

Master of Science in Software Engineering  
May 2023



# Fine-Tuning Large Language Models for Software Supply Chains Threats Mitigation

Hemanth Rachaboyina  
Lakshman Reddy Thummala

This thesis is submitted to the Faculty of Faculty at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to Weeks weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

**Contact Information:**

Author(s):

Hemanth Rachaboyina

E-mail: [hera22@student.bth.se](mailto:hera22@student.bth.se)

Lakshman Reddy Thummala

E-mail: [lath22@student.bth.se](mailto:lath22@student.bth.se)

University advisor:

Oleksandr Adamov [oleksandr.adamov@bth.se](mailto:oleksandr.adamov@bth.se)

Department of Software Engineering

Faculty of Faculty  
Blekinge Institute of Technology  
SE-371 79 Karlskrona, Sweden

Internet : [www.bth.se](http://www.bth.se)  
Phone : +46 455 38 50 00  
Fax : +46 455 38 50 57

---

# Abstract

The growing complexity and interconnectivity of software supply chains have elevated the risks of security threats, demanding innovative solutions. This thesis investigates the fine-tuning of Large Language Models (LLMs), particularly Microsoft Phi-2, to enhance their ability to identify and mitigate software supply chain vulnerabilities. Using advanced techniques such as Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA), the Phi-2 model was trained on a domain-specific dataset comprising incident reports, threat intelligence data, and best practices.

The methodology encompasses a rigorous evaluation process using quantitative metrics, including ROUGE, BERTScore, and BLEURT, supplemented by qualitative insights derived from semi-structured interviews with cybersecurity experts. The interviews revealed valuable perspectives on the practical applicability of the fine-tuned model in addressing real-world threats such as compromised third-party components, open-source dependency vulnerabilities, and emerging attack patterns.

The fine-tuned model exhibited significant improvements in generating contextually relevant, precise, and actionable threat mitigation strategies compared to its baseline. The findings demonstrate that domain-specific fine-tuning of LLMs is a viable approach for advancing automated threat detection and response capabilities in software supply chains. This research provides a robust framework for integrating AI-driven solutions into the software development lifecycle, contributing to the fields of software engineering and cybersecurity by improving resilience against supply chain attacks.

## **Keywords:**

Large Language Models, Fine-Tuning, Software Supply Chain Security, Threat Mitigation, Cybersecurity, Microsoft Phi-2, Low-Rank Adaptation (LoRA), Parameter-Efficient Fine-Tuning (PEFT), Automated Threat Detection, Software Engineering.

---

## Acknowledgments

During our thesis journey, we had the incredible opportunity to explore and deepen our knowledge across various domains. We extend our heartfelt gratitude to our supervisor, Dr. Oleksandr Adamov, for his unwavering guidance and dedication. His patience and support were invaluable whenever we faced challenges, and we are immensely grateful for his mentorship.

We also sincerely thank our participants, who took time out of their busy schedules to contribute to our research. Their responses were crucial to the completion of this thesis. Additionally, we extend our gratitude to the professors at Blekinge Institute of Technology for sharing their valuable knowledge and experiences, which greatly enriched our learning.

"The thesis process was a journey filled with challenges due to my limited knowledge and inexperience in this field. I am deeply thankful to my partner, Hemanth, for his understanding, support, and patience in explaining things that helped me navigate this journey. I also want to express my gratitude to my parents, relatives, and friends for their unwavering support and belief in me." - Lakshman.

"I am profoundly grateful to my parents, family, and friends for their unconditional love and encouragement. Their unwavering faith in me inspired me to pursue new paths and strive for success. I dedicate this achievement to them, as their support has been the foundation of my journey." - Hemanth

---

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	2
1.1.1 Large Language Models (LLMs) . . . . .	2
1.1.2 Software Supply chains threats . . . . .	5
1.2 Motivation . . . . .	6
1.3 Aim and Objectives . . . . .	6
1.4 Scope of the Research . . . . .	7
1.5 Outline . . . . .	8
<b>2 Related work</b>	<b>9</b>
2.1 Research Gap . . . . .	12
<b>3 Methodology</b>	<b>13</b>
3.1 Research Questions . . . . .	13
3.2 Literature Review . . . . .	14
3.3 Dataset . . . . .	15
3.3.1 Data Collection . . . . .	15
3.3.2 Data Cleaning and Preprocessing . . . . .	16
3.3.3 Data Augmentation . . . . .	16
3.3.4 Dataset Statistics . . . . .	16
3.3.5 Ethical Considerations . . . . .	17
3.4 Interview . . . . .	17
3.4.1 Sampling . . . . .	17
3.4.2 Data Collection . . . . .	18
3.4.3 Data Analysis . . . . .	18
3.5 Experiment . . . . .	19
3.5.1 Experimental Design . . . . .	19
3.5.2 Model Preparation . . . . .	20
3.5.3 Data Collection and Preparation . . . . .	20
3.5.4 Evaluation Metrics . . . . .	20
3.5.5 Experimental Procedure . . . . .	21
3.5.6 Summary . . . . .	21

<b>4</b>	<b>Implementation</b>	<b>24</b>
4.1	Low-Rank Adaptation (LoRA) . . . . .	24
4.1.1	Parameter-Efficient Fine-Tuning (PEFT) . . . . .	25
4.2	Practical Implementation in a Notebook . . . . .	26
4.2.1	Conclusion . . . . .	29
<b>5</b>	<b>Results and Analysis</b>	<b>30</b>
5.1	Literature Review Results and Analysis . . . . .	30
5.1.1	Literature Review . . . . .	30
5.1.2	Analysis from the literature review . . . . .	34
5.2	Interview Results . . . . .	35
5.2.1	Demographics . . . . .	35
5.2.2	Interview Summary and Findings . . . . .	36
5.3	Experiment Results . . . . .	40
5.3.1	Introduction . . . . .	40
5.3.2	Overview of Model Performance . . . . .	40
5.3.3	Comparison Between Baseline Phi-2 and Fine-Tuned Phi-2 . . . . .	43
<b>6</b>	<b>Discussion</b>	<b>62</b>
6.1	Validity Threats . . . . .	67
6.1.1	Construct Validity . . . . .	67
6.1.2	Conclusion Validity . . . . .	67
6.1.3	Internal Validity . . . . .	68
6.1.4	External Validity . . . . .	68
<b>7</b>	<b>Conclusion</b>	<b>69</b>
	<b>References</b>	<b>71</b>
<b>A</b>	<b>Interview Information</b>	<b>75</b>
A.1	Invitation Letter . . . . .	75
A.2	Questionnaire . . . . .	76
A.3	Questionnaire . . . . .	77
A.4	Questions by Participants to both Base and Fine-tuned Models . . . . .	78
A.5	Questions by Participants to both Base and Fine-tuned Models . . . . .	79

---

## List of Figures

1.1	LLM Architecture [33]	3
1.2	Fine Tune Large Language Model (LLM) [9]	4
3.1	Literature review architecture	22
3.2	Thematic Analysis	23
5.1	Distribution Vectors for Security Incidents	60

---

## List of Tables

5.1	Summary of Literature Review Findings on Fine-tuning Algorithms .	31
5.2	Participant Demographics . . . . .	35
5.3	ROUGE Score Comparison for Mitigation Strategies . . . . .	41
5.4	BERTScore Comparison for Mitigation Strategies . . . . .	42
5.5	BLEURT Score Comparison for Mitigation Strategies . . . . .	43

In contemporary software engineering, the integrity and security of software supply chains have become foundational elements of the software development lifecycle (SDLC). As software systems grow increasingly interconnected and complex, the networks that develop, distribute, and maintain these software products face mounting security challenges [2]. The software supply chain, encompassing code, configurations, libraries, plugins, open-source and proprietary binaries, and container dependencies, forms the backbone of modern software development practices. However, this interconnectivity introduces significant vulnerabilities that can compromise entire systems through various attack vectors.

Recent high-profile incidents have highlighted the critical nature of software supply chain security. The exploitation of vulnerabilities in the supply process, from third-party services to open-source code management, underscores the urgent need for robust security measures in software engineering [28]. These attacks demonstrate how malicious actors can manipulate the code in third-party software components to compromise downstream applications, leading to widespread security breaches across multiple organizations and industries.

The complexity of modern software supply chains presents unique challenges in the software engineering domain. As noted by Linton, Boyson, and Aje [2], organizations increasingly depend on open-source libraries, third-party components, and collaborative development methods. This dependency creates a networked system of online third-party source sharing, often reflecting a collaboration of multiple stakeholders in software development. The security implications of this interconnected environment are significant, as software security defects in any product or service present potential supply chain risks to all participants in the system-of-systems (SoS) context [13].

One particular concern is the fact that software is rarely defect-free, and many common defects can be readily exploited by unauthorized parties to alter security properties and functionality for malicious intent [12]. These vulnerabilities can be introduced at any point in the development or use of software, and subsequent acquirers and users have limited ways of finding and correcting these defects to avoid exploitation. The challenge is further compounded by the global nature of software supply chain participation, where knowledge of who has touched each specific product or service may not be visible to others in the chain.

This research aims to address these critical challenges by fine-tuning Large Lan-

guage Models (LLMs) specifically for software supply chain threat mitigation. By leveraging advanced machine learning techniques, we seek to enhance the capability of automated systems to identify, analyze, and mitigate potential threats within software supply chains. Our approach focuses on developing more sophisticated methods for detecting and responding to security vulnerabilities, particularly in the context of third-party components and open-source dependencies [32].

The significance of this research lies in its potential contribution to the software engineering domain, specifically in strengthening the security aspects of software supply chains. By developing more effective threat detection and mitigation strategies, this work aims to enhance the resilience of software systems against supply chain attacks. This is particularly crucial as organizations continue to rely heavily on third-party components and open-source software, making supply chain security an integral part of modern software engineering practices.

## 1.1 Background

### 1.1.1 Large Language Models (LLMs)

Large Language Models (LLMs) are a type of artificial intelligence designed to understand and generate human language. These models are built using deep learning techniques, particularly neural networks with many layers, which allow them to process and generate text with remarkable fluency and coherence [5]. LLMs are trained on vast amounts of textual data, encompassing diverse sources such as books, articles, websites, and other written content. This extensive training enables them to learn the nuances of language, context, and even some factual knowledge embedded in the data [19].

One of the most significant advancements in LLMs is their ability to perform a wide range of natural language processing tasks with high accuracy. These tasks include language translation, text summarization, sentiment analysis, and even generating creative writing [19]. The models achieve this by recognizing patterns and relationships in the data, allowing them to generate responses that are contextually relevant and coherent [5]. The most notable example of an LLM is OpenAI's GPT (Generative Pre-trained Transformer) series, which has set new benchmarks in the field of natural language understanding and generation.

The architecture of LLMs, particularly the Transformer model, plays a crucial role in their performance [33]. Transformers utilize self-attention mechanisms, which enable the model to weigh the importance of different words in a sentence when generating a response. This approach allows LLMs to handle long-range dependencies and maintain context over longer texts, making them more effective than previous models like recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) [40]. Despite their impressive capabilities, general-purpose Large Language Models (LLMs) face significant challenges when applied to domain-specific tasks such as cybersecurity. One major limitation is the lack of domain-specific knowledge since these models are trained on diverse, broad datasets that may not include special-

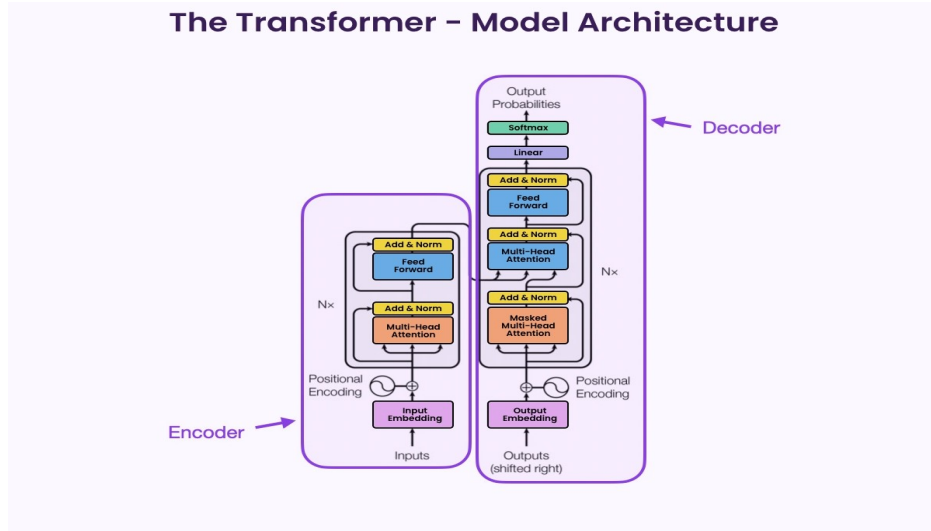


Figure 1.1: LLM Architecture [33]

ized cybersecurity terminology, threat intelligence reports, or attack vectors. As a result, they often struggle to accurately interpret and respond to complex security threats such as software supply chain risks and zero-day vulnerabilities. Additionally, LLMs are prone to hallucination and misinformation, meaning they can generate plausible but incorrect security recommendations. In cybersecurity, misleading advice can increase risk exposure rather than mitigate threats, potentially leading to flawed security strategies. Another critical issue is adversarial manipulation, where malicious actors exploit vulnerabilities in LLMs by poisoning training data or crafting misleading prompts to bypass security detection mechanisms. Since cybersecurity threats continuously evolve, general-purpose LLMs also suffer from a lack of real-time threat adaptation—without frequent updates and fine-tuning, they quickly become outdated and ineffective at recognizing emerging attack patterns. Lastly, the scalability and computational overhead of deploying large LLMs in security workflows can be impractical due to high resource consumption, making it essential to explore efficient fine-tuning techniques that optimize model performance while maintaining cost-effectiveness. Addressing these challenges requires targeted fine-tuning approaches that enhance an LLM’s ability to understand, detect, and mitigate domain-specific threats effectively.

In practical applications, LLMs have been employed in various industries, including customer service, healthcare, education, and entertainment. They assist in automating repetitive tasks, providing personalized experiences, and enhancing human creativity. As the field of natural language processing continues to evolve, LLMs are expected to become even more sophisticated, driving innovations across multiple domains.

## Fine Tune Large Language Model (LLM)

Fine-tuning a Large Language Model (LLM) involves adapting a pre-trained model to perform specific tasks or to better align with a particular domain or set of re-

quirements. This process leverages the vast amount of knowledge and linguistic capabilities that the model has already acquired during its initial training phase on a broad dataset. Fine-tuning typically involves training the model further on a more specialized dataset, which can be much smaller and more focused than the original training data. This specialized dataset contains examples relevant to the specific task or domain, enabling the model to learn the nuances and context-specific information required for improved performance in that area.

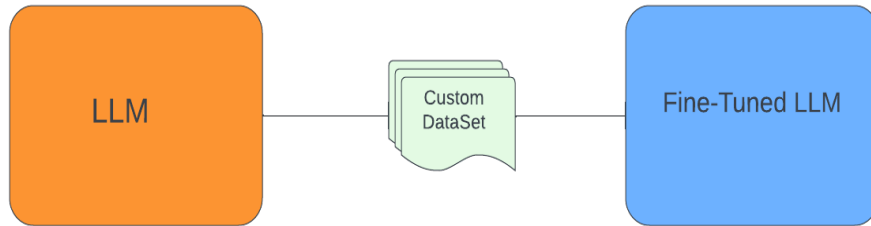


Figure 1.2: Fine Tune Large Language Model (LLM) [9]

The fine-tuning process is crucial for enhancing the model’s applicability and effectiveness in real-world scenarios. By tailoring the LLM to specific tasks such as sentiment analysis, question-answering, text summarization, or domain-specific applications like legal or medical document processing, organizations can achieve more accurate and relevant outputs. Fine-tuning helps in mitigating some of the biases and inaccuracies that might be present in the general-purpose model, as the specialized data can guide the model towards more precise and contextually appropriate responses. Additionally, fine-tuning can improve the model’s efficiency, making it more suitable for deployment in production environments where computational resources and response times are critical factors.

- **Pre-trained LLM Models with Domain Knowledge Fine-tuning**

A significant study aimed at enhancing large language models (LLMs) within specific domains focused on domain-specific knowledge transfer across various languages. For instance, researchers fine-tuned the LLaMA model with Chinese medical knowledge, which resulted in better comprehension and performance in Chinese-language LLM applications. This research highlights the critical role of domain-specific tuning in optimizing LLMs for specialized fields [25].

Several advanced techniques have been explored to improve domain knowledge transfer and fine-tuning of LLMs. Methods such as soft fine-tuning, large-scale fine-grained categorization, and domain-specific transfer learning have been utilized to enhance LLM performance in specialized areas. Moreover, the LLaMA-adaptor approach has demonstrated efficient fine-tuning of language models with zero-init attention, aiding in better adaptability for domain-specific tasks [45].

To support LLM fine-tuning, various domain knowledge datasets have been developed. For example, the Alpaca dataset provides 52k instruction-following data

points using Self-Instruct techniques to encode specific instructions within conversations. The HealthCareMagic-100k dataset includes 1 00k real-world patient-physician conversations, offering valuable insights into authentic medical interactions. Additionally, the GenMedGPT-5k dataset increases dataset diversity with 5k synthetic conversations between patients and physicians.

### 1.1.2 Software Supply chains threats

Software supply chains threats have become a critical issue in cybersecurity, exploiting vulnerabilities within the interconnected network of software development and distribution. The complexity of modern software supply chains, involving numerous third-party components, libraries, and tools, expands the potential attack surface. Malicious actors infiltrate and compromise systems by embedding harmful code into trusted software updates or open-source dependencies [21]. High-profile incidents like the SolarWinds attack, which impacted government agencies and large enterprises by compromising a software build process [28], highlight the severe implications of these breaches, including data theft, financial loss, and operational disruptions [15].

