual effort needed for code comprehension and debugging.

The adaptability of LLMs and NLPs in understanding both structured and unstructured data is transforming software engineering, creating new opportunities for efficiency and security within the software supply chain .

Using LLMs and NLPs to study prior failures

Large Language Models (LLMs) and Natural Language Processing (NLP) tools have recently shown promise in analyzing historical failures across domains. By processing vast datasets of incident reports, research papers, and technical documentation, LLMs can identify patterns and common causes of failures, aiding in risk assessment and mitigation strategies. For instance, LLMs like BERT and GPT-3 can extract relevant insights from unstructured data sources, making it easier to detect trends and recurrent issues in complex fields like cybersecurity and aerospace engineering (Brown 2020).

NLP models, which excel in text analysis, can mine data from failure logs, security advisories, and documentation to highlight factors contributing to past incidents. This capability aids developers in predicting potential points of failure and enhancing code robustness by learning from prior mistakes (Anandayuvaraj et al. 2024a).

The integration of LLMs and NLPs into the study of past failures holds significant potential for preventive measures across industries, turning historical data into actionable insights to improve resilience and security.

As previous noted manual methods to study are slow and often are unsuccessful in preventing past failures (Anandayuvaraj et al. 2024a). In addition, the newer methods to use LLMs although evolving and promising are not evaluated enough and we aim to bridge this gap.

Problem Definition

In recent years, the software supply chain has become increasingly vulnerable to attacks, with a dramatic rise in incidents underscoring its critical security weaknesses. A 2021 Sonatype report (Sonatype 2021) revealed a stark escalation: from 216 attacks between February 2015 and June 2019, to 929 recorded between July 2019 and May 2020, to over 12,000 incidents documented by 2020–2021. By 2022, this figure had surged to 88,000 (Sonatype 2022). Such breaches, including high-profile cases like SolarWinds (Huddleston et al. 2021) and ShadowHammer (Kaspersky 2019), expose severe vulnerabilities that pose risks to national security and organizational integrity worldwide.

Current approaches to analyzing failures in the software supply chain are largely manual, requiring extensive time and specialized expertise to read through vast amounts of technical documentation, vulnerability reports, and dependency changes. This manual analysis results in significant lag, delaying response times and limiting the proactive mitigation of emerging threats.

To address these challenges, recent research has begun exploring AI-driven solutions, particularly using NLP and Large Language Models (LLMs), for automating the identification of software vulnerabilities. While these AI-driven methods show promise in improving the speed and accuracy of risk assessment, they are not yet widely adopted or fully integrated into standard supply chain security practices. Our project will evaluate these emerging methods, based on studying the model's responses and metrics such as accuracy and F-1 score. We will also propose improvements to make them more effective and feasible for widespread use. We will test the proposed enhancements through comprehensive evaluations using open-source software failure data and quantitative metrics. This paper serves as a stepping stone toward enhancing supply chain resilience, with a focus on identifying areas where LLM-based tools can provide improved insights and reliability by extended the work of Singla et al. (Singla et al. 2023).

Methodology

Review of Existing Approaches

This study first comprehensively reviews traditional and machine learning-based methods for analyzing software supply chain vulnerabilities. Conventional approaches, as highlighted in studies by Ellison et al. (Ellison et al. 2010a) and Petroski et al. (Petroski 1994), rely on manual analysis of open-source intelligence (OSINT) data, such as technical documentation and vulnerability reports, to identify failure patterns. Although these methods provide detailed insights, they are time-consuming, require specialized expertise, and are limited in scalability. Previous work has demonstrated that while manual approaches are thorough, they are inadequate for rapidly identifying vulnerabilities across complex supply chains in modern software ecosystems.

In addition to manual techniques, prior research applied early machine learning (ML) approaches, including decision trees and clustering models, to automate the classification of vulnerabilities. However, as studies by Zimmermann et al. (Zimmermann et al. 2019) have noted, these models often struggled to capture nuanced dependencies and latent security threats within large datasets. Consequently, these early ML techniques were effective but limited in adapting to the complexity of evolving supply chain attacks.