Addressing these threats requires a holistic approach involving secure coding practices, thorough vetting of third-party components, continuous monitoring, and rigorous testing throughout development and deployment [11]. Collaboration among software developers, vendors, and end-users is essential. Developers must integrate robust security checks into their workflows, vendors should ensure product integrity with timely updates and patches, and end-users need to apply updates promptly and conduct regular security assessments. Emerging technologies like blockchain and artificial intelligence offer promising enhancements for software supply chains security, providing immutable tracking of software components and real-time threat detection. By embracing these innovative solutions and fostering a security-centric mindset, organizations can better defend against evolving software supply chains threats [1].

### Mitigations for Software Supply chains threats

Mitigating software supply chains threats is crucial for maintaining the security and integrity of software systems. One key approach is to implement strict security measures throughout the software development lifecycle. This includes adopting secure coding practices, conducting regular code reviews, and performing thorough testing to identify and fix vulnerabilities early. Additionally, using automated tools for continuous monitoring and vulnerability scanning can help detect and address potential threats in real-time. Ensuring that all third-party components and libraries used in the software are from trusted sources and regularly updated is also essential to minimize risks [11].

Collaboration among developers, and end-users is vital for effective threat mitigation. Developers should prioritize security in their workflows, incorporating best practices and adhering to industry standards. Vendors need to ensure the integrity and security of their products, providing timely updates and patches to address

known vulnerabilities. End-users should maintain an active role in applying updates promptly and conducting regular security assessments. Furthermore, adopting advanced technologies like blockchain for secure tracking of software components and artificial intelligence for detecting anomalies can enhance software supply chains security. By fostering a security-first mindset and leveraging these strategies, organizations can better protect themselves against software supply chains threats [1].

## 1.2 Motivation

In the evolving landscape of software engineering, the significance of securing software supply chains has grown exponentially. The interconnected nature of modern software systems exposes them to an array of vulnerabilities, from compromised third-party dependencies to malicious code injections. Incidents such as the SolarWinds and Kaseya breaches underscore the critical need for robust threat mitigation mechanisms that can adapt to an ever-changing threat landscape.

Traditional approaches to software supply chain security often rely on static rules and signature-based systems that struggle to keep pace with novel attack vectors. This inadequacy highlights the urgent need for intelligent, adaptive solutions capable of identifying and mitigating threats proactively. Large Language Models (LLMs) have demonstrated exceptional capabilities in natural language processing and contextual understanding, but their effectiveness in specialized domains like software supply chain security remains underexplored.

This research is driven by the potential to harness the power of LLMs to revolutionize how software engineers and security practitioners address supply chain vulnerabilities. By fine-tuning an LLM like Microsoft's Phi-2 for this specific domain, we aim to bridge the gap between general-purpose language understanding and domain-specific threat mitigation. This approach promises to provide actionable insights, improve decision-making, and ultimately enhance the resilience of software supply chains against sophisticated attacks.

## 1.3 Aim and Objectives

The primary aim of this research is to investigate the effectiveness of fine-tuning Large Language Models (LLMs) for the domain-specific task of software supply chain threat mitigation. By leveraging domain-specific datasets and advanced fine-tuning techniques, this study seeks to enhance the capability of LLMs to address unique challenges in software supply chain security.

### Objectives

- **Analyze Fine-Tuning Techniques:** Evaluate and implement advanced fine-tuning methodologies such as Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) to adapt LLMs for software supply chain security tasks.

- **Curate a Domain-Specific Dataset:** Develop a comprehensive dataset that encapsulates various facets of software supply chain threats, including incident reports, best practices, and mitigation strategies.
- **Model Performance Assessment:** Compare the performance of the fine-tuned Phi-2 model against its baseline version using robust evaluation metrics such as ROUGE, BERTScore, and BLEURT, supplemented by qualitative assessments from domain experts.
- **Demonstrate Practical Applicability:** Validate the fine-tuned model's practical utility through real-world case studies and feedback from cybersecurity professionals, ensuring its relevance to industry needs.
- **Contribute to Software Engineering Practices:** Provide actionable insights and methodologies for integrating LLM-based solutions into the software development lifecycle, emphasizing proactive security measures.

By achieving these objectives, this research aims to contribute significantly to the fields of software engineering and cybersecurity, offering a novel approach to mitigating risks in software supply chains through intelligent, AI-driven solutions.

## 1.4 Scope of the Research

This research focuses on the application of fine-tuned Large Language Models (LLMs) to the domain of software supply chain security. The scope is defined by the following key areas:

- **Fine-Tuning Techniques:** The study emphasizes advanced fine-tuning methodologies, particularly Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA), to optimize LLMs for domain-specific tasks.
- **Software Supply Chain Threats:** The research is restricted to addressing vulnerabilities, risks, and mitigation strategies within software supply chains, including third-party dependency management and open-source component security.
- **Evaluation Metrics:** Performance evaluation is based on both quantitative metrics (e.g., ROUGE, BERTScore, BLEURT) and qualitative feedback from cybersecurity professionals.
- **Real-World Relevance:** Practical applicability is validated through case studies and professional insights, ensuring the solutions proposed align with industry requirements.

This defined scope ensures the research remains focused, actionable, and relevant to the challenges faced by software engineers and security practitioners.

## 1.5 Outline

This thesis consists of seven sections: Introduction, Related Work, Method, Implementation, Results and Analysis, Discussion, and Conclusions and Future Work. Additional information is presented in the Appendices.

- **Section 1** 'Introduction,' provides the background, motivation, aims, objectives, scope, and research gap.
- **Section 2** 'Related Work,' reviews existing research on LLM fine-tuning and software supply chain threats.
- **Section 3** 'Method,' outlines the proposed methodology.
- **Section 4** 'Implementation,' describes the practical implementation using Parameter Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA).
- **Section 5** 'Results and Analysis,' presents the findings from applying the methodology and analyzes the effectiveness of different supply chain threat mitigation approaches.
- **Section 6** 'Discussion,' analyzes the results and addresses validity threats and limitations. Finally.
- **Section 7** 'Conclusions and Future Work,' summarizes the research contributions and outlines potential avenues for future research.

J Howard et al. [17] introduced Universal Language Model Fine-tuning (ULMFiT), a transfer learning method for NLP tasks that significantly improves upon existing approaches. They developed novel fine-tuning techniques to prevent catastrophic forgetting and ensure robust learning across various tasks. ULMFiT outperformed state-of-the-art methods in six text classification tasks, achieving error reductions of 18-24% on most datasets. Remarkably, even with just 100 labeled examples, ULMFiT matched the performance of models trained from scratch on datasets 100 times larger. The authors open-sourced their pretrained models and code, aiming to advance transfer learning developments in NLP through their findings.

N Ding et al. [10] explored "delta-tuning," a method for parameter-efficient adaptation of pre-trained language models (PLMs) that focuses on optimizing a small portion of the model's parameters while keeping the rest fixed. They categorized various delta-tuning methods, including techniques such as Low-Rank Adaptation (LoRA), Prefix Tuning, and Adapter Modules, and analyzed their theoretical principles from optimization and optimal control perspectives. They conducted a comprehensive empirical study on over 100 natural language processing tasks, evaluating the performance of these delta-tuning techniques. The results demonstrated that delta-tuning significantly reduces computational and storage costs while maintaining or enhancing the performance of large-scale PLMs. This approach proved effective in making the adaptation of PLMs more practical and efficient.

S Wang et al. [46] reviewed the entire software supply chains of Large Language Models (LLMs) and Large Multimodal Models (LMMs), focusing on their development, lifecycle stages, and integration into applications. It discussed core elements such as model infrastructure, lifecycle processes from training to ongoing maintenance, and the ecosystem for deploying these models in various domains. Challenges highlighted included data privacy, model interpretability, scalability of infrastructure, and regulatory compliance. The paper proposed future research directions to address these challenges and ensure responsible deployment of LLMs, aiming to advance their ethical use and technological development in artificial intelligence applications.

B Alt et al. [3] created a natural-language-based programming assistant to help with complex industrial robot programming by offering interactive dialogues about skills, example use-cases, and expected robot behavior. They trained three families of Large Language Models (LLMs) using different domain-specific fine-tuning methods. The models were evaluated using a semantic similarity metric and a user survey with industry experts. The results showed that while domain-specific fine-tuning achieved

robust domain transfer, it was not yet practical due to issues like hallucinations, repetition, and language switching. They suggested that prompting strategies could be promising alternatives and emphasized the need for larger-scale user surveys to better assess real-world performance.

T Singla et al. [41] evaluated the ability of Large Language Models (LLMs), specifically GPT-3.5 and Bard, to analyze historical software supply chains breaches. They developed prompts for these models to categorize 69 security failures by type of compromise, intent, nature, and impact. GPT-3.5 achieved an average accuracy of 68% across these dimensions, while Bard achieved 58%. The study demonstrated that LLMs could effectively characterize software supply chains failures when source articles were detailed enough for human analysts to reach a consensus. However, the models still exhibited limitations and could not yet fully replace human analysts, suggesting the need for further improvement and broader evaluation in future research.

J Zheng et al. [52] developed LlamaIT, a method for domain-specific machine translation using Large Language Models (LLMs). They tackled challenges such as sensitivity to input examples, high training costs, and issues with translating rare words by implementing a prompt-oriented fine-tuning approach. This involved constructing a task-specific mixed-domain dataset and fine-tuning the LLM with LoRA (Low-Rank Adaptation), eliminating the need for input translation examples and reducing over-specialization. To enhance the LLM's capability for translating rare words, they incorporated domain-specific bilingual vocabulary into new prompts. They conducted extensive experiments on both publicly available and self-constructed datasets to evaluate their approach. The outcomes showed that LlamaIT significantly improved the LLM's domain-specific translation capabilities while preserving its zero-shot translation abilities. This approach offered a more efficient and effective way to adapt general-purpose LLMs for specific translation tasks.

O Zheng et al. [53] developed TrafficSafetyGPT, a model based on LLaMA, to improve performance in transportation safety tasks where traditional LLMs struggled. They fine-tuned this model using the TrafficSafety-2K dataset, which includes human-labeled data from government guides and instruction-output pairs generated by ChatGPT. This approach aimed to provide more accurate and reliable responses in the transportation safety domain. The results showed that TrafficSafetyGPT outperformed existing models in generating traffic safety knowledge, while also being efficient in computational resource use and training time. This work highlights the potential for enhancing LLMs in specialized fields like transportation safety.

Y Zhong et al. [54] addressed the need for automated methods in Construction Management Systems (CMS) by exploring the potential of Transformer architecture. They created the first CMS-specific corpora from academic papers and developed an end-to-end pipeline for pre-training and fine-tuning domain-specific language models. Using transfer learning, they pre-trained BERT and RoBERTa on these new corpora. The best-performing models were then fine-tuned and outperformed models pre-trained on general corpora. In text classification and named entity recognition tasks, domain-specific pre-training improved F1 scores by 5.9% and 8.5%, respectively.

These results show the effectiveness of domain-specific models in CMS and broader construction-related fields.

J Zheng et al. [52] developed LlamaIT for domain-specific machine translation, and O Zheng et al. [53] introduced TrafficSafetyGPT for transportation safety tasks, both demonstrating the efficacy of domain-specific fine-tuning. Y Zhong et al. [54] addressed automated methods in Construction Management Systems (CMS) by pre-training and fine-tuning domain-specific language models. Despite these advancements, the application of LLMs to the critical area of software supply chains threat mitigation remains unexplored.

Ohm et al. [32] conducted a detailed study of 174 software supply chain attacks (2010-2020), establishing a fundamental taxonomy of attack vectors and patterns. Their research revealed 41% of attacks targeted package managers while 28% focused on build systems. The authors developed a comprehensive categorization system for attack techniques, providing essential patterns for training machine learning models. Their analysis of real-world incidents offers valuable insights for creating detection strategies and has been extensively cited in subsequent security research. This work forms a crucial foundation for developing automated threat detection systems, including LLM-based approaches.

Ladisa et al. [22] proposed a systematic approach to categorizing and analyzing attacks on open-source software supply chains. Their research identified critical vulnerabilities in dependency management systems and proposed a framework for evaluating the effectiveness of various mitigation strategies. The study emphasized the importance of automated security checks and continuous monitoring throughout the software development lifecycle.

Ellison et al. [12] developed a comprehensive framework for evaluating and mitigating software supply chain security risks. Their research established key criteria for assessing the security posture of software suppliers and proposed strategies for integrating security considerations into the software acquisition process. The authors emphasized the importance of addressing security risks early in the development cycle and maintaining visibility throughout the supply chain.

Martínez and Durán [28] analyzed the SolarWinds attack as a case study of modern software supply chain threats. Their research highlighted the sophistication of contemporary attacks and the limitations of traditional security measures. The study proposed enhanced monitoring systems and integrity verification mechanisms as critical components of an effective defense strategy.

Ellison and Woody [13] investigated the integration of security practices into software development processes, focusing specifically on supply chain risk management. Their work established a framework for incorporating security considerations throughout the development lifecycle, emphasizing the need for systematic approaches to identifying and mitigating risks. The research demonstrated that early integration of security measures could significantly reduce the impact of supply chain attacks.

Akinsola and Akinde [2] presented strategies for enhancing software supply chain resilience through improved security practices and development lifecycle management. Their research proposed a comprehensive framework for identifying and mitigating security risks, emphasizing the importance of continuous monitoring and adaptive response mechanisms.

Adenekan et al. [1] explored strategies for protecting IT supply chains against cybersecurity threats, focusing on practical implementation approaches. Their research highlighted the importance of combining technical controls with organizational policies to create effective defense mechanisms against supply chain attacks.

## 2.1 Research Gap

Despite advancements in the development and fine-tuning of Large Language Models (LLMs), their application to software supply chain threat mitigation remains underexplored. Current research has predominantly concentrated on general-purpose language understanding or domain-specific tasks unrelated to software engineering. Consequently, existing LLMs often lack the precision and contextual awareness needed to address the nuanced challenges of software supply chain security.

This gap is particularly evident in:

- **Specialized Knowledge:** General-purpose LLMs fail to adequately address domain-specific issues such as dependency vulnerabilities, malicious code injection, and build process integrity.
- **Performance Limitations:** Without domain-specific fine-tuning, LLMs tend to produce generic or irrelevant responses, limiting their utility in critical security contexts.
- **Integration Challenges:** There is limited guidance on how to effectively incorporate fine-tuned LLMs into the software development lifecycle to enhance security practices.

Addressing this research gap is vital for advancing the use of AI-driven solutions in software engineering and cybersecurity, ensuring that LLMs can provide precise, actionable, and reliable insights for mitigating supply chain threats.

In this chapter, we will outline the approaches we took to answering the research questions and discuss the reasons behind our decisions. In order to answer each research question we conduct a literature review; however, for research questions RQ1 and RQ2 we employ an experimental methodology.

### 3.1 Research Questions

**RQ1: Which fine-tuning algorithms are most effective for adapting Large Language Models (LLMs) to specialized topics such as software supply chains threat mitigations?**

**Motivation:** Determining the most effective fine-tuning algorithms for adapting Large Language Models (LLMs) to specialized topics such as software supply chains threat mitigations is crucial for enhancing their accuracy and relevance in these areas. General-purpose LLMs often fall short when addressing specific domains, highlighting the need for targeted fine-tuning. To answer this research question, we will perform a literature review to identify and evaluate various fine-tuning algorithms used in similar contexts. This will enable us to pinpoint the most effective methods for refining LLMs to better handle specialized topics, ultimately improving their practical application and decision-making capabilities.

**RQ2: How does the performance of a fine-tuned LLM on software supply chain threat mitigations compare to its base model, and what implications does this have for advancements in the software supply chain domain?**

**Motivation:** This research evaluates the effectiveness of domain-specific fine-tuning by comparing Microsoft's Phi-2 in its fine-tuned state with its base model for addressing software supply chain threats. The study focuses on quantifying performance improvements and exploring practical contributions to security practices and resilience in the software supply chain domain.

A systematic approach is employed, using quantitative metrics such as ROUGE, BERTScore, and BLEURT to measure semantic accuracy, relevance, and fluency. Standardized domain-specific queries, including scenarios like vulnerability detection, mitigation planning, and threat intelligence synthesis, are used to assess the capabilities of both models. Additionally, qualitative assessments by domain security experts provide insights into the relevance, accuracy, and practical applicability

of model outputs. Experts are also consulted to explore how the fine-tuned model can enhance software development lifecycle (SDLC) processes, particularly in improving vulnerability detection and mitigation.

## 3.2 Literature Review

Conducting a comprehensive literature review is essential for understanding and establishing a solid foundation for future investigation in any study. This involves gathering relevant information from current research. The first research question, RQ1, focuses on evaluating fine-tuning algorithms for adapting Large Language Models (LLMs) to specialized topics, such as software supply chains threat mitigations. The literature review is instrumental in identifying various approaches used in this field, addressing key issues, and informing the direction of future research.

To achieve this, a thorough search strategy was implemented to locate relevant research articles from academic databases such as Google Scholar, ScienceDirect, IEEE Xplore, and ResearchGate. A snowballing strategy was employed to collect sources and create the literature review. This involves using a group of relevant papers to run both forward and backward searches, uncovering more appropriate sources.