We reviewed several studies (Singla et al. 2023; Anandayuvaraj et al. 2024b; Detloff 2024) that employed Large Language Models (LLMs) as an alternative to address the limitations of manual analysis—such as being time-consuming and costly—as well as the challenges faced by early ML techniques in capturing nuanced patterns. These studies indicate that LLMs may offer an optimal solution, balancing efficiency with the ability to capture complex dependencies. Building on these insights, we explored methods to enhance the effectiveness of LLMs for analyzing software supply chain vulnerabilities. The rest of this section includes the methods used to improve the effectiveness of LLMs.

Data Selection

As this paper extends the work of Singla et al. (Singla et al. 2023), it is essential to maintain coherence by using the same dataset employed by the previous authors to evaluate the LLMs. We use the CNCF's "Catalog of Supply Chain Compromises" as the baseline dataset (CNCF Security Technical Advisory Group 2023). This is a catalog of 69 software supply chain security failures analyzed from news articles and blogs from 1984-2022, maintained by the Cloud Native Computing Foundation (CNCF). Each entry briefly describes the failure and its impacts. These articles were not authored by the CNCF; instead, they were selected by the CNCF as reliable descriptions of the failures. The level of detail in the articles varied considerably.

The dimensions for analysis, as outlined in Table 2, were also reused to maintain consistency and minimize bias. For each dimension, the LLMs were given the content of these articles and asked to classify each failure using the prompts (refer to code under abstract for the prompts). The dimension *Lessons Learned* was excluded, as it is subjective and challenging to compare directly with the work of Singla et

al.

LLM selection

We used two popular, state-of-the-art LLMs that are publicly available at the time of writing (November 2024): OpenAI's GPT-4o model (OpenAI 2024) and Google's Gemini-1.5 pro model (Blog 2024). Their properties are summarized in Table 1. Other large language models are available, Claude (Anthropic 2024) and Llama (Llama 2024), but GPT and Gemini are the most widely used. Also, it draws a better comparison with an earlier study conducted by Singla et al. (Singla et al. 2023) which used OpenAI's GPT-3.5 model (OpenAI 2024) and Google's Bard (Google 2024).

ChatGPT-4o, OpenAI's LLM GPT-4o is a large language model created by OpenAI. It uses a deep learning architecture known as a transformer. Following pre-training, GPT-4o is further refined through a meticulous fine-tuning process with human oversight, which enhances its ability to generate precise, contextually appropriate responses. These capabilities position GPT-4o as a versatile tool for various applications, spanning domains such as customer support, virtual assistance, content synthesis, and educational support (Dwivedi 2024).

Gemini-1.5 pro, Google's LLM Gemini is another popular and accurate LLM created by Google. Gemini is Google's next-generation AI model designed to perform a wide range of natural language and multimedia tasks. Built to be versatile and highly responsive, Gemini integrates advanced language understanding, image processing, and multi-modal capabilities, making it suitable for applications in conversation, content creation, and real-time information retrieval. (Blog 2024).

Prompt engineering

A prompt refers to the specific instructions or questions provided to a Large Language Model (LLM). The response generated by an LLM can vary significantly, often as a result of even minor adjustments to the prompt (Liu et al. 2021).
Prompt engineering is the method of carefully designing these prompts to optimize the quality of the model's responses (White et al. 2023).

We used prompt engineering to iteratively develop prompts. We referred to various studies on prompt engineering (White et al. 2023; Dang et al. 2022; OpenAI 2023). We expanded upon the prompt engineering techniques outlined in the study by Singla et al. (Singla et al. 2023). Following a similar approach, we refined the prompt by starting with a basic query and incrementally applying each prompt engineering technique in a cumulative sequence until performance reached its peak, retaining adjustments that demonstrated improvement. We tested different prompt engineering approaches like one-shot and few-shot prompting. Table 4 illustrates our methodology, using the dimension "Type of Compromise" as an example. This prompt engineering phase was conducted on a subset of 20% of the dataset, specifically utilizing the most recent articles from the catalog as of June 2023. This prompt engineering phase was conducted on a subset of 20% of the dataset; we used the most recently published articles from the catalog as of June 2023. Refer to *code* for the final version of each prompt.

Experimental Results

LLMs to solve software supply chain failures

As this field is still emerging, there isn't a substantial body of literature available to definitively answer whether LLMs can fully replace manual analysis. However, Recent studies have explored the potential of Large Language Models (LLMs) to assist or even replace manual analysis in examining software failures. Singla et al. (Singla et al. 2023) demonstrated that LLMs could effectively replicate manual analyses of software supply chain security failures, achieving an average accuracy of 68% with GPT-3.5 across various dimensions such as type of compromise, intent, nature, and impacts.

More recent work by Anandayuvaraj et al. (Anandayuvaraj et al. 2024a) introduced the Failure Analysis Investigation with LLMs (FAIL) system, which automates the collection, analysis, and summarization of software failures reported in the news. FAIL achieved an F1 score of 90% in identifying relevant news articles and extracted 90% of the facts about failures, indicating that LLMs, when precisely utilized, can accurately label data related to software failures. While these advancements are promising, further improvements are necessary, as explored in the following sections.

Improved LLM techniques

To extend the study by Singla et al. (Singla et al. 2023) and suggest techniques to improve the use of LLMs to analyze and mitigate software supply chain failures, we focused on 2 techniques.

- Integrating different models (Table 2, Table 3)
- Developing better prompts (Table 4)

The accuracy of GPT-4o improved across all dimensions, with the most notable gains observed in *Impacts*, where accuracy increased from 50% to 70%, and *Nature*, which rose from 69% to 84%. Although Gemini 1.5-pro was tested on approximately 30% of the dataset, it also demonstrated substantial improvements, particularly in the *Type of Compromise* dimension, where accuracy nearly doubled from 35% to 65%.
The overall accuracy distribution for all articles across the correct dimensions using GPT-4o is displayed in Fig. 2. Nearly all articles (63 out of 65) achieved an accuracy of 50% or higher, with 49 articles reaching 75% or higher—a considerable improvement over the results reported in the previous work by Singla et al.
Furthermore, the application of advanced prompt-engineering techniques (White et al. 2023) led to improved results. As shown in Table 4, one-shot prompting with GPT-4o increased accuracy on the testing dataset by 7%, achieving a final accuracy of 85%. This table builds upon Table 6 in the work of Singla et al., providing a comparative extension of their findings.

Table 1: Specifications of the LLMs : GPT-4o, Gemini 1.5 pro, Claude 3.5 Sonnet

Model	Cost-to-access	Token limit	Parameters	Tuning knobs
GPT-4o	Input: $2.50/1M tokens, Output: $10/1M tokens, Cached: $1.25/1M tokens	128k input tokens, 16K output tokens	est. 1.8 trillion	**Temperature**: Higher values mean greater randomness of the new predicted word. Default: 1. **Top_P**: Nucleus sampling. Model considers the results of tokens with top_p probability mass. top_p = 0.2 means when predicting the next word, only tokens in the top 20% probability mass are considered. Default: 1.
Gemini 1.5 pro	Free: 2 RPM, 32k TPM, 50 RPD Pay: Input: $1.25/1M tokens, Output: $5/1M tokens, Cached: $0.3125/1M tokens	2M input tokens, 8k output tokens	1.5 trillion	**Temperature, Top_p, Top-K:** Top-K controls token selection by choosing from the K most probable tokens. A top-K of 1 means always selecting the most probable token (greedy decoding), while higher values introduce randomness. Tokens are first filtered by top-K, then by top-P, with temperature sampling determining the final choice. Lower values yield more deterministic responses; higher values add randomness. Default: 1.
Claude 3.5 Sonnet	Input-$3.00/1M tokens, Output-$15.00/1M tokens, Cached write: $3.75/1M tokens, Cached read: $0.3/1M tokens	200k input tokens, 8K output tokens	175 billion	**Temperature**

Table 2: Accuracy of the GPT-4.0 vs GPT-3.5 model after the current reimplementation

Dimension	GPT 4.0	GPT 3.5
Type of compromise	66%	59%
Intent	95%	90%
Nature	84%	69%
Impacts	70%	50%

Table 3: Accuracy of the Gemini 1.5-pro vs Bard model after the current reimplementation for 20 articles