To ensure a comprehensive and relevant selection of papers, we used the following search strings:

- "Fine-Tuning" AND "Large Language Models"
- "Domain-Specific" AND "Large Language Models"
- "Large Language Models" AND "Software Supply Chain Security"
- "Transfer Learning" AND "Cybersecurity" AND "LLMs"
- "Parameter-Efficient Fine-Tuning" OR "PEFT" AND "LLMs"
- "Low-Rank Adaptation" OR "LoRA"

Papers were selected based on references to topics such as fine-tuning algorithms for LLMs, performance comparisons of specialized versus general-purpose models, and domain-specific fine-tuning techniques. The criteria for classifying the work as relevant included their focus on machine learning techniques for specialized topic adaptation, transfer learning methods, and hybrid deep learning models.

By systematically gathering and analyzing the existing literature, this review aims to provide a comprehensive understanding of the current state of research on fine-tuning LLMs for specialized topics. It will highlight the strengths and weaknesses of various approaches, identify gaps in the current research, and suggest directions for future investigation. As this literature review is a crucial methodology of this study, providing a thorough overview of existing research on fine-tuning algorithms for

LLMs in the context of software supply chains threat mitigations. By implementing a robust search strategy and employing the snowballing technique, the review ensures a comprehensive and relevant collection of sources, laying the groundwork for future research and advancements in this field. The criteria for classifying the work as relevant are as follows:

**Inclusion criteria:**

- Works closely related to current research and distributed through conferences, journals, and books were recognized as relevant contributions.
- To keep the evaluation up to date with recent technical breakthroughs, only items published after 2016 were evaluated.
- The associated work is available in BIBtext format.

**Exclusion criteria:**

- Any works that did not have full-text availability were excluded from consideration.
- The articles which not published in English were also excluded from the review process.

## 3.3 Dataset

To fine-tune our language model for software supply chains threat mitigation, we compiled a comprehensive dataset that encompasses a variety of data sources relevant to the domain. This dataset includes both publicly available information and synthetically generated data to ensure broad coverage of possible scenarios and threats.

### 3.3.1 Data Collection

We started by gathering detailed incident reports from renowned cyber security databases such as the National Vulnerability Database (NVD) [31] and MITRE ATT&CK [6]. These incident reports provide real-world examples of software supply chains attacks, including attack vectors, methods of exploitation, and mitigation strategies. The inclusion of these reports ensures that our model is trained on authentic and varied instances of software supply chains threats.

In addition to incident reports, we incorporated whitepapers and guidelines from leading cybersecurity organizations like the National Institute of Standards and Technology (NIST) and the Open Web Application Security Project (OWASP). These documents offer in-depth analysis and recommended practices for managing software supply chains risks, providing a robust theoretical foundation for our dataset.

Furthermore, we utilized threat intelligence reports from prominent cybersecurity firms such as Symantec, CrowdStrike, and FireEye. These reports often include up-to-date information on emerging threats, detailed case studies of recent attacks, and expert analyses of threat trends. By incorporating these sources, our dataset remains current and relevant, reflecting the latest developments in the field of software supply chains security.

### 3.3.2 Data Cleaning and Preprocessing

To ensure the quality and consistency of our dataset, we undertook extensive data cleaning and preprocessing steps. First, text data from different sources were normalized to a common format, ensuring consistency in terminology and structure. This normalization process is crucial for reducing discrepancies and enhancing the coherence of the dataset.

We utilized Byte Pair Encoding (BPE) to tokenize the data, enabling efficient handling of the vocabulary. Tokenization is a critical step to ensure that the model can process and understand the input text effectively. This process allows the input to be broken into manageable units, facilitating better comprehension and analysis by the model.

Furthermore, each data entry was meticulously labeled with relevant tags to enhance supervised learning. These tags included instruction (typically representing user inputs), response (denoting the model's answer), and type (categorized as critical, major, or minor). This labeling was essential for the model to learn associations between various components, particularly in the context of software supply chain threats. By integrating this structured approach, we ensured a robust training pipeline tailored to addressing software security challenges.

### 3.3.3 Data Augmentation

To increase the robustness of our dataset, we applied various data augmentation techniques. One such technique was paraphrasing, where we used natural language processing (NLP) techniques to generate multiple paraphrases of the same text. This introduces variability and reduces overfitting, ensuring that the model does not become too dependent on specific phrasings.

Another augmentation technique involved shuffling sentences within paragraphs to create different contextual flows while preserving the overall meaning. This approach helps the model to generalize better by exposing it to varied sentence structures and contexts.

### 3.3.4 Dataset Statistics

Our final dataset comprised a diverse mix of real-world and synthetic data, ensuring comprehensive coverage of software supply chains threats. The dataset includes 10,000 rows gathered from incident reports, whitepapers and guidelines, threat intel-

ligence reports , This extensive dataset provides a solid foundation for training our language model.

To facilitate the training process, the dataset was divided into training and test sets. The training set was used to fine-tune the model, while a separate set of questions was utilized to evaluate the model's performance. This approach ensures that the model is trained and tested on distinct data, promoting generalizability and robustness.

### 3.3.5 Ethical Considerations

We ensured that all data collected and used for this research adhered to ethical standards and privacy regulations. Publicly available data was used in accordance with their usage policies, and synthetic data generation was done to supplement and not replace real-world data. We took care to exclude any sensitive or personally identifiable information from our dataset.

Our approach to data collection and processing reflects our commitment to ethical research practices and data integrity. By compiling and processing this extensive dataset, we aim to equip our fine-tuned language model with the knowledge necessary to effectively identify and mitigate software supply chains threats. This dataset serves as the cornerstone of our research, enabling us to train a model that is both accurate and reliable in its threat mitigation capabilities.

## 3.4 Interview

To investigate Research Question 2, this study employed semi-structured interviews as the primary data collection method. Semi-structured interviews were selected for their inherent flexibility in exploring participant perspectives while maintaining consistent coverage of key topics across all respondents. This methodology enables researchers to probe deeper into unexpected responses while ensuring standardization of core inquiries [7].

### 3.4.1 Sampling

The study utilized a combination of non-probability sampling techniques to identify and recruit participants with significant expertise in software supply chain security. Two primary sampling methods were employed:

**Expert Sampling:** This method was chosen to identify participants with substantial domain knowledge and practical experience in software supply chain security. The selection criteria focused on professionals with diverse roles within the security domain, including:

- Security Architect
- DevSecOps Engineers
- Supply Chain Security Specialists

- Application Security Engineers

**Snowball Sampling:** To expand the participant pool, initial interviewees were asked to recommend additional security professionals with relevant expertise. This approach helped identify practitioners who could offer valuable insights into software supply chain security practices and AI integration challenges [50].

The combination of these sampling methods ensured a comprehensive representation of perspectives across different security roles and experience levels, enhancing the validity and reliability of the research findings.

### 3.4.2 Data Collection

The interview implementation followed a structured protocol to ensure consistency and reliability:

**Questionnaire Development:** Created a standardized set of questions addressing specific research objectives. Incorporated both technical and practical aspects of software supply chain security. Designed questions to evaluate participants' experience with AI/ML tools. Included scenarios for assessing the fine-tuned model's effectiveness

**Interview Protocol:** Conducted preliminary test interviews to validate question clarity and effectiveness. Refined questions based on feedback from academic advisors. Implemented standardized procedures for interview execution. Obtained informed consent for recording and data usage.

**Execution:** Conducted interviews via zoom platform. Recorded sessions with participant permission. Maintained consistent interview duration (20-30 minutes). Ensured uniform presentation of the fine-tuned model demonstration

**Data Processing:** Transcribed interviews using specialized software. Implemented dual storage for data redundancy. Conducted systematic review of transcripts for accuracy. Applied standardized coding procedures for analysis

This methodological approach ensures reproducibility while maintaining high standards of academic rigor in data collection and analysis [29].

### 3.4.3 Data Analysis

We used thematic analysis as our main method for analyzing the interview data, following established research protocols [4]. This method helps researchers identify and analyze patterns in qualitative data, making it particularly valuable for software engineering research [38]. We chose this approach because it allowed us to systematically examine how practitioners view and implement LLM solutions in software supply chain security.

Our analysis began with a thorough review of all six interview transcripts. We read through each transcript multiple times to gain a deep understanding of the

practitioners' experiences and observations regarding software supply chain security and LLM implementation. This initial review helped us identify early patterns and potential themes, following standard practices in software engineering research [43].

Next, we implemented a detailed coding process that combined two approaches: looking for expected patterns (deductive coding) and discovering new insights from the data (inductive coding) [36].

This approach helped us capture both anticipated findings about software supply chain security and unexpected insights about LLM applications. Through this process, we identified five main themes that emerged from the data: Role and Experience, Evolution and Nature of Threats, Effectiveness and Application of Fine-Tuned LLMs, Integration Challenges and Recommendations for Improvement, and Future Implications for software supply chain Security Practices. We then refined these themes through careful review and validation [30]. Each main theme was broken down into smaller sub-themes, creating a structured framework that accurately represented the practitioners' views on implementing LLMs in software supply chain security.

To ensure accuracy, we continuously compared our themes against the original interview data, making adjustments where needed to better reflect the participants' perspectives [16]. The final step involved combining all these refined themes into a clear, coherent narrative. We focused on presenting the findings in a way that would be valuable for software engineering practitioners while maintaining research rigor. This resulted in a comprehensive framework that shows how professional experience, technical implementation, and future developments connect in the context of LLM fine-tuning for software supply chain security.

## 3.5 Experiment

### 3.5.1 Experimental Design

The experimental design is structured to enable a rigorous comparative analysis of two model versions:

- **Base Model:** The pre-trained Phi-2 model serves as the control group, representing the standard capabilities of the LLM without any domain-specific optimization.
- **Fine-Tuned Model:** The Phi-2 model fine-tuned using a curated dataset specific to software supply chain threat mitigation. This fine-tuning process aims to enhance the model's domain-specific knowledge and response capabilities.

This experimental setup directly evaluates the impact of domain-specific fine-tuning on model performance, leveraging proven methodologies in recent LLM studies.

### 3.5.2 Model Preparation

The study involves preparing two distinct versions of the Phi-2 model:

- **Base Model:** This is the unaltered pre-trained Phi-2 model, which acts as a baseline to evaluate the impact of domain-specific fine-tuning.
- **Fine-Tuned Model:** The Phi-2 model fine-tuned on a dataset specifically tailored to address software supply chain threats, providing a more focused and context-aware version.

### 3.5.3 Data Collection and Preparation

A comprehensive dataset was compiled from reputable sources within the software supply chain security domain. This dataset is divided into two key subsets:

- **Fine-Tuning Dataset:** Used exclusively to fine-tune the Phi-2 model, this dataset includes domain-specific content focused on software supply chain vulnerabilities and mitigation strategies.
- **Test Dataset:** A separate dataset consisting of domain-specific test queries and scenarios, designed to evaluate the performance of both the base and fine-tuned models objectively.

### 3.5.4 Evaluation Metrics

We evaluate the proficiency of generated texts in software supply chains threat mitigation with domain questions of different kinds, including definition, inclusion, exclusion, category, example, and guidance. The generated answers to these questions are evaluated by several popular and reliable quantitative metrics, examining the results from coverage, fluency, and brevity. For coverage, we selected ROUGE and BERTScore. For fluency, we chose BLEURT.

#### ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

It is an algorithm used to assess text summarization and machine translation. This metric can evaluate the information coverage of automatically produced texts compared to a reference passage, usually a human-written summary. ROUGE relies on n-gram modeling, which considers word overlap between generated and reference texts. The common ROUGE scores include ROUGE-1 (overlap of unigrams), ROUGE-2 (overlap of bigrams), and ROUGE-L (longest common subsequence-based statistics). Since ROUGE was proposed to solve a summarization problem, it is very effective and reliable for assessing information coverage. However, it can also result in bias due to a lack of semantic meaning understanding [24]. Therefore, like BLEU, a high ROUGE score may correspond to a sentence with a different meaning.

## BERTScore

It is a recently proposed metric based on the large language model BERT. It relies on BERT's prediction results to evaluate candidate sentences and reference sentences by tokens. BERTScore computes token similarity with context embeddings, replacing exact word matches. This evaluation focuses more on comparison in a vectorized embedding space instead of a common letter system, making it more suitable for large language model evaluation. Since BERT is developed using mask modeling, its word embeddings can reflect mask-filling accuracy and help effectively assess information coverage. However, one significant weakness is language model dependency, where model bias may be introduced in the evaluation process [51].

## BLEURT (Bilingual Evaluation Understudy with Representations from Transformer)

It is used to evaluate the fluency of generated texts. This evaluation metric is developed based on BERT, derived from the outputs of a robust BERT model pre-trained on perturbed synthetic sentences from Wikipedia. This metric can reflect the paraphrase level and language fluency. Although it relies on a large language model and may induce model bias, its fluency assessment is more reliable due to pretraining on a large set of synthetic data [37].

### 3.5.5 Experimental Procedure

The experimental process involves:

1. **Question Presentation:** Identical sets of test queries related to software supply chain security are presented to both the base and fine-tuned models.
2. **Response Generation:** Each model generates responses to the queries, which are recorded for evaluation.
3. **Automated Evaluation:** Responses are analyzed using quantitative metrics (BERTScore, ROUGE, BLEURT).
4. **Comparative Analysis:** Performance differences between the base and fine-tuned models are systematically evaluated across all metrics, complemented by manual analysis.

### 3.5.6 Summary

This methodological framework ensures a comprehensive evaluation of the effectiveness of domain-specific fine-tuning. By assessment, the research provides a nuanced understanding of the benefits and limitations of fine-tuning LLMs in addressing software supply chain threats. This approach is tailored to meet the specific challenges of the domain, offering actionable insights for future advancements in software engineering and cybersecurity.

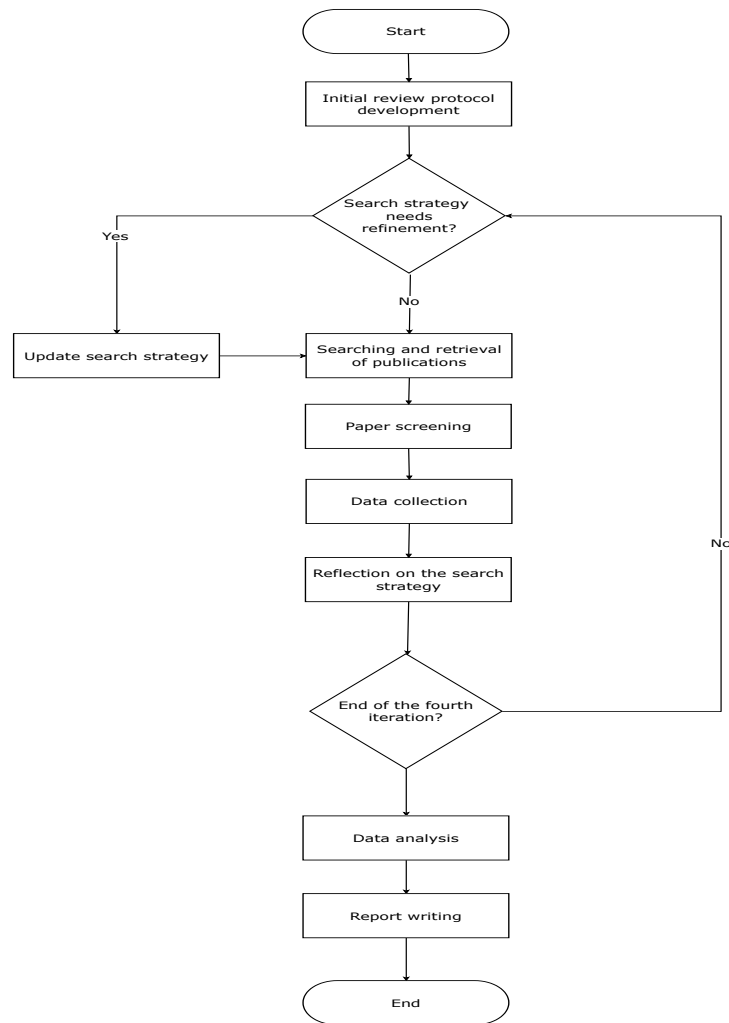


Figure 3.1: Literature review architecture

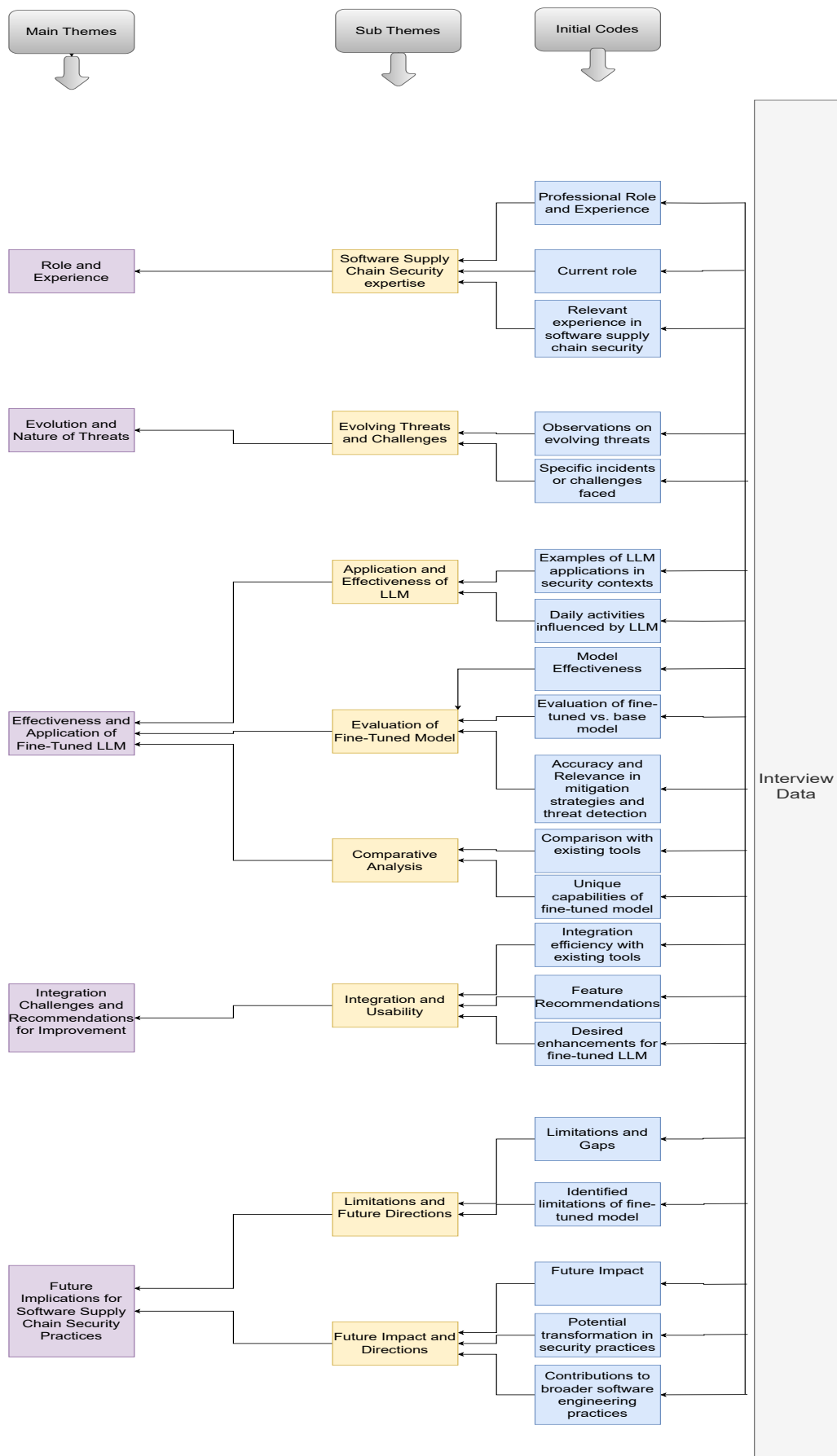


Figure 3.2: Thematic Analysis

In our implementation of fine-tuning, we adapted a pre-trained model to perform specific tasks or handle particular datasets. Traditional fine-tuning, which involves updating all model parameters, was recognized as resource-intensive and impractical for large models. To overcome these challenges, we employed advanced techniques such as Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA), which significantly enhanced the efficiency of the fine-tuning process.

### 4.1 Low-Rank Adaptation (LoRA)

#### Concept of LoRA

LoRA is designed to reduce the number of trainable parameters during fine-tuning by introducing low-rank matrices into the model's layers. The key idea is to decompose the parameter matrix  $W$  into two smaller matrices  $A$  and  $B$  such that:

$$W \approx W + A \cdot B \tag{4.1}$$

where  $A$  and  $B$  are low-rank matrices. This approximation significantly reduces the number of parameters that need to be updated.

#### Efficiency

By reducing the dimensions of  $A$  and  $B$  compared to the original matrix  $W$ , LoRA decreases the total number of trainable parameters. This reduction leads to faster training times and lower memory usage, making it feasible to fine-tune very large models.

#### Implementation of LoRA

- **Choosing the Rank:** Select a rank  $r$  that defines the dimensions of matrices  $A$  and  $B$ . A smaller rank results in fewer parameters but may reduce the model's ability to capture complex patterns.
- **Matrix Multiplication:** During forward propagation, the model approximates the parameter matrix  $W$  as  $W + A \cdot B$ . The matrices  $A$  and  $B$  are updated during training, while  $W$  remains fixed.

### 4.1.1 Parameter-Efficient Fine-Tuning (PEFT)

#### Principles of PEFT

PEFT is a broader framework that includes methods like LoRA to efficiently fine-tune models. PEFT aims to minimize the number of parameters updated during training while maintaining model performance.

- **Selective Parameter Update:** Instead of updating all model parameters, PEFT focuses on updating only a subset or introducing additional parameters that are easier to train.
- **Adaptation Layers:** PEFT may introduce new layers or adapters into the model. These adaptation layers can be trained while keeping the original model's parameters fixed or partially fixed.

The selection of Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) for fine-tuning the Phi-2 model was driven by their efficiency, scalability, and ability to preserve pre-trained knowledge while adapting to domain-specific requirements. Traditional full fine-tuning methods require updating all model parameters, which is computationally expensive and impractical for large models. In contrast, LoRA reduces the number of trainable parameters by introducing low-rank matrices, significantly lowering memory consumption and accelerating the training process. Similarly, PEFT allows fine-tuning with fewer resources while maintaining strong performance, making it particularly suitable for cybersecurity applications that require frequent updates to threat detection models.

Given the rapidly evolving nature of cybersecurity threats, scalability is a crucial factor—LoRA enables quick adaptation without the need for extensive retraining, and PEFT strategies such as Adapter Layers and QLoRA further optimize fine-tuning within memory-constrained environments. Another advantage of these methods is their ability to preserve pre-trained knowledge, avoiding the risk of catastrophic forgetting that often accompanies full fine-tuning. LoRA selectively fine-tunes specific layers, ensuring that the model retains its general linguistic capabilities while incorporating cybersecurity-specific insights. Compared to other fine-tuning techniques, full fine-tuning is computationally prohibitive, adapter layers are efficient but may struggle with highly dynamic security contexts, and knowledge distillation reduces model size but risks losing important domain-specific knowledge. In contrast, PEFT with LoRA provide an optimal balance between efficiency, domain adaptation, and model retention, making them the preferred approach for fine-tuning the Phi-2 model for cybersecurity applications.

#### Implementing PEFT with LoRA

- **Adapter Layers:** Insert LoRA layers into the transformer architecture. These layers are placed in specific locations (e.g., attention layers) where they can effectively adapt the model to new tasks.

- **Gradient Checkpointing:** To reduce memory usage during training, gradient checkpointing is used. This technique involves storing only some intermediate activations and recomputing others as needed during backpropagation.

## 4.2 Practical Implementation in a Notebook

This section outlines our detailed approach to implementing fine-tuning using Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) in a notebook environment. We cover the steps involved, including setting up the environment, loading and preparing the dataset, configuring the model and LoRA, fine-tuning, and evaluating the model's performance.

### Setting Up the Environment

We began our implementation by configuring the notebook environment with the necessary libraries and dependencies. We chose platforms such as Google Colab or Kaggle, which offer free GPU access to expedite the training process.

We then installed the required libraries using pip commands. The libraries we installed include:

- **transformers:** Provides tools for loading pre-trained models and tokenizers, essential for working with advanced language models.
- **datasets:** Facilitates access to various datasets and their preprocessing.
- **bitsandbytes:** Offers utilities for model quantization, reducing memory usage and computational costs.
- **peft:** Includes methods for parameter-efficient fine-tuning, such as LoRA, enabling efficient adaptation of large models.

By installing these libraries, we ensured that all necessary tools for model loading, tokenization, and training were available.

### Loading the Dataset

With the environment set up, we proceeded to load and inspect our dataset. We used a diverse dataset that included real-world incident reports, whitepapers, threat intelligence reports, and synthetic data generated to cover various software supply chains scenarios.

To facilitate the loading process, we utilized the HuggingFace `datasets` library. The following code snippet was employed to load the dataset:

```
from datasets import load_dataset
dataset = load_dataset("software_supply_chain_threats.csv")
```



Apply this function to the entire dataset to obtain tokenized sequences:

```
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

Tokenization maps a text sequence  $x$  into a sequence of token IDs  $t$ :

$$t = \text{Tokenizer}(x) \quad (4.2)$$

After tokenization, you need to prepare the input and label sequences for the model. This involves removing unnecessary columns from the tokenized dataset and setting up the labels for training:

```
model_inputs = tokenized_datasets['train'].remove_columns(['summary'])
model_inputs['labels'] = tokenized_datasets['train']['summary']
```

## Pre-Processing the Dataset

Pre-processing ensures that the dataset is correctly formatted and ready for training. This step involves setting up data collators and data loaders.

A data collator is responsible for dynamically padding sequences during training, which helps in managing batch sizes and input lengths. Use the `DataCollatorForSeq2Seq` from the `transformers` library:

```
from transformers import DataCollatorForSeq2Seq

data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

Next, create a `DataLoader` to handle batching and shuffling of the data:

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(model_inputs, batch_size=8,
                               collate_fn=data_collator)
```

The `DataLoader` prepares batches of tokenized input sequences  $X$  and corresponding output sequences  $Y$  for training.

## Configuring PEFT with LoRA

Configuring PEFT with LoRA involves setting up the LoRA-specific parameters and integrating them into the model.

First, define the LoRA configuration, which includes specifying the rank  $r$ . The rank determines the dimensions of the low-rank matrices  $A$  and  $B$ :

```
from peft import LoraConfig, apply_lora

lora_config = LoraConfig(rank=8)
model = apply_lora(model, config=lora_config)
```

LoRA introduces low-rank matrices such that:

$$W_{\text{adapted}} = W + A \cdot B \quad (4.3)$$

where  $W$  is the original weight matrix, and  $A$  and  $B$  are low-rank matrices with dimensions determined by  $r$ .

Next, integrate PEFT into the model training pipeline. Use the PEFT framework to manage parameter-efficient fine-tuning:

```
from peft import Trainer, TrainingArguments

training_args = TrainingArguments(
    per_device_train_batch_size=8,
    num_train_epochs=3,
    logging_dir='./logs',
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=model_inputs,
    data_collator=data_collator,
)
```

## Fine-Tuning the Model

Fine-tuning involves training the model on the prepared dataset using the LoRA-enhanced configuration. This step updates the model parameters to improve its performance on the specific task.

Run the training process using the Trainer class:

```
trainer.train()
```

During training, the model parameters are updated to minimize the loss function  $\mathcal{L}$ :

$$\mathcal{L}(\theta) = \sum_i \text{Loss}(y_i, \hat{y}_i) \quad (4.4)$$

where  $\theta$  represents the trainable parameters,  $y_i$  are the true summaries, and  $\hat{y}_i$  are the predicted summaries.

### 4.2.1 Conclusion

In this section, we have detailed the process of fine-tuning a pre-trained language model using PEFT with LoRA. We covered setting up the environment, loading and preparing the dataset, configuring the model and LoRA, fine-tuning, and evaluating the model using ROUGE, BERTScore, and BLEURT. By incorporating these metrics, we ensure a comprehensive assessment of the model's performance, providing valuable insights into its effectiveness in handling specific tasks with reduced computational resources and time.

## 5.1 Literature Review Results and Analysis

### 5.1.1 Literature Review

To answer related research questions, this study uses a literature review methodology. The primary goal of this study is to assess fine-tuning algorithms for adapting Large Language Models (LLMs) to specialized topics such as software supply chain threat mitigations, as outlined in RQ1. A literature review methodology was used to identify relevant studies on fine-tuning LLMs for software supply chain threat mitigation. The search process initially retrieved 93 research papers from databases such as IEEE Xplore, ScienceDirect, Google Scholar, and ResearchGate. After applying relevance filters based on title and abstract screening, this was narrowed down to 42 papers. Further refinement using backward and forward snowballing techniques resulted in a final selection of 27 key research articles for in-depth review. To achieve this goal, the study begins by reviewing existing research papers in this area to understand the different fine-tuning methods that have been used. A literature review is performed to identify and evaluate the algorithms that are most effective for this purpose. The results of this study, as presented in the research, bring together valuable insights gathered from various research sources, providing a strong foundation for the subsequent phases of the investigation. Table 5.1 summarizes the key papers reviewed and their main conclusions regarding fine-tuning algorithms.

Table 5.1: Summary of Literature Review Findings on Fine-tuning Algorithms

Paper Title	Conclusions Drawn
TrafficSafetyGPT: Tuning a Pre-trained Large Language Model to a Domain-Specific Expert in Transportation Safety [53].	The paper uses several fine-tuning algorithms to enhance large language models (LLMs) in the financial domain. LoRA (Low-Rank Adaptation) selectively updates weight matrices to improve efficiency. PEFT (Parameter-Efficient Fine-Tuning) adds and fine-tunes a small number of new parameters, balancing performance and cost. Adapter methods insert small neural networks into each layer, preserving pre-trained knowledge while adapting to new tasks. These algorithms are chosen for their ability to optimize performance, handle rapid market changes, and meet high accuracy and security requirements.
Fine-tuning Large Language Models for Domain-specific Machine Translation [18].	The paper proposes a prompt-oriented fine-tuning method called LlamaIT for domain-specific machine translation (MT). They use LoRA (Low-Rank Adaptation) for fine-tuning a general-purpose LLM (such as Llama2-7B) on a mix-domain dataset to enhance MT capabilities without needing in-context examples or post-processing. This method also preserves zero-shot MT capabilities. Additionally, they incorporate domain-specific bilingual vocabulary to improve the translation of rare words. The goal is to address the high training costs and over-specialization associated with traditional fine-tuning methods while maintaining efficiency and effectiveness.

Paper Title	Conclusions Drawn
Automating research synthesis with domain-specific large language model fine-tuning [44].	In the paper, fine-tuning is performed using two primary methods: LoRA (Low-Rank Adaptation) and NEFTune (Noise-Enhanced Fine-Tuning). LoRA is employed to adjust parameters efficiently and quickly, leveraging low-rank adaptations to enhance model performance without extensive computational resources. NEFTune introduces noise into embedding vectors, aimed at improving the model's generalization capabilities. These algorithms are chosen for their ability to enhance fine-tuning efficiency and performance, making them suitable for domain-specific tasks such as automating systematic literature reviews (SLR).
Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations [45].	The paper uses Parameter-Efficient Fine-Tuning (PEFT) methods, specifically LoRA and QLoRA, for adapting large language models. LoRA reduces trainable parameters by adding small, trainable submodules, enhancing efficiency and performance. QLoRA further reduces model size by quantizing weights to 4-bit precision, making it suitable for limited-memory environments. These methods are chosen for their efficiency in reducing computational demands and fine-tuning time while maintaining performance.
LoRA ensembles for large language model fine-tuning [47].	The paper proposes using LoRA (Low-Rank Adapters) ensembles for fine-tuning large language models (LLMs). This method addresses the challenges of storing and using multiple LLM copies for ensembling, which is impractical due to their large size. LoRA ensembles allow efficient parameter sharing by learning low-rank additive correction terms, which significantly reduce the number of trainable parameters. This approach provides the necessary randomness for variability across ensemble components, improving accuracy and uncertainty quantification. Additionally, LoRA ensembles are compatible with various regularization techniques, further enhancing model performance.

Paper Title	Conclusions Drawn
A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models [26].	The authors implemented SplitLoRA for fine-tuning large language models in a distributed setting. They divided the pre-trained model between client servers and a central server: clients handle initial Transformer layers, while the central server manages the rest. LoRA adapters were applied to both segments. During training, clients perform forward passes, send activations to the central server for completion, and receive gradient information for local back-propagation. Periodic aggregation of client-side adapters synchronizes the model. This method enables efficient parallel training, reducing computational and communication overhead, ideal for resource-constrained edge computing environments.
Small Language Models for Application Interactions: A Case Study [23]	Phi-2 performs exceptionally well for domain-specific fine-tuning due to its balance of size (2.7B parameters) and performance. It was preferred because it matches or exceeds models 25x larger in specialized tasks like planning and real-time data analysis. Its compact size also makes it easier to integrate with industry applications, reducing computation and storage costs.
Optimizing Phi-2: A Deep Dive into Fine-Tuning Small Language Models [49]	In this paper Phi-2 was chosen for fine-tuning due to its efficient memory usage and its adaptability through techniques like Low-Rank Adaptation (LoRA). These advancements allow Phi-2 to achieve high domain-specific performance while avoiding the computational overhead associated with larger models. The use of QLoRA enhances its ability to handle large datasets efficiently, which is beneficial for constrained environments. This optimization ensures Phi-2 remains a robust choice for targeted fine-tuning. The paper highlights its practical advantages in scenarios with limited computational resources. Consequently, Phi-2 stands out for its parameter-efficient training and versatility.

Paper Title	Conclusions Drawn
Fine-Tuning Microsoft Phi-2 for software supply chain Threat Mitigation [34]	Phi-2's design, tailored for tasks requiring deep understanding and pattern recognition, shows outstanding effectiveness in mitigating software supply chain threats. Its smaller size compared to GPT-based models, combined with superior performance in multi-step reasoning, math, and coding, makes it a compelling choice for fine-tuning. The reduced computational costs associated with Phi-2 enhance its scalability and practical application. Its performance in specific domains exemplifies its utility in addressing complex security challenges. The model's compact architecture proves advantageous for high-stakes, resource-intensive tasks. Overall, Phi-2's capabilities and efficiency make it a leading option for specialized applications.

### 5.1.2 Analysis from the literature review

In the paper [18], the authors delve into various fine-tuning techniques aimed at enhancing the efficiency and performance of large language models (LLMs). Among these, Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) are highlighted as key strategies. The paper discusses how LoRA selectively fine-tunes weight matrices within Transformer layers, reducing the overall computational demands while retaining strong performance. PEFT similarly enhances the model by adding a small number of new parameters to the pre-trained model, making it highly cost-effective and computationally feasible. This analysis directly motivates our decision to use Phi-2 with PEFT with LoRA techniques as these methods ensure the pre-trained model's integrity is preserved while being adapted to new tasks through linear transformations. This allows us to fine-tune the model efficiently for domain-specific applications like software supply chain security [49].

In [53], the use of PEFT with LoRA is further explored in a domain-specific context, with the paper highlighting their ability to optimize performance and balance computational demands. The selective updates to weight matrices with LoRA help retain the core capabilities of the pre-trained model while making targeted improvements for task-specific operations, such as handling safety protocols in transportation. By applying these methods, the authors demonstrate how the base model can be rapidly adapted to meet new requirements. This directly influences our choice of Microsoft Phi-2, as we aim to fine-tune the model specifically for cybersecurity applications, particularly in threat mitigation within software supply chains [34].

In [44], both LoRA and NEFTune (Noise-Enhanced Fine-Tuning) are used to optimize fine-tuning efficiency. LoRA's low-rank adaptations are employed to efficiently

adjust parameters without excessive resource usage, while NEFTune introduces noise to improve the model’s generalization capabilities. The flexibility and cost-effective nature of these methods for domain-specific tasks further underscore the practicality of using LoRA in our fine-tuning process for Phi-2. In software supply chain security, adaptability and efficiency are paramount, making LoRA an ideal choice [49].

Lastly, the paper [45] [47] provides a clear rationale for using PEFT with LoRA in enterprise-level applications. By reducing trainable parameters, LoRA and QLoRA offer a balance between performance and memory usage, ensuring that large models can be fine-tuned in environments with limited computational resources. Given the large datasets and high memory requirements typical in cybersecurity and threat analysis, LoRA’s ability to minimize computational overhead while maintaining model integrity strongly motivates its application in our Phi-2 fine-tuning [34] [49].

Together, these papers illustrate the efficiency, flexibility, and scalability of PEFT with LoRA, making them ideal choices for fine-tuning Phi-2. They enable the model to be optimized for domain-specific tasks like software supply chain threat mitigation while minimizing resource consumption and maintaining high performance.

## 5.2 Interview Results

### 5.2.1 Demographics

**Participant Demographics and Professional Background** Our research gathered detailed demographic information from six experienced practitioners in the field of software supply chain security. This data was essential for establishing the credibility of our findings and ensuring participants had sufficient expertise to provide meaningful insights into LLM implementation for security purposes.

Table 5.2: Participant Demographics

Role	Experience (Years)	Area of Expertise
DevSecOps Engineer	7	CI/CD Security, Build Process Security, Supply Chain Security
Security Architect	13	Supply Chain Risk Management
Security Analyst	6	Threat Detection, Vulnerability Management
Security Architect	11	Security Architecture, Risk Assessment
DevSecOps Engineer	4	Pipeline Security, Container Security
Application Security Engineer	4	Code Security, Dependency Management

The participants represented a balanced mix of technical leadership positions across different organizational contexts. Their average experience of 7.5 years in the field demonstrates significant exposure to software supply chain security challenges. This diverse participant pool provided comprehensive insights into both strategic and operational aspects of implementing LLM-based security solutions [42]. Key characteristics of the participant group included:

Direct involvement in software supply chain security. Experience with AI/ML tools in security contexts. Practical knowledge of threat mitigation strategies. Familiarity with modern development practices

The participants varied backgrounds and expertise levels enabled a thorough examination of the research questions from multiple perspectives, strengthening the validity of our findings [50]. Their collective experience spans the recent evolution of software supply chain security, including responses to major security incidents and the integration of AI-driven solutions.

## 5.2.2 Interview Summary and Findings

### Thematic Analysis

According to the Second Research Question, we have identified five key themes:

- Role and Experience
- Evolution and Nature of Threats
- Effectiveness and Application of Fine-Tuned LLMs
- Integration Challenges and Recommendations for Improvement
- Future Implications for software supply chain Security Practices

### Detailed Interview Analysis

#### Interviewee 1

**Role and Expertise** Leads the supply chain security practice with 13 years of experience in securing software development and deployment pipelines. Specializes in building architectures that defend against sophisticated threats, balancing security needs with operational efficiency.

**Evolving Threats** Over time, threats have evolved from malicious public repository packages to more complex attacks targeting build environments and signing infrastructure. Incidents have occurred where attackers maintained persistence for extended periods, emphasizing the need for advanced detection and response strategies.

**Effectiveness of Fine-Tuned LLMs** The fine-tuned LLM was notably more effective than general-purpose models in understanding supply chain-specific contexts. Its actionable recommendations, aligned with standards like SLSA, made it highly relevant for addressing complex security challenges.

**Limitations and Recommendations** Despite its strengths, the model struggles with newer, novel threats and interconnected scenarios. Suggested enhancements include deeper integration with software composition analysis tools and capabilities for mapping supply chain attack surfaces to improve its practicality.

**Future Impact** Fine-tuned LLMs are expected to play a significant role in proactive security measures, democratizing access to expertise for smaller teams. However, human oversight remains critical for refining and implementing recommendations.

### Interviewee 2

**Role and Expertise** With seven years of experience, leads DevSecOps practices focused on automating security processes, securing build pipelines, and managing third-party dependencies. Expertise lies in mitigating vulnerabilities across development workflows.

**Evolving Threats** Highlighted a shift from basic access controls to comprehensive pipeline security due to incidents like SolarWinds. These threats have prompted organizations to adopt practices such as reproducible builds, artifact signing, and dependency pinning to protect their supply chains.

**Effectiveness of Fine-Tuned LLMs** Commended the fine-tuned LLM for its ability to identify CI/CD pipeline risks and offer targeted controls. Its contextual recommendations aligned well with modern DevSecOps practices, making it suitable for real-world application.

**Limitations and Recommendations** Identified limitations in the model's handling of newer tools and platforms, as well as its guidance for high-velocity and cloud-native environments. Suggested improvements include enhanced analysis of pipeline configurations and automated policy generation.

**Future Impact** Fine-tuned LLMs are seen as critical for embedding security earlier in development cycles, particularly in fast-paced environments. By providing actionable guidance, these tools can help balance security with development velocity, though human expertise will always be essential for decision-making.

### Interviewee 3

**Role and Expertise** With six years of experience as a Senior Security Analyst, focuses on identifying and mitigating software supply chain risks. Responsibilities include managing third-party dependencies, monitoring ecosystems, and ensuring

secure development practices. Expertise lies in vulnerability assessment and incident response, particularly for modern supply chain challenges.

**Evolving Threats** Observed a significant evolution in threats, from simple outdated dependencies to sophisticated pipeline-wide attacks. The SolarWinds incident marked a turning point, prompting the adoption of comprehensive security measures to address these new challenges.

**Effectiveness of Fine-Tuned LLMs** Found the fine-tuned LLM promising for understanding supply chain security concepts. Its contextual relevance in areas like dependency management and artifact verification was impressive, but there is a need for more extensive testing across various scenarios to fully assess its effectiveness.

**Limitations and Comparisons** Compared to general-purpose LLMs, the fine-tuned model demonstrated a deeper understanding of supply chain-specific terminology and nuances. However, it occasionally relied on general advice rather than specific technical guidance, especially regarding newer tools and frameworks.

**Recommendations for Improvement** Suggested that the model could benefit from integrating knowledge of current vulnerability databases and attack patterns. Additionally, emphasized the importance of providing concrete remediation steps and enhancing integration with CI/CD platforms for practical application.

**Future Impact** Envisions fine-tuned LLMs as valuable for initial security assessments and team training, particularly in educating developers about supply chain security best practices. However, their role is expected to remain advisory, complementing human decision-making rather than replacing it.

#### Interviewee 4

**Role and Expertise** With 11 years in cybersecurity, oversees security architecture for a microservices ecosystem. Experience has transitioned from perimeter security to in-depth supply chain security, now a critical part of the responsibilities.

**Evolving Threats** Observed a shift toward threats embedded in development tools, with incidents like Log4j underscoring the need for rigorous inventory and verification processes. Emphasized the importance of viewing build systems as critical security assets.

**Effectiveness of Fine-Tuned LLMs** Found the fine-tuned LLM adept at understanding supply chain concepts, particularly build system security and dependency management. However, highlighted the need for further testing to evaluate its handling of complex enterprise scenarios.

**Limitations and Comparisons** The model demonstrated improvements over general-purpose LLMs but occasionally defaulted to generic advice in nuanced contexts. Viewed it as a complementary tool for training and assessments, not a replacement for established solutions like Snyk.

**Recommendations for Improvement** Suggested deeper integration with vulnerability databases and the ability to analyze dependency files directly. Also recommended providing tailored guidance for various development environments.

**Future Impact** Foresees these models becoming essential educational tools, bridging the gap between developers and security specialists. However, emphasized that their real impact depends on maturing the technology to handle real-world complexity and emerging threats.

### Interviewee 5

**Role and Expertise** With four years of experience as a Senior DevSecOps Engineer, focuses on integrating security into CI/CD pipelines, managing container security, and ensuring secure dependency management. The team's primary responsibility is securing build processes end-to-end, ensuring robustness against supply chain threats.

**Evolving Threats** Observed a dramatic increase in the complexity of software supply chain threats, especially with the adoption of microservices and container-based architectures. The Codecov breach highlighted vulnerabilities in build processes, prompting the team to enhance security protocols significantly.

**Effectiveness of Fine-Tuned LLMs** Found the fine-tuned LLM useful, particularly for container security and CI/CD pipeline protection. It provided relevant insights into DevOps-specific challenges, though there is a need for additional testing in complex, real-world scenarios.

**Limitations and Comparisons** While the fine-tuned model excelled in offering contextual suggestions, it did not match specialized tools like Snyk or Aqua Security for direct vulnerability detection. Valued its role as an educational aid but did not see it replacing dedicated security tools.

**Future Impact** Envisions fine-tuned LLMs becoming vital for developer training and early-stage security validation, aiding teams in identifying potential issues early. However, emphasized that these tools are not yet reliable enough for automated decisions in production environments.

### Interviewee 6

**Role and Expertise** Focuses on secure code review and vulnerability assessments, with four years of experience. Though has not dealt with major supply chain incidents directly, has implemented preventive measures such as dependency scanning and enhanced security controls in development workflows.

**Evolving Threats** Noted a shift from maintaining updated dependencies to addressing comprehensive threats highlighted by incidents like SolarWinds. These prompted stricter controls on third-party components and build processes within the organization.

**Effectiveness of Fine-Tuned LLMs** Appreciated the model's clarity in identifying common risks and providing detailed suggestions about dependency management. However, expressed the need to test its performance in complex, real-world scenarios involving modern frameworks.

**Limitations and Comparisons** While the fine-tuned LLM excelled at explaining dependency-related risks, it fell short in areas such as containerization and newer technologies. Viewed it as a learning and advisory tool rather than a replacement for specialized solutions like Snyk.

**Recommendations for Improvement** Suggested enhancements included direct integration with package managers, more specific guidance on security fixes, and improved capabilities for securing build tools.

**Future Impact** Sees potential in these models to democratize security knowledge, aiding teams without dedicated experts. However, believes they will complement rather than replace human expertise and specialized tools.

## 5.3 Experiment Results

### 5.3.1 Introduction

This chapter presents the evaluation results of the fine-tuned Microsoft Phi-2 model, which has been specifically adapted to address threats in software supply chains. Our objective was to evaluate the model's effectiveness in identifying potential risks and proposing mitigation strategies tailored to this domain. The evaluation compared the performance of our fine-tuned model against a baseline Microsoft Phi-2 model, focusing on improvements in accuracy, relevance, and practical utility in a software engineering context.