Dimension	Gemini 1.5-pro	Bard
Type of compromise	65%	35%
Intent	90%	85%
Nature	75%	70%
Impacts	55%	50%

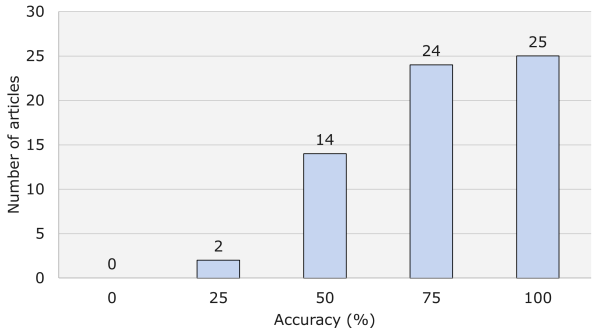

Figure 2: Distribution of the accuracy by articles for GPT-4.0. GPT answered 4 questions — so 5 possible outcomes per case. It shows most articles were classified accurately.

Conclusion and Future Directions

This study demonstrates that Large Language Models (LLMs) provide a viable and effective approach for analyzing historical software supply chain failures and identifying potential vulnerabilities in future scenarios. By leveraging advanced prompt engineering techniques, as explored in this paper, LLMs can offer significant improvements in accuracy and consistency, making them a valuable resource in failure analysis and risk management. Our findings indicate that LLMs are not only capable of addressing the time and resource limitations associated with traditional manual and early machine learning approaches but also

Table 4: Techniques used to improve the prompts, illustrated for the prompt associated with the dimension of type of compromise. *'ID'* denotes the order in which the techniques were used. The accuracy column contains the change in accuracy from the previous technique and the final accuracy in brackets. Accuracy was measured over 20% of the labelled data (we repeatedly analyzed the 14 most recent articles).

ID	Technique	Prompt	Accuracy (%)
0	Initial prompt without any techniques	"Classify the attack from the following choices Choice 1: Dev Tooling Choice 2: Negligence Choice 3: Publishing Infrastructure Choice 4: Source Code Choice 5: Trust and Signing Choice 6: Malicious Maintainer Choice 7: Attack Chaining Based on the information provided in the Articles. Article: {article} "	33
1	Providing context/definitions- adding definitions of the options (improving upon ID: 0)	"Classify the attack from the following choices Choice 1: Dev Tooling- Occurs when the development machine, SDK, tool chains, or build kit have been exploited. These exploits often result in the introduction of a backdoor by an attacker to own the development environment. Choice 2: Negligence- Occurs due to a lack of adherence to best practices. TypoSquatting attacks are a common type of attack associated with negligence, such as when a developer fails to verify the requested dependency name was correct (spelling, name components, glyphs in use, etc). ... Based on the information provided in the Article, Article: value"	+36 (69)
2	Reflection Pattern- asking the LLM to explain its answer (improving upon ID: 1)	Adding the sentence "Explain your answer using the given definitions and return the option." Before passing the article.	+2 (71)
3	Template technique (JSON format) and adding delimiters (improving upon ID: 2)	Adding "Use JSON format with the keys: 'explanation', 'choice'. Based on the information provided in the Article delimited by triple backticks. Article: `{article}`" in the end.	+7 (78)
4	One shot- adding an example (improving upon ID: 3)	"" ... Choice 7: Attack Chaining- Sometimes a breach may be attributed to multiple lapses, with several compromises chained together to enable the attack. The attack chain may include types of supply chain attacks as defined here. However, catalogued attack chains often include other types of compromise, such as social engineering or a lack of adherence to best practices for securing publicly accessible infrastructure components. For example: "The successful breach resulted from a phishing attack that targeted multiple Dropbox employees using emails impersonating the CircleCI continuous integration and delivery platform and redirecting them to a phishing landing page where they were asked to enter their GitHub username and password." is attack chaining and it involved social engineering and multiple steps. Explain your answer using the given definitions and return the option. Use JSON format with the keys: 'explanation', 'choice'. Based on the information provided in the Article, Article: value"	+7 (85)

excel at capturing nuanced dependencies within complex datasets. Therefore, methods such as cumulative prompt refinement, structured response formats, and context-aware prompting should be considered or at least explored when deploying LLMs for similar analytical tasks in software security.