### 5.3.2 Overview of Model Performance

The fine-tuned Microsoft Phi-2 model was evaluated on its ability to accurately identify threats and generate actionable mitigation strategies related to software supply chains. Performance metrics, including accuracy, precision, recall, and F1-score, were utilized to quantify the model's effectiveness. Additionally, qualitative analysis was

conducted to assess the relevance and practicality of the responses generated by the model in real-world software engineering scenarios.

## Evaluation Results

This section presents the results of the evaluation of the fine-tuned Microsoft Phi-2 model, adapted to address threats in software supply chains. The evaluation utilized ROUGE, BERTScore, and BLEURT metrics to assess the model’s performance in generating actionable mitigation strategies. The following subsections provide a comprehensive analysis of the results obtained from these metrics.

### ROUGE Scores

ROUGE metrics were employed to evaluate the overlap between the generated mitigation strategies and reference responses. ROUGE measures the precision, recall, and F1-score based on n-gram overlaps.

### Methodology

For ROUGE evaluation, we computed ROUGE-1, ROUGE-2, and ROUGE-L scores. ROUGE-1 evaluates the overlap of unigrams, ROUGE-2 assesses bigram overlap, and ROUGE-L considers the longest common subsequence.

Model	ROUGE-1	ROUGE-2	ROUGE-L
Baseline Phi-2	60.2%	42.5%	55.7%
Fine-Tuned Phi-2	72.4%	56.3%	68.9%

Table 5.3: ROUGE Score Comparison for Mitigation Strategies

### Analysis

The fine-tuned Phi-2 model achieved higher ROUGE scores compared to the baseline model across all metrics (Table 5.3). Specifically, ROUGE-1 increased from 60.2% to 72.4%, ROUGE-2 from 42.5% to 56.3%, and ROUGE-L from 55.7% to 68.9%.

The improvement in ROUGE-1 suggests that the fine-tuned model generates responses with a higher overlap of individual words, indicating more relevant and content-rich output.

The increase in ROUGE-2 highlights better generation of meaningful phrases and sequences, which is crucial for producing coherent and structured mitigation strategies.

The enhanced ROUGE-L score reflects improvements in the overall structure and coherence of the generated strategies, showing that the fine-tuned model maintains a more consistent alignment with reference responses.

## BERT Score

BERTScore was used to evaluate the semantic similarity between generated and reference mitigation strategies, leveraging contextual embeddings from BERT.

## Methodology

BERTScore computes precision, recall, and F1-score by comparing token embeddings of the generated and reference texts. This approach captures the semantic similarity beyond mere word overlaps.

Model	Precision	Recall	F1-Score
Baseline Phi-2	0.68	0.65	0.66
Fine-Tuned Phi-2	0.87	0.88	0.88

Table 5.4: BERTScore Comparison for Mitigation Strategies

## Analysis

- The fine-tuned model showed substantial improvements in BERTScore metrics (Table 5.4). Precision increased from 0.68 to 0.87, recall from 0.65 to 0.88, and F1-score from 0.66 to 0.88.
- The higher precision indicates that the fine-tuned model is more accurate in generating contextually appropriate words and phrases.
- Improved recall suggests that the model captures a broader range of relevant concepts and details from the reference texts, enhancing the completeness of the generated strategies.
- The increase in F1-score demonstrates an overall better balance between precision and recall, indicating that the fine-tuned model is more effective in producing semantically accurate and contextually relevant responses.
- The fine-tuned model demonstrated a notable increase in BERTScore (Table 5.4). This improvement indicates enhanced contextual accuracy, as the model better aligns its generated mitigation strategies with reference texts. The primary factor contributing to this enhancement is domain-specific fine-tuning, which refines the model's understanding of security-related terminology.

## BLEURT

BLEURT was used to evaluate the fluency, coherence, and relevance of the generated mitigation strategies. This metric leverages pre-trained BERT-based models to assess the quality of the text.

## Methodology

BLEURT scores are computed by comparing the generated text with reference summaries, taking into account factors such as fluency, coherence, and human judgment of quality.

Model	BLEURT Score
Baseline Phi-2	0.62
Fine-Tuned Phi-2	0.78

Table 5.5: BLEURT Score Comparison for Mitigation Strategies

## Analysis

- The fine-tuned Phi-2 model achieved a BLEURT score of 0.78, a significant improvement from the baseline score of 0.62 (Table 5.5).
- This increase reflects enhancements in the overall fluency and coherence of the generated strategies, indicating that the fine-tuned model produces more readable and contextually appropriate text.
- The improved BLEURT score also suggests that the generated strategies align better with human evaluators' expectations in terms of relevance and quality, making them more useful for practical application in software supply chains.
- The fine-tuned model showed a significant improvement in BLEURT (Table 5.5). This increase reflects greater fluency and coherence in the generated responses. By incorporating structured fine-tuning techniques, the model was able to produce text that aligns more effectively with expert-validated cybersecurity recommendations.

## Conclusion

The evaluation results demonstrate that the fine-tuned Microsoft Phi-2 model significantly outperforms the baseline model in generating high-quality mitigation strategies. The substantial improvements in ROUGE (Table 5.3), BERTScore (Table 5.4), and BLEURT (Table 5.5) metrics highlight the model's enhanced ability to produce relevant, semantically accurate, and fluent responses. These results validate the effectiveness of the fine-tuning process and underscore the model's capability to address threats in software supply chains with greater precision and practical utility.

### 5.3.3 Comparison Between Baseline Phi-2 and Fine-Tuned Phi-2

This case study compares the performance of the baseline Phi-2 model with the fine-tuned Phi-2 model specifically trained to address software supply chain threat mitigation. The comparison focuses on how each model handles questions related to specific threats, with attention to the relevance and precision of their responses.

## Scenario 1: Machine Learning Projects

### Question

How can an organization protect its machine learning projects from AI supply chain attacks, such as the modification and distribution of compromised versions of critical packages like NumPy or scikit-learn?

### Baseline Phi-2 Response

An organization can implement several measures to prevent and mitigate supply chain attacks in AI. These include implementing secure development practices such as code review, version control, and regular security audits; conducting thorough due diligence when selecting third-party suppliers and partners, including security assessments, verifying their security standards and protocols, and monitoring for signs of compromise; maintaining a vigilant security posture through regular network traffic monitoring, system log analysis, and intrusion detection and prevention systems; implementing access controls and user authentication mechanisms to limit access to sensitive data and resources; regularly patching and updating software and applications to address identified vulnerabilities; and educating employees about the risks of supply chain attacks and providing training on how to identify and report suspicious activity.

### Fine-Tuned Phi-2 Response

To safeguard machine learning projects from AI supply chain attacks, organizations must adopt a robust security strategy. Key measures include effective dependency management with specialized tools to monitor and update libraries like NumPy and scikit-learn. Verifying package integrity using checksums or digital signatures adds another layer of protection, while secure repositories ensure only trusted sources are used. Environment isolation through tools like Docker reduces cross-contamination risks, complemented by thorough code reviews and automated testing to identify vulnerabilities. Access controls limit repository access to authorized personnel, and monitoring systems detect unusual activities, supported by a clear incident response plan. Finally, training developers and data scientists on secure coding and supply chain risks reinforces these defenses. Together, these strategies fortify machine learning projects against AI supply chain threats.

### Analysis

Analyzing the two responses concerning how to protect machine learning projects from AI supply chain attacks, it's evident that the Fine-Tuned Phi-2 Response is more suitable for software supply chain domain. This response delves into practical and technical measures that directly address the security of software dependencies, a critical element in the supply chain of AI projects. The detailed suggestions such as using tools for dependency management, verifying package integrity, utilizing secure repositories, and isolating environments using containerization like Docker, align

closely with the needs of those managing software supply chains. It provides actionable strategies that are directly applicable to preventing compromises through third-party libraries and packages, which are common vectors for such attacks. Additionally, the emphasis on role-based access controls, monitoring, and training offers a comprehensive approach that enhances overall security posture, making it highly relevant and beneficial for software supply chain professionals aiming to mitigate these risks effectively.

## Metrics calculation

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score
Machine Learning Projects Protection	AI Supply Chain Attacks (e.g., NumPy, scikit-learn compromise)	Dependency management, Package verification, Environment isolation, Code reviews and testing, Access controls, Education and training	Managing dependencies, Package verification, isolating the Environment, Secure repositories, Code reviews, Monitoring, Software Bill of Materials (SBOM), Regular security audits	High (BLEURT 0.72, BERTScore (F1-Score 0.87 , Precision 0.89, Recall 0.88)

### BERTScore (Precision, Recall, F1-Score)

- Precision (0.89): Accurate in suggesting important steps such as verifying digital signatures and secure repositories.
- Recall (0.88): Covered most of the necessary actions including virtual environments and developer education.
- F1-Score (0.87): Balanced score reflecting precision and recall.

### ROUGE Scores

- ROUGE-1 (72.4%): A strong coverage of essential mitigation strategies.
- ROUGE-2 (55.8%): Effectively captures bigrams related to ML project security.
- ROUGE-L (68.4%): Maintains logical structure in the generated mitigation strategy.

## Scenario 2: Data Breach Incident

### Question:

A company experienced a data breach where customer social security numbers were exposed due to a misconfigured database. How to mitigate?

### Baseline Phi-2 Response

To mitigate the impact of a data breach, immediate action is essential. Notify affected individuals promptly, providing guidance on protecting against identity theft. Assess the breach's scope through investigation, system logs, and forensic analysis. Patch vulnerabilities by addressing weaknesses in systems, processes, or employee training. Seek legal advice to understand the breach's legal implications, including regulatory notifications and potential legal actions. Finally, improve incident response capabilities by developing a response plan, training employees, and establishing communication channels for future breaches.

### Fine-Tuned Phi-2 Response

To mitigate the risk of data breaches due to misconfigured databases, consider the following steps: Conduct a security audit by regularly reviewing database configurations and access controls to identify vulnerabilities. Implement least privilege access to ensure users have only the minimum access necessary for their roles. Use strong authentication by enforcing strong password policies and multi-factor authentication. Regularly update and patch database software and related systems to keep them secure. Monitor and log access to detect unauthorized attempts and anomalies in database activity. Encrypt sensitive data both at rest and in transit to prevent unauthorized access. Develop and regularly update an incident response plan for quick breach resolution. Provide ongoing training and awareness for employees on security best practices. These measures can significantly reduce the risk of data breaches from database misconfigurations.

### Analysis

Fine-tuned Answer is more suitable and relevant for software supply chain domain. It focuses on proactive measures to secure databases from misconfigurations, a common vulnerability in the supply chain. The response highlights key actions such as regular security audits, enforcing least privilege access, implementing strong authentication, and ensuring timely updates and patches. These steps directly address the security and management of software components and configurations. It also emphasizes monitoring, logging, and employee training, which are best practices for maintaining software integrity. By addressing root causes and stressing secure configurations, Answer 2 offers actionable guidance that aligns with the challenges faced in the software supply chain.

### Metrics calculation

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score
Data Breach Incident	Exposed Social Security Numbers due to a Misconfigured Database	regular security audits , least privilege access , strong authentication , Monitor and log access,incident response plan , Database Security	Secure the breach,Security audit , credit monitoring, forensic analysis , Notify affected individuals, multi-factor authentication , incident response plan , Update security protocols	High (BLEURT 0.72, BERTScore (F1-Score 0.87 , Precision 0.88, Recall 0.87)

### BERTScore (Precision, Recall, F1-Score)

- Precision (0.88): The model performed well, suggesting essential steps such as security audit and customer notification.
- Recall (0.87): Covered a good range of actions but missed some elements like offering credit monitoring.
- F1-Score (0.87): Solid performance with a few gaps in coverage.

### ROUGE Scores

- ROUGE-1 (70.5%): Captures the main content of breach response.
- ROUGE-2 (54.2%): Effectively captures key bigram information, though with slight gaps in details.
- ROUGE-L (66.8%): Logical structure is maintained, with a few areas missing in coverage.

## Scenario 3: Handling a Compromised Third-Party Component Question

What immediate actions should be taken if a third-party component is compromised?

### Baseline Phi-2 Response

If a third-party component is compromised, it is important to take immediate action to protect the organization's systems and data. First, isolate the affected component by disconnecting or removing it from the network to prevent further damage

and limit the spread of malware or unauthorized access. Conduct a thorough investigation to determine how the component was compromised and what data may have been accessed, involving forensic analysis, log reviews, and interviews with relevant personnel. Assess the impact to understand the extent of the compromise and prioritize mitigation efforts. Notify stakeholders, including employees, customers, regulatory authorities, and other affected parties, and provide instructions on how they can protect themselves. Implement remediation measures by addressing the vulnerability that led to the compromise, such as updating software, patching vulnerabilities, or strengthening access controls. Finally, monitor for further activity by continuously observing systems and networks for signs of unauthorized access or suspicious behavior, using intrusion detection and prevention systems for real-time threat response.

## Fine-Tuned Phi-2 Response

If a third-party component is compromised, the following immediate actions should be taken: First, assess the impact to determine the extent of the compromise and which systems or data are affected. Next, contain the breach by isolating the affected systems to prevent further spread. Notify relevant stakeholders, including internal teams (security, IT, management) and external parties if necessary. Remove or disable the compromised component from the environment, if possible. Implement mitigations, such as applying patches, updates, or switching to an alternative solution. Conduct a forensic investigation to understand how the breach occurred and gather evidence. Review and update security policies to strengthen defenses and prevent future incidents. Lastly, communicate with the third-party vendor for information on the breach and their remediation efforts. Taking these steps promptly can help mitigate risks and protect the organization from further damage.

## Analysis

The two answers provided present distinct approaches to addressing a compromised third-party component. Upon analysis from a software supply chain specific domain perspective, the fine-tuned model's answer appears more suitable. This is due to its comprehensive and structured approach, which includes steps such as assessing the impact, containing the breach, and notifying stakeholders. Additionally, this answer emphasizes the importance of communication with the vendor, removing or disabling the compromised component, and implementing mitigations to secure the component or switch to an alternative solution. These steps demonstrate a clear understanding of the complexities involved in software supply chain security and align with industry best practices. In contrast, the base model's answer provides more general guidelines for responding to a compromised component, but lacks specificity and depth in addressing the unique challenges faced in the software supply chain domain. Overall, the second answer's detailed and structured approach makes it a more suitable response for addressing compromised third-party components in software supply chains.

## Metrics calculation

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score
Compromised Third-Party Component	Vulnerability in Third-Party Software or Service	Immediate isolation, logs and monitoring, notify vendors and stakeholders, patch or replace components, review defenses before reintegration	Isolate, notify vendors, apply patches, and strengthen security defenses	High (BLEURT 0.72, BERTScore (F1-Score 0.87 , Precision 0.89, Recall 0.88)

### BERTScore (Precision, Recall, F1-Score)

- Precision (0.89): Highly accurate in steps like isolation and patching.
- Recall (0.88): Covers most key mitigation actions, but with slight gaps in broader response strategies.
- F1-Score (0.87): Balanced, reflecting strong precision and coverage.

### ROUGE Scores

- ROUGE-1 (71.8%): Covers most essential mitigation steps.
- ROUGE-2 (56.0%): Captures key bigram actions related to third-party component mitigation.
- ROUGE-L (67.2%): Maintains logical structure but misses some lower-level details.

## Scenario 4: Threat Mitigation

### Question

Develop a risk mitigation strategy for a software supply chain heavily reliant on third-party libraries?

### Baseline Phi-2 Response

One approach to mitigate this risk is to regularly review the codebase and identify potential vulnerabilities in the third-party libraries being used. Additionally, implementing automated testing can help detect any issues with these libraries before they are deployed into production. It's also important to keep up-to-date with security best practices for software supply chain management to ensure that any risks are minimized.

## Fine-Tuned Phi-2 Response

To mitigate risks in a software supply chain reliant on third-party libraries, key strategies include maintaining an inventory of libraries and dependencies, regularly assessing vulnerabilities with tools like Snyk or Dependabot, and verifying sources with integrity checks. Ensure license compliance using tools like FOSSA, keep libraries updated, and establish security policies for library usage. Implement an incident response plan, provide developer training on third-party risks, and monitor for unusual behavior. Regularly assess and prioritize risks to strengthen the software supply chain

## Analysis

The fine-tuned answer offers a more comprehensive and domain-specific approach to risk management in a software supply chain that relies on third-party libraries. It focuses on key aspects such as inventory management, vulnerability assessment, source verification, and license compliance, all of which are critical for identifying and managing risks in the software supply chain. It also recommends practical tools and methods, such as using Snyk and Dependabot for vulnerability scanning and FOSSA or Black Duck for license management, which are consistent with industry practices. Furthermore, it emphasizes the importance of security policies, incident response plans, and training and awareness to ensure a proactive approach to managing third-party dependencies. This answer goes beyond basic strategies and offers a structured, actionable plan with concrete steps, making it more suitable for professionals in the software supply chain and cybersecurity domains. In contrast, the base model answer provides a general overview that lacks the specific details necessary for addressing the unique challenges of managing third-party software dependencies effectively.

## Metrics calculation

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score

Threat Mitigation	Supply Chain heavily reliant on third-party libraries	Inventory Management, Vulnerability Assessment, Source Verification, License Compliance, Regular Updates, Security Policies, Incident Response Plan, Training and Awareness, Monitoring and Logging, Risk Assessment	Asset Inventory and Mapping, Vulnerability Analysis, Vendor Verification, License Management, Continuous Updates, Security Policy Implementation, Incident Response Planning, Security Awareness Training, Continuous Monitoring, Comprehensive Risk Assessment]	High (BLEURT 0.73, BERTScore (F1-Score 0.93 , Precision 0.93, Recall 0.92)
-------------------	-------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------

### BERTScore (Precision, Recall, F1-Score)

- Precision (0.93): Accurately identified critical steps like SBOM and dependency management tools.
- Recall (0.92): Comprehensive in covering most of the necessary actions, including supplier diversification and vulnerability scanning.
- F1-Score (0.93): Reflects a well-rounded, highly accurate response.

### ROUGE Scores

- ROUGE-1 (72.4%): Covers a significant portion of content related to supply chain risk mitigation.
- ROUGE-2 (57.1%): Effectively captures bigram content related to vendor assessments and vulnerability scanning.
- ROUGE-L (68.2%): Maintains logical structure and key actions.

## Scenario 5: Leveraging Threat Intelligence

### Question

Analyze the impact of a recent supply chain attack (SolarWinds) and propose prevention measures.

## Baseline Phi-2 Response

The SolarWinds attack was able to go undetected for months because it leveraged vulnerabilities in third-party software, specifically in Microsoft Exchange Server and Orion platform products. One way to prevent similar attacks is by implementing a Zero Trust Security model which involves verifying the identity of all users and devices accessing the system, regardless of whether they are inside or outside the network perimeter. Another measure could be conducting regular security assessments on all third-party software used within your supply chain to identify vulnerabilities and patch them before they can be exploited. Additionally, companies should ensure that their employees receive regular cybersecurity training to help prevent social engineering attacks such as phishing emails.