Future research can extend the use of LLMs beyond the software industry to evaluate supply chain vulnerabilities across various sectors, such as healthcare, finance, and critical infrastructure. Investigating the applicability of these models in diverse contexts could provide insight into domain-specific vulnerabilities and further enhance LLM adaptability. Additionally, integrating multimodal models, like Gemini, which combines text and image processing, could broaden the scope of analysis by incorporating visual data. Another promising direction involves the refinement of LLM interpretability, enabling stakeholders to better understand model decisions and, ultimately, improve transparency and trust in AI-driven analysis. Continued advancements in LLM fine-tuning and prompt engineering are likely to further increase their efficacy, making them indispensable tools for proactive failure analysis across industries.

Acknowledgments

Per the class policies on the use of LLMs: OpenAI's ChatGPT model (v4-o) was used during manuscript preparation. Prompt: *Can you make the following clearer? "TEXT SNIPPET"*. The answers were reviewed to ensure it did not change the ideas.

References

Anandayuvaraj, D.; Campbell, M.; Tewari, A.; and Davis, J. C. 2024a. FAIL: Analyzing Software Failures from the News Using LLMs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, 506–518. New York, NY, USA: Association for Computing Machinery. ISBN 9798400712487.

Anandayuvaraj, D.; Campbell, M.; Tewari, A.; and Davis, J. C. 2024b. FAIL: Analyzing Software Failures from the News Using LLMs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, 506–518. New York, NY, USA: Association for Computing Machinery. ISBN 9798400712487.

Anandayuvaraj, D.; and Davis, J. C. 2023. Reflecting on Recurring Failures in IoT Development. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, 1–5. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-9475-8.

Anandayuvaraj, D.; Thulluri, P.; Figueroa, J.; Shandilya, H.; and Davis, J. C. 2023. Incorporating Failure Knowledge into Design Decisions for IoT Systems: A Controlled Experiment on Novices. In *5th International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT 2023)*.

Anthropic. 2024. Claude. Accessed: 2024-11-02.

Blog, G. D. 2024. Gemini API and AI Studio Now Offer Grounding with Google Search. Accessed: 2024-11-02.

Brown, T. B. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

CNCF Security Technical Advisory Group. 2023. Catalog of Supply Chain Compromises. https://github.com/cncf/tag-security/tree/main/supply-chain-security/compromises. GitHub repository.

Cybersecurity, C. I. 2018. Framework for improving critical infrastructure cybersecurity. *URL: https://nvlpubs. nist. gov/nistpubs/CSWP/NIST. CSWP*, 4162018: 7.

Dang, H.; Mecke, L.; Lehmann, F.; Goller, S.; and Buschek, D. 2022. How to prompt? Opportunities and challenges of zero-and few-shot learning for human-AI interaction in creative applications of generative models. *arXiv preprint arXiv:2209.01390*.

Detloff, M. 2024. Exploring the extent of similarities in software failures across industries using LLMs. *arXiv preprint arXiv:2408.03528*.

Dwivedi, L. 2024. An In-Depth Look at ChatGPT-4o: An AI Powerhouse. Accessed: 2024-11-02.

Ellison, R. J.; Goodenough, J. B.; Weinstock, C. B.; and Woody, C. 2010a. Evaluating and Mitigating Software Supply Chain Security Risks. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST. Section: Technical Reports.

Ellison, R. J.; Goodenough, J. B.; Weinstock, C. B.; and Woody, C. 2010b. Evaluating and mitigating software supply chain security risks.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Gawande, A. 2010. *Checklist manifesto, the (HB)*. Penguin Books India.

Gill, R. 2023. What is Open-Source Intelligence? https://www.sans.org/blog/what-is-open-source-intelligence/. Accessed: 2023-06-21.

Gokkaya, B.; Aniello, L.; and Halak, B. ???? Software supply chain: review of attacks, risk assessment strategies and security controls. .

Google. 2024. Bard, Google AI, and Search Updates. Accessed: 2024-11-02.