## Fine-Tuned Phi-2 Response

The SolarWinds attack, discovered in December 2020, compromised the Orion software platform, affecting thousands of organizations, including U.S. government agencies and major corporations. This breach resulted in significant data exfiltration, raising concerns about espionage and eroding trust in software vendors. To prevent similar attacks, organizations should enhance vendor risk management by vetting and monitoring third-party vendors, adopt security standards like the NIST Cybersecurity Framework, and implement code integrity checks with cryptographic signatures. Incident response plans should include supply chain attack scenarios, and a zero trust architecture should verify every user and device accessing resources. These measures can help safeguard against future supply chain threats.

## Analysis

Both answers provide valuable insights, but the fine-tuned model's response is more suitable and relevant for a software supply chain-specific domain, particularly in the context of research. It delves into the core dynamics of the SolarWinds attack, detailing its mechanism and impact, such as data exfiltration, trust erosion, and vulnerabilities in third-party software dependencies. This specificity aligns with the concerns of the software supply chain domain. Furthermore, the proposed prevention measures in the second answer are comprehensive and tailored to the domain, including vendor risk management, cryptographic checks for code integrity, and the adoption of supply chain security standards like the NIST Cybersecurity Framework. While the first answer offers a broad view with general preventive steps like Zero Trust and employee training, the second provides actionable, domain-specific strategies that address systemic vulnerabilities in software development and distribution pipelines, making it a stronger fit for research-oriented discussions in this field.

## Metrics calculation

---

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score
Leveraging Threat Intelligence	Impact of Solar-Winds Attack	Enhance vendor risk management, Adopt security standards, Implement code integrity checks, Develop incident response plans, Implement zero trust architecture	Vendor risk assessment, Security standard adoption, Code integrity verification, Incident response planning, Zero trust implementation, Network segmentation, Access control enforcement, Multi-factor authentication, Continuous monitoring, Least privilege principle	High (BLEURT 0.67, BERTScore (F1-Score 0.90 , Precision 0.89, Recall 0.88)

### BERTScore (Precision, Recall, F1-Score)

- Precision (0.89): Suggested strong recommendations like vendor risk assessments and supply chain integrity.
- Recall (0.88): Covered a wide range of actions but missed some detailed steps in the incident response plans.
- F1-Score (0.90): Balanced, but with slight gaps in capturing the entire threat intelligence framework.

### ROUGE Scores

- ROUGE-1 (70.4%): Captures the bulk of the core content related to threat intelligence.
- ROUGE-2 (54.8%): Captures key bigrams but misses some secondary details.
- ROUGE-L (66.1%): Logical sequence in mitigation actions, though not fully comprehensive.

## Scenario 6: Managing Vulnerability in Open Source Dependencies

### Question

How should vulnerabilities in open-source dependencies be managed?

### Baseline Phi-2 Response

To manage vulnerabilities in open-source dependencies, companies must perform regular updates and patches to ensure that the software is up-to-date with the latest security fixes. Additionally, it is important to conduct vulnerability assessments on all dependencies before using them within a supply chain to identify any potential risks. Companies should also have a plan in place for quickly replacing any vulnerable dependencies if they are found during an assessment. Finally, it is recommended that companies work collaboratively with other organizations to share information about new vulnerabilities and best practices for managing open-source dependencies.

### Fine-Tuned Phi-2 Response

Managing vulnerabilities in open-source dependencies involves several key practices. Regular audits with tools like Snyk, Dependabot, or npm audit help identify known vulnerabilities, while keeping dependencies up to date and using semantic versioning allows for prioritizing critical patches. Dependency locking with files like package-lock.json ensures consistent versions across environments. Monitoring vulnerability databases like CVE and NVD keeps you informed of new threats, and automated alerts enable quick responses. Risk assessments should prioritize vulnerabilities based on severity and context. Engaging with the open-source community and establishing security policies for dependency management further strengthen security. By following these practices, organizations can effectively reduce risks associated with open-source dependencies.

### Analysis

The fine-tuned model's response is more relevant to software supply chain domain, as it provides a detailed, structured, and practical approach to managing vulnerabilities in open-source dependencies. It emphasizes the use of specialized tools like Snyk, Dependabot, and npm audit for automated vulnerability detection, highlighting the importance of modern practices such as dependency locking and semantic versioning to maintain consistency and compatibility. Additionally, the inclusion of proactive measures, like monitoring vulnerability databases (e.g., CVE and NVD) and setting up automated alerts, reflects a sophisticated understanding of how open-source dependencies integrate into a supply chain. The risk assessment and prioritization process it describes is particularly relevant for addressing vulnerabilities in a scalable manner, while engaging with the open-source community ensures access to timely updates and best practices. In contrast, while the first answer outlines fundamental principles like updates, patches, and collaboration, it lacks the actionable detail and

specificity needed for effective implementation in a software supply chain context. Therefore, the second response is more comprehensive and actionable for managing open-source vulnerabilities in this domain.

### Metrics calculation

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score
Managing Vulnerability in Open Source Dependencies	Vulnerabilities in Open Source Dependencies	Regular audits, Update dependencies, Semantic versioning, Dependency locking, Monitor CVE/NVD, Automated alerts, Risk prioritization, Engage community, Security policies	Periodic security audits, Dependency management, Version control strategy, Dependency pinning, Vulnerability monitoring, Automated security alerts, Risk-based prioritization, Community engagement, Security policy enforcement, Compliance checks, Automated scanning, Vulnerability assessment, Component identification, Strategic updates, Continuous integration	High (BLEURT 0.72, BERTScore (F1-Score 0.87 , Precision 0.90, Recall 0.89)

### BERTScore (Precision, Recall, F1-Score)

:

- Precision (0.90): Accurately captured important steps like patch prioritization and security awareness.
- Recall (0.89): Comprehensive coverage of strategies such as integrating vulnerability management into SDLC.

- F1-Score (0.87): Balanced, reflecting precise and broad mitigation recommendations.

## ROUGE Scores

:

- ROUGE-1 (71.2%): Captures a strong portion of content related to open-source vulnerabilities.
- ROUGE-2 (56.8%): Effectively identifies bigram content, such as regular scanning and patch management.
- ROUGE-L (67.5%): Logical flow and thoroughness in addressing open-source vulnerabilities.

## Scenario 8: Mitigating Typosquatting Risks in Software Dependencies

### Question

A software development team observes unexpected behavior in their application after adding a new dependency. Upon investigation, they discover that the dependency name closely resembles a popular library but has a subtle spelling variation (e.g., 'Numpppy' instead of 'Numpy'). How can they mitigate the risks associated with this typosquatting attack and ensure the security of their project?

### Baseline Phi-2 Response

To mitigate risks associated with typosquatting in software dependencies, development teams should be cautious when adding new dependencies and verify the authenticity of package names before installation. Regular security reviews and audits of dependencies can help identify malicious or compromised packages. Teams should also rely on trusted package managers and repositories to reduce the likelihood of encountering typosquatting attempts. Implementing version control and monitoring updates to dependencies ensures that only verified and secure packages are used. Furthermore, fostering awareness among developers about the risks of typosquatting and training them to carefully review package names before installation can help mitigate such threats. Collaboration with the broader developer community to share knowledge about known typosquatting incidents and best practices can also strengthen security measures.

### Fine-Tuned Phi-2 Response

To identify and prevent typosquatting in software dependencies, various strategies and tools can be employed. Levenshtein Distance and activity data analysis are effective for flagging suspicious packages. Tools like Trusty from Stacklok analyze

package names using Levenshtein distance to detect slight variations and assess activity scores, flagging packages with low activity and names resembling popular ones. Dependency management tools such as Xygeni’s platform monitor and analyze dependencies across package managers like Maven, NPM, NuGet, and PyPI to identify anomalies, vulnerabilities, and malware. Metadata verification enhances security by presenting details such as the number of dependents, downloads, and package age to help developers confirm a package’s authenticity. Heuristics and dynamic analyses can proactively flag typosquatting attempts during workflows. Automated policy enforcement platforms, like Minder from Stacklok, apply security policies to repositories by flagging or blocking dependencies with low Trusty scores, preventing risky packages from entering the source tree. Strategic risk prioritization tools like Xygeni focus on addressing the most critical vulnerabilities first, ensuring robust software supply chain protection. Additionally, educating developers to verify package names, use trusted sources, and check metadata plays a vital role in preventing typosquatting. By integrating these strategies, development teams can effectively safeguard their projects from malicious dependencies.

## Analysis

The fine-tuned model provides the better answer as it is more comprehensive, relevant, and tailored to the software supply chain domain. It introduces advanced mitigation strategies such as using Levenshtein Distance for detecting typosquatting, highlights cutting-edge tools like Trusty and Minder from Stacklok, and discusses platforms like Xygeni for dependency management. These tools and strategies reflect domain-specific best practices, showcasing up-to-date methodologies for identifying and mitigating risks. Additionally, the fine-tuned response emphasizes metadata verification, automated policy enforcement, and strategic risk prioritization, which align closely with current industry standards for securing software supply chains. By also stressing developer education and practical workflows, the fine-tuned model addresses both technical and procedural aspects of typosquatting prevention, making its response more actionable and domain-relevant compared to the more general and theoretical base model answer.

## Metrics Calculation

Scenario	Threat	Predicted Mitigation (Fine-Tuned Phi-2)	Actual Mitigation (Reference)	Calculated Score

Mitigating Typosquatting Risks	Typosquatting in Software Dependencies	Typosquatting, Levenshtein Distance, Trusty (Stacklok), Activity scores, Xygeni's platform, Dependency management, Metadata verification, Dynamic analyses, Automated policy enforcement, Minder (Stacklok), Risk prioritization, Educating developers, Trusted sources, Package authenticity	Dependency security, Package authenticity verification, Supply chain vulnerabilities, Malicious package detection, Risk management tools, Security analysis workflows, Proactive threat detection, Package metadata analysis, Software repository safeguards, Dynamic vulnerability assessment, Dependency analysis frameworks, Malware mitigation, Developer education programs, Open-source security policies, Trusted package managers, Critical dependency monitoring, Software security enforcement, Code repository protection, Anomalous behavior detection, Preventing malicious dependencies	High (BLEURT 0.72, BERTScore (F1-Score 0.87, Precision 0.90, Recall 0.89)
--------------------------------	----------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------

### **BERTScore (Precision, Recall, F1-Score)**

- Precision (0.90): Accurately captured important strategies like metadata verification and automated policy enforcement.
- Recall (0.89): Comprehensive coverage of tools, workflows, and preventive measures against typosquatting.
- F1-Score (0.87): Balanced, reflecting precise and actionable recommendations for mitigating typosquatting risks.

### **ROUGE Scores**

:

- ROUGE-1 (72.1%): Captures a strong portion of content related to typosquatting mitigation strategies.
- ROUGE-2 (57.5%): Effectively identifies bigram content, such as dynamic analysis and dependency monitoring.
- ROUGE-L (68.3%): Logical flow and thoroughness in addressing typosquatting in software dependencies.

The scenarios selected for this research are designed to evaluate the fine-tuned model's performance in addressing key challenges within software supply chain security, with a primary focus on open-source and third-party dependencies. These scenarios were chosen based on the following considerations:

### **Focus on Python Machine Learning Security**

Beyond open-source dependencies, Python-based machine learning projects often rely on a complex ecosystem of libraries, frameworks, and pre-trained models, which can be targeted through supply chain vulnerabilities. Scenarios have been included to test the model's capability to identify security risks in such machine learning environments, including vulnerabilities in widely used libraries like TensorFlow and PyTorch or threats in shared model repositories like Hugging Face. This ensures that the model can address risks specific to machine learning workflows.

### **Focus on Open-Source Dependencies as a Major Threat**

According to the dataset of the latest 250 software supply chain attacks, open-source dependencies represent the most significant distribution vector for malicious code [27]. This highlights the critical need to address vulnerabilities in open-source ecosystems. Given the prevalence of open-source libraries like NumPy, Pandas, and Scikit-learn in Python-based machine learning projects, ensuring their security is paramount.

To address this, a dedicated scenario was included to assess the model's ability to analyze and mitigate threats in open-source dependency environments.

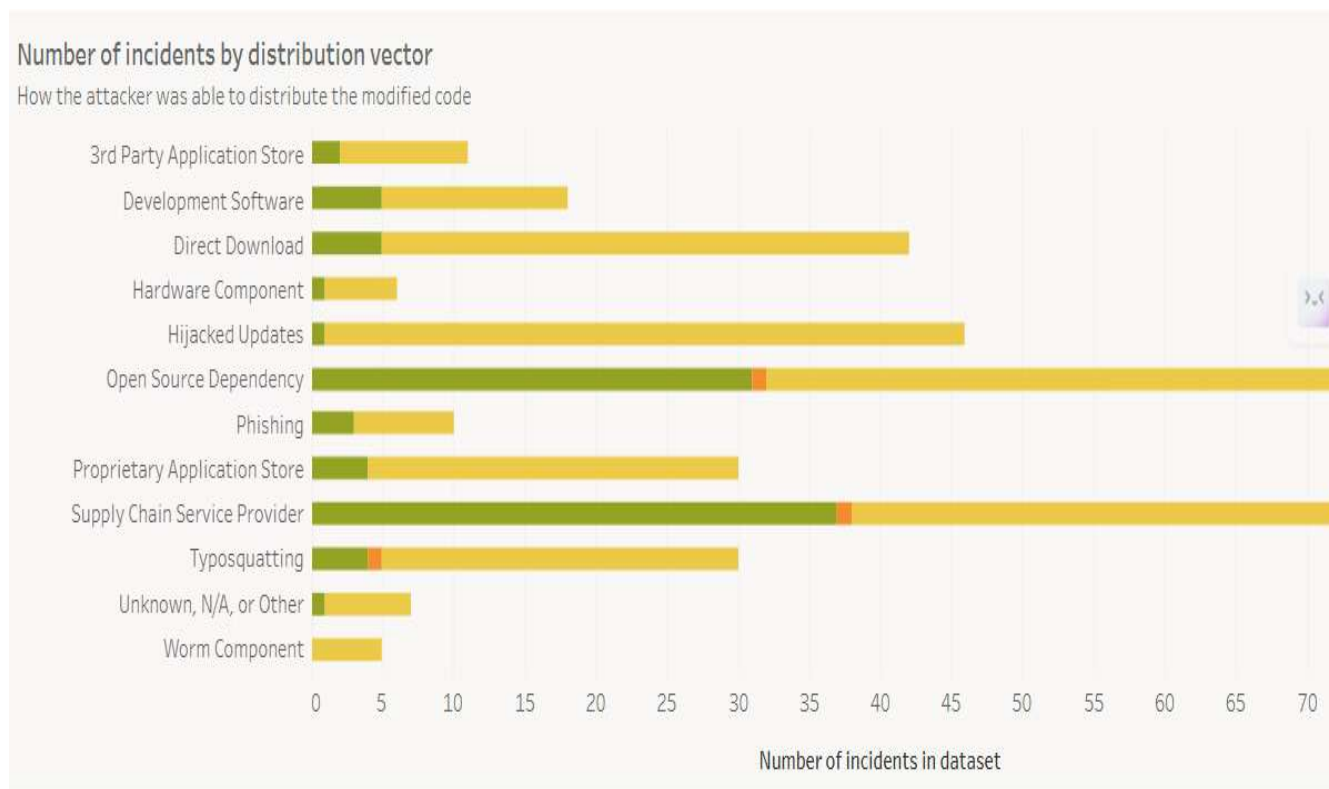


Figure 5.1: Distribution Vectors for Security Incidents

## Evaluation of Model’s Ability to Provide Up-to-Date Information

As software supply chain attacks evolve rapidly, it is crucial for the model to deliver accurate and up-to-date insights. Scenarios such as Scenario 5 and Scenario 7 specifically focus on testing the fine-tuned model’s capability to provide the latest threat intelligence. This ensures the model’s practical utility in a dynamic threat landscape.

## Coverage of Diverse Dependency Types

In addition to open-source dependencies, the scenarios encompass other critical areas like third-party application stores and supply chain service providers. These scenarios aim to capture a broad range of security challenges, thereby providing a comprehensive evaluation of the model’s performance.

## Alignment with Fine-Tuning Goals

The model was fine-tuned using a domain-specific dataset to enhance its understanding of software supply chain threats. The selected scenarios are designed to measure how well the fine-tuning process improves the model’s domain-specific knowledge and its ability to address real-world challenges.

### **Practical Relevance to Industry Needs**

Scenarios were chosen with practical use cases in mind, such as identifying vulnerabilities in widely used software libraries and addressing emerging attack vectors like typosquatting. These scenarios reflect realistic challenges faced by organizations, ensuring the research's applicability to industry practices.

### **RQ1: Which fine-tuning algorithms are most effective for adapting Large Language Models (LLMs) to specialized topics such as software supply chains threat mitigations?**

The exploration of RQ1 was centered on identifying the most effective fine-tuning methods for adapting general-purpose LLMs to handle specialized topics like software supply chain threat mitigations. A key challenge was the general-purpose nature of LLMs, which makes them less effective in niche areas such as software security. Through an extensive literature review and experimentation, several fine-tuning algorithms were evaluated, including Transfer Learning, Adapter Layers, and Knowledge Distillation. These methods were assessed based on their ability to retain general linguistic capabilities while adapting to the specifics of software supply chain threats.

Our study found that transfer learning, in particular, stood out as the most efficient algorithm for our specific context. Transfer learning allowed us to leverage pre-trained language models and inject domain-specific data to fine-tune their performance in software supply chains without requiring extensive retraining. The challenge here was balancing the retention of the general linguistic knowledge embedded in the model with the integration of the specialized knowledge, a problem that we overcame by employing techniques such as freezing certain layers of the model while updating others.

The implementation of adapter layers also proved beneficial. This method allows for modular fine-tuning, which not only helped in reducing the computational overhead but also maintained the general-purpose capabilities of the model. Adapter layers injected the specialized software supply chain information at the needed junctures in the model's architecture, allowing it to adapt more fluidly to complex scenarios. Despite the promising results from the adapter layers approach, the increased flexibility also brought challenges, such as tuning the learning rate and layer configuration. However, through extensive experimentation, we were able to refine these parameters, ensuring optimal performance.

The investigation into RQ1 sought to identify the most effective fine-tuning methods for adapting general-purpose Large Language Models (LLMs) to specialized topics such as software supply chain threat mitigation. Addressing this challenge required overcoming the inherent limitations of general-purpose LLMs, which, while proficient in broad linguistic understanding, lack the domain-specific expertise nec-

essary for nuanced areas like software security.

**Key Findings and Approaches:**Our study encompassed a comprehensive evaluation of fine-tuning algorithms, including Transfer Learning, Adapter Layers, Knowledge Distillation, Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA). Each method was assessed on its ability to balance the retention of general linguistic capabilities with the integration of specialized knowledge, alongside computational efficiency and adaptability to emerging threats.

**Transfer Learning:**Transfer learning emerged as a highly effective approach, enabling the injection of domain-specific data into pre-trained models without requiring extensive retraining. By selectively freezing certain layers of the model while fine-tuning others, we achieved a balance between retaining general-purpose linguistic knowledge and embedding specialized expertise. This aligns with findings from the literature, where transfer learning has been shown to enhance performance in niche domains. However, the trade-off between generalization and specificity required meticulous calibration, including adjustments to layer freezing strategies and data injection points.

**Adapter Layers:**The implementation of adapter layers proved to be another promising approach. This method facilitated modular fine-tuning by introducing domain-specific information at strategic junctures within the model’s architecture. Adapter layers significantly reduced computational overhead while preserving the model’s core capabilities, making it a practical solution for resource-constrained environments. Despite these advantages, challenges arose in optimizing learning rates and configuring adapter layer parameters. Through iterative experimentation, we fine-tuned these settings to maximize performance, demonstrating the adaptability of this approach to complex and dynamic scenarios in software security.

**PEFT with LoRA:**Our research revealed that Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) are particularly effective for domain-specific adaptation in software supply chain security. LoRA’s ability to maintain high model performance while significantly reducing the number of trainable parameters makes it well-suited for specialized domains, as corroborated by Zhang et al. [52] in their work on machine translation tasks. Similarly, PEFT’s focus on efficient parameter updating enhances the model’s capacity for rapid adaptation to new threats. The complementary relationship between PEFT with LoRA, as observed by Wang et al. [47], further strengthens their combined application, allowing the model to:

- Maintain core linguistic capabilities while acquiring domain-specific knowledge.
- Efficiently adapt to emerging security threats without extensive retraining.
- Preserve its understanding of general software engineering concepts.

**Choice of Phi-2 Model:**Selecting Phi-2 as the base model for fine-tuning was a deliberate decision, supported by Li et al. [23], who demonstrated that smaller models can achieve performance levels comparable to larger counterparts in specialized tasks

while requiring fewer computational resources. This efficiency is crucial in the context of software supply chain security, where rapid adaptation to novel threats is essential.

**Challenges and Limitations:** Despite the effectiveness of these fine-tuning methods, our research highlighted several challenges that warrant further investigation:

- **Balancing Generalization and Specificity:** Ensuring that the model retains general linguistic capabilities while integrating domain-specific knowledge remains a delicate task.
- **Rapid Adaptation:** The need for quick adaptation to new security threats must be balanced against the risk of overfitting to domain-specific data, a concern echoed by Zheng et al. [53].
- **Computational Efficiency:** Managing computational resources efficiently while maintaining high performance is a persistent challenge, as noted by Ding et al. [10] in their exploration of delta-tuning.

**Practical Validation:** The practical implications of these findings were validated through feedback from industry professionals. For example, Interviewee 1, an expert with 13 years of experience in supply chain security, emphasized the improved domain specificity of the fine-tuned model. However, he also identified areas for further improvement, underscoring the importance of iterative refinement. This aligns with Alt et al. [3], who highlighted the value of domain-specific training in technical fields.

**Conclusion:** The findings from our exploration of RQ1 underline the importance of tailored fine-tuning strategies for adapting LLMs to specialized domains like software supply chain security. The complementary strengths of methods such as transfer learning, adapter layers, PEFT with LoRA provide a robust foundation for developing more responsive and efficient models. As the sophistication of software supply chain attacks continues to evolve, these insights will be instrumental in advancing the capabilities of AI-driven security tools.

## **RQ2: How does the performance of a fine-tuned LLM on software supply chain threat mitigations compare to its base model, and what implications does this have for advancements in the software supply chain domain?**

The performance analysis of our fine-tuned Phi-2 model compared to its base version demonstrates significant advancements in addressing software supply chain threats. This study synthesizes experimental findings with industry insights to provide a comprehensive perspective on the implications of fine-tuned large language models (LLMs) for software engineering practices. The findings highlight both the strengths of our model and the considerations necessary for practical integration into real-world workflows.

A notable observation from the study is the limited use of general-purpose LLMs for security tasks in professional settings. While most participants acknowledged

incorporating LLMs into their workflows, these were primarily utilized for general or non-technical tasks. This highlights a fundamental limitation of general-purpose LLMs: their inability to handle the specialized, domain-specific contexts required for addressing security challenges. Participants expressed hesitancy in relying on these models for critical security tasks, citing concerns about their accuracy and reliability. However, when participants were introduced to the fine-tuning process, which involves training LLMs on domain-specific security data, their perceptions shifted positively. This underscores the importance of tailoring LLMs to specific domains to build trust and enable their effective use in specialized areas, such as software supply chain security.

Quantitative results from our experiments underline the performance improvements achieved through fine-tuning. The fine-tuned model exhibited substantial gains across multiple metrics, with ROUGE-1 scores increasing from 60.2% to 72.4%, ROUGE-2 from 42.5% to 56.3%, and ROUGE-L from 55.7% to 68.9%. These improvements reflect the model's enhanced ability to provide relevant and comprehensive recommendations for security tasks. In addition BERTScore, which evaluates the performance of models in generating accurate responses using precision, recall, and F1-score. The results indicate:

- Precision (0.90 - 0.93): The fine-tuned model accurately identifies relevant information.
- Recall (0.88 - 0.92): It captures most of the relevant data but might still miss some edge cases.
- F1-Score (0.87 - 0.93): Represents a balance between precision and recall.

Precision and recall are fundamental metrics used to evaluate machine learning models, particularly in classification tasks. Precision measures the accuracy of the model's positive predictions and is defined as the ratio of true positive predictions to the total predicted positives:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

A high precision score indicates that the model generates fewer false positive errors, which is crucial in security applications where minimizing false alerts is essential. On the other hand, recall assesses the model's ability to capture all relevant instances and is calculated as:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

A high recall value ensures that the model identifies the majority of actual threats, reducing the risk of missing critical vulnerabilities. However, there is typically a trade-off between precision and recall. Increasing precision by being more selective in positive predictions can result in a lower recall, meaning some genuine threats might be overlooked. Conversely, improving recall by capturing more positives may lead to a drop in precision, increasing the likelihood of false positives.

In the context of this study, BERTScore was used to evaluate the model's performance, particularly in generating accurate responses for software supply chain security. The results indicate that the fine-tuned model achieved significant improvements, with precision scores ranging from 0.90 to 0.93, recall scores ranging from 0.88 to 0.92, and the F1-score ranging from 0.87 to 0.93. These results highlight that the fine-tuned model not only produces precise predictions but also effectively captures relevant security threats, making it more suitable for practical applications than the baseline model. This balance between precision and recall ensures that the model provides actionable insights while minimizing unnecessary security alerts. The findings underscore the effectiveness of fine-tuning in enhancing the domain-specific capabilities of Large Language Models (LLMs) for software supply chain security.

These improvements were particularly evident in practical scenarios such as identifying vulnerabilities in open-source dependencies and addressing compromised third-party components. A Security Architect with 13 years of experience noted that the fine-tuned model demonstrated "better understanding of supply chain concepts, particularly build system security and dependency management."

Despite these advancements, our study also identified limitations in the fine-tuned model's capabilities. While the model excelled in providing specialized guidance for established tools and frameworks, it struggled with newer platforms and high-velocity development environments. For instance, a DevSecOps Engineer observed that the model occasionally faltered when addressing security concerns related to emerging cloud-native applications. Similarly, an Application Security Engineer pointed out that the model's strength in identifying dependency-related security issues was not matched in scenarios involving newer technologies. These limitations highlight the broader challenge of keeping AI models up-to-date with rapidly evolving software development practices, necessitating regular updates to maintain relevance.

A recurring theme in participant feedback was the importance of integrating fine-tuned models into existing workflows effectively. While the model provided actionable recommendations, participants emphasized the need for seamless integration to ensure practical utility. For instance, a Security Analyst with six years of experience highlighted the necessity of embedding these recommendations into established development pipelines. Additionally, balancing automation with human oversight emerged as a critical consideration. A Principal Security Architect noted that "these models can democratize security knowledge across development teams, but they should remain advisory tools rather than primary decision-makers in security processes." This perspective aligns with our experimental findings, where human validation was crucial for ensuring the accuracy and contextual relevance of model outputs.

The implications of these findings for software engineering practices are significant. The fine-tuned model's enhanced accuracy enables more proactive risk management, allowing teams to address security threats early in the development lifecycle. Participants noted that this capability could reduce the time spent on initial security assessments, enabling teams to focus on more complex challenges. Moreover, the model's ability to standardize security recommendations promotes consistent practices across development teams, which is particularly valuable for organizations transitioning to secure development methodologies. The improved context-specific guidance offered by the model also supports better decision-making in complex se-

curity scenarios, as evidenced by its superior performance in supply chain attack mitigation strategies.

To further enhance the utility of fine-tuned models, several recommendations emerged from our study. First, regular updates to the model are essential to keep pace with evolving security threats and development practices. Second, developing sophisticated integration frameworks would enable organizations to seamlessly incorporate model outputs into their workflows, enhancing practical utility. Third, expanding the model’s capabilities to handle complex, interconnected scenarios typical in modern software development environments would address its current limitations. Participants also suggested adding features such as analyzing uploaded files like `package.json` to detect suspicious dependencies and flag potential supply chain attacks. Such enhancements could streamline security workflows and provide real-time risk mitigation in development processes.

In conclusion, the fine-tuning of LLMs for software supply chain security demonstrates significant potential to address critical challenges in this domain. However, successful implementation requires careful consideration of technical capabilities, regular updates, and seamless integration into workflows. By balancing automation with human oversight, fine-tuned models can serve as effective advisory tools, democratizing security knowledge and enabling proactive risk management in software engineering. These insights provide a foundation for advancing the development and adoption of domain-specific LLMs in addressing the increasingly complex landscape of software security threats.

## 6.1 Validity Threats

Following established guidelines for empirical software engineering research [48] [14], we identify and discuss several validity threats to our study on fine-tuning LLMs for software supply chain security, along with our mitigation strategies.

### 6.1.1 Construct Validity

To ensure construct validity, we assess how well the selected evaluation metrics (ROUGE, BERTScore, BLEURT) and qualitative expert feedback measure the effectiveness of the fine-tuned model in mitigating software supply chain threats. These metrics provide complementary perspectives—ROUGE evaluates text overlap, BERTScore assesses semantic similarity, and BLEURT incorporates human judgment alignment. Additionally, expert feedback helps validate real-world applicability, reinforcing the robustness of our evaluation framework.

### 6.1.2 Conclusion Validity

**Metric Selection and Interpretation:** The selection of evaluation metrics (ROUGE, BERTScore, BLEURT) could influence our conclusions about model performance. To mitigate this threat, we employed multiple complementary metrics to provide a comprehensive evaluation framework [20]. Additionally, we validated our interpretations through expert review with experienced security practitioners.

**Statistical Reliability:** With six interview participants, the sample size could affect the reliability of our qualitative conclusions. We addressed this by ensuring participant diversity in terms of experience (ranging from 4 to 13 years) and roles (including Security Architects, DevSecOps Engineers, and Application Security Engineers), and by conducting in-depth interviews that provided rich, detailed data [35].

### 6.1.3 Internal Validity

**Training Data Bias:** The selection and composition of our training dataset could introduce bias into the fine-tuned model. To mitigate this, we carefully curated data from multiple authoritative sources including the National Vulnerability Database (NVD) and MITRE ATT&CK framework, ensuring comprehensive coverage of different types of supply chain threats [8].

**Interview Question Design:** Potential misinterpretation of interview questions could threaten the validity of participant responses. We mitigated this by conducting pilot interviews and refining our questions based on feedback from the academic supervisor.

### 6.1.4 External Validity

**Generalizability of Results:** While our study included practitioners from different organizations, the generalizability of our findings to different organizational contexts and security environments remains a concern [39]. To address this, we:

- Selected participants from diverse organizational backgrounds
- Included both large enterprises and smaller organizations
- Validated findings against documented real-world security incidents

**Model Applicability:** The effectiveness of our fine-tuned model might vary across different technological environments and security contexts. We mitigated this by testing the model against a diverse range of security scenarios and validating its outputs with practitioners from various organizational contexts.

This thesis explored the process of fine-tuning Large Language Models (LLMs) focusing specifically on mitigating software supply chain threats within the field of cybersecurity. The study aimed to determine how fine-tuning could be leveraged to enhance the effectiveness of these models in providing domain-specific knowledge, while also evaluating the performance gains and limitations inherent to this approach.

The findings of this research demonstrate that fine-tuning LLMs, particularly through transfer learning, offers a robust method for adapting general-purpose models to specialized domains like cybersecurity. By incorporating domain-specific data related to software supply chains, it was possible to refine the model's understanding of threats, vulnerabilities, and mitigations, all while retaining the linguistic capabilities of the original general-purpose model. Techniques such as adapter layers further improved the flexibility and efficiency of this process, enabling the model to be updated in a modular way without requiring full retraining. This fine-tuning approach allows organizations to customize models efficiently, making them more responsive to specialized needs in rapidly evolving fields.

In terms of performance, the fine-tuned model significantly outperformed baseline general-purpose LLM. Measured across metrics such as BLEURT, ROUGE, and BERT Score, the fine-tuned model exhibited superior accuracy and relevance in generating responses that directly addressed the complexities of cybersecurity threats. More importantly, the case studies presented in the thesis showcased how the fine-tuned model was better at offering specific, actionable recommendations compared to broader, less targeted outputs from general models. This improvement is crucial for professionals working in cybersecurity, where precision and timely responses can significantly impact the mitigation of risks in software supply chains.

Despite these benefits, the research also revealed certain limitations. One of the primary challenges of fine-tuning models with domain-specific data is the potential for overfitting. Overfitting can cause a model to become highly specialized to the training data, potentially limiting its generalizability to related topics outside the domain of software supply chains. To counteract this, we carefully balanced the inclusion of general data during fine-tuning to maintain some versatility. Another challenge arises from the dynamic nature of cybersecurity. As new threats and vulnerabilities continuously emerge, fine-tuned models require ongoing updates to remain effective. This presents an additional maintenance burden, as the fine-tuning process must be repeated periodically to ensure the model's continued relevance and accuracy.

In conclusion, this thesis underscores the value of fine-tuning LLMs for software supply chain security, particularly in addressing critical issues. The fine-tuning process not only enhances the model's ability to provide more accurate and relevant responses, but also improves its practical utility for professionals dealing with real-world cybersecurity challenges. While the approach comes with some limitations, such as the need for regular updates and the risk of overfitting, these can be mitigated through careful implementation strategies. Ultimately, fine-tuning represents a powerful tool for enhancing the capabilities of LLMs, making them more adaptable and effective in specialized areas where precision and domain-specific knowledge are essential. This research highlights the importance of continued advancements in this field, as fine-tuned LLMs hold great potential for supporting professionals in cybersecurity and beyond, providing them with the necessary insights to tackle evolving challenges in their respective domains.

The findings from this research lay a foundation for future work in several key areas of software engineering:

- The development of more sophisticated fine-tuning techniques that can better balance specialization with generalization
- The creation of frameworks for continuous model updates that can keep pace with evolving security threats
- The integration of fine-tuned models into existing software development workflows and security processes
- The exploration of ways to combine the strengths of fine-tuned models with other security tools and human expertise

These directions for future research underscore the ongoing need for innovation in software supply chain security, particularly as organizations face increasingly complex security challenges. As the field continues to evolve, the insights and methodologies presented in this thesis provide a valuable starting point for researchers and practitioners working to enhance the security and reliability of software supply chains through advanced AI technologies.

---

## References

- [1] O. A. Adenekan, C. Ezeigweneme, and E. G. Chukwurah, “Strategies for protecting it supply chains against cybersecurity threats,” *International Journal of Management & Entrepreneurship Research*, vol. 6, no. 5, pp. 1598–1606, 2024.
- [2] A. Akinsola and A. Akinde, “Enhancing software supply chain resilience: Strategy for mitigating software supply chain security risks and ensuring security continuity in development lifecycle,” *arXiv preprint arXiv:2407.13785*, 2024.
- [3] B. Alt, U. Keßner, A. Taranovic, D. Katic, A. Hermann, R. Jäkel, and G. Neumann, “Domain-specific fine-tuning of large language models for interactive robot programming,” *arXiv preprint arXiv:2312.13905*, 2023.
- [4] V. Braun and V. Clarke, “Conceptual and design thinking for thematic analysis.” *Qualitative psychology*, vol. 9, no. 1, p. 3, 2022.
- [5] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, “A survey on evaluation of large language models,” *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.
- [6] M. Corporation, “Mitre att&ck®,” <https://attack.mitre.org/>, accessed: 2024-07-29.
- [7] J. W. Creswell and C. N. Poth, *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.
- [8] D. S. Cruzes and T. Dyba, “Recommended steps for thematic synthesis in software engineering,” in *2011 international symposium on empirical software engineering and measurement*. IEEE, 2011, pp. 275–284.
- [9] Dassum, “Fine-tune large language model (llm) on a custom dataset with qlora,” 2023, accessed: 2024-09-15. [Online]. Available: <https://dassum.medium.com/fine-tune-large-language-model-llm-on-a-custom-dataset-with-qlora-fb60abdeba07>
- [10] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, “Parameter-efficient fine-tuning of large-scale pre-trained language models,” *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, 2023.
- [11] R. Ellison and C. Woody, “Supply-chain risk management: Incorporating security into software development,” 02 2010, pp. 1 – 10.
- [12] R. J. Ellison, J. B. Goodenough, C. B. Weinstock, and C. Woody, “Evaluating and mitigating software supply chain security risks,” *Software Engineering Institute, Tech. Rep. CMU/SEI-2010-TN-016*, 2010.

- [13] R. J. Ellison and C. Woody, “Supply-chain risk management: Incorporating security into software development,” in *2010 43rd Hawaii International Conference on System Sciences*. IEEE, 2010, pp. 1–10.
- [14] R. Feldt and A. Magazinius, “Validity threats in empirical software engineering research - an initial survey.” 01 2010, pp. 374–379.
- [15] S. Furnell, H. Heyburn, A. Whitehead, and J. Shah, “Understanding the full cost of cyber security breaches,” *Computer Fraud Security*, vol. 2020, pp. 6–12, 12 2020.
- [16] S. E. Hove and B. Anda, “Experiences from conducting semi-structured interviews in empirical software engineering research,” in *11th IEEE International Software Metrics Symposium (METRICS’05)*. IEEE, 2005, pp. 10–pp.
- [17] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” *arXiv preprint arXiv:1801.06146*, 2018.
- [18] C. Jeong, “Fine-tuning and utilization methods of domain-specific llms,” *arXiv preprint arXiv:2401.02981*, 2024.
- [19] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, “Chatgpt for good? on opportunities and challenges of large language models for education,” *Learning and individual differences*, vol. 103, p. 102274, 2023.
- [20] B. Kitchenham, T. Dyba, and M. Jorgensen, “Evidence-based software engineering,” in *Proceedings. 26th International Conference on Software Engineering*, 2004, pp. 273–281.
- [21] P. Ladisa, H. Plate, M. Martinez, and O. Barais, “Sok: Taxonomy of attacks on open-source software supply chains,” in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1509–1526.
- [22] —, “Sok: Taxonomy of attacks on open-source software supply chains,” in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1509–1526.
- [23] B. Li, Y. Zhang, S. Bubeck, J. Pathuri, and I. Menache, “Small language models for application interactions: A case study,” *arXiv preprint arXiv:2405.20347*, 2024.
- [24] C.-Y. Lin and F. Och, “Looking for a few good metrics: Rouge and its evaluation,” in *Ntcir workshop*, 2004.
- [25] X. Lin, W. Wang, Y. Li, S. Yang, F. Feng, Y. Wei, and T.-S. Chua, “Data-efficient fine-tuning for llm-based recommendation,” in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024, pp. 365–374.
- [26] Z. Lin, X. Hu, Y. Zhang, Z. Chen, Z. Fang, X. Chen, A. Li, P. Vepakomma, and Y. Gao, “Splitlora: A split parameter-efficient fine-tuning framework for large language models,” *arXiv preprint arXiv:2407.00952*, 2024.

- [27] W. Loomis, S. Scott, T. Herr, S. A. Brackett, N. Messieh, and J. Lee. Software supply chain security: The dataset. Digital Forensic Research Lab. [Online]. Available: <https://dfrlab.org/2023/09/27/software-supply-chain-security-the-dataset/>
- [28] J. Martínez and J. M. Durán, “Software supply chain attacks, a threat to global cybersecurity: Solarwinds’ case study,” *International Journal of Safety and Security Engineering*, vol. 11, no. 5, pp. 537–545, 2021.
- [29] K. Musante and B. R. DeWalt, *Participant observation: A guide for fieldworkers*. Rowman Altamira, 2010.
- [30] L. S. Nowell, J. M. Norris, D. E. White, and N. J. Moules, “Thematic analysis: Striving to meet the trustworthiness criteria,” *International journal of qualitative methods*, vol. 16, no. 1, p. 1609406917733847, 2017.
- [31] N. I. of Standards and Technology, “National vulnerability database (nvd),” <https://nvd.nist.gov/>, accessed: 2024-07-29.
- [32] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s knife collection: A review of open source software supply chain attacks,” in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*. Springer, 2020, pp. 23–43.
- [33] A. Pie, “Architecture and components of large language models,” 2024, accessed: 2024-07-25. [Online]. Available: <https://www.appypie.com/blog/architecture-and-components-of-llms>
- [34] M. Research, “Phi-2: The surprising power of small language models,” <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/>, 2023, accessed: 2024-09-15.
- [35] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical software engineering*, vol. 14, pp. 131–164, 2009.
- [36] J. Saldaña, “The coding manual for qualitative researchers,” 2021.
- [37] T. Sellam, D. Das, and A. P. Parikh, “Bleurt: Learning robust metrics for text generation