Huddleston, J.; Ji, P.; Bhunia, S.; and Cogan, J. 2021. How VMware Exploits Contributed to SolarWinds Supply-chain Attack. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, 760–765. IEEE.

Jones, C.; and Bonsignour, O. 2011. *The economics of software quality*. Addison-Wesley Professional.

Kaspersky. 2019. ShadowHammer: Malicious updates for ASUS laptops. https://www.kaspersky.com/blog/shadow-hammer-teaser/26149/. Accessed: 2023-06-18.

Ladisa, P.; Plate, H.; Martinez, M.; and Barais, O. 2022. Taxonomy of Attacks on Open-Source Software Supply Chains. .

Leape, L. L. 2002. Reporting of adverse events. *The New England journal of medicine*, 347(20): 1633–1638.

Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; and Neubig, G. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. arXiv:2107.13586.

Llama. 2024. Llama AI. Accessed: 2024-11-02.

Maxam, A. 2023. *Threat Hunting in Cybersecurity: A Comprehensive Study*. Master's thesis, University of Unknown.

Navarro, J. 2024. Machine Learning Algorithms in Supply Chain Vulnerability Management. *MZ Computing Journal*, 5(2): 1–7.

Ohm, M.; Plate, H.; Sykosch, A.; and Meier, M. 2020. Backstabber's knife collection: A review of open source software supply chain attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 23–43. Springer.

Okafor, C.; Schorlemmer, T. R.; Torres-Arias, S.; and Davis, J. C. 2022. SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain*

Offensive Research and Ecosystem Defenses, SCORED'22, 15–24. New York, NY, USA: Association for Computing Machinery. ISBN 9781450398855.

OpenAI. 2023. GPT Best Practices. https://platform.openai.com/docs/guides/gpt-best-practices.

OpenAI. 2024. OpenAI Platform - Models Documentation. Accessed: 2024-11-02.

Pashchenko, I.; Plate, H.; Ponta, S. E.; Sabetta, A.; and Massacci, F. 2018. Vulnerable open source dependencies: counting those that matter. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10. Oulu Finland: ACM. ISBN 978-1-4503-5823-1.

Petroski, H. 1994. *Design paradigms: Case histories of error and judgment in engineering*. Cambridge University Press.

Petroski, H.; et al. 1994. *Design paradigms: Case histories of error and judgment in engineering*. Cambridge University Press.

Singla, T.; Anandayuvaraj, D.; Kalu, K. G.; Schorlemmer, T. R.; and Davis, J. C. 2023. An empirical study on using large language models to analyze software supply chain security failures. In *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 5–15.

Sonatype. 2021. State of the Software Supply Chain. https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021.

Sonatype. 2022. State of the Software Supply Chain. Technical Report 8th Annual, Sonatype.

Synopsys. ???? 2023 OSSRA Report. https://www.synopsys.com/software-integrity/engage/ossra/rep-ossra-2023-pdf.

Team, T. R. F. 2022. What Is Open Source Intelligence and How Is it Used? https://www.recordedfuture.com/open-source-intelligence-definition. Accessed: 2023-06-21.

Vasilakis, N.; Benetopoulos, A.; Handa, S.; Schoen, A.; Shen, J.; and Rinard, M. C. 2021. Supply-Chain Vulnerability Elimination via Active Learning and Regeneration. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 1755–1770. Virtual Event Republic of Korea: ACM. ISBN 978-1-4503-8454-4.

Vaughan, D. 1996. *The Challenger launch decision: Risky technology, culture, and deviance at NASA*. University of Chicago press.

White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; and Schmidt, D. C. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*.

Zahan, N.; Zimmermann, T.; Godefroid, P.; Murphy, B.; Maddila, C.; and Williams, L. 2022. What are weak links in the npm supply chain? In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '22, 331–340. New York, NY, USA: Association for Computing Machinery. ISBN 9781450392266.

Zimmermann, M.; Staicu, C.-A.; Tenny, C.; and Pradel, M. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *28th USENIX Security Symposium (USENIX Security 19)*, 995–1010. Santa Clara, CA: USENIX Association. ISBN 978-1-939133-06-9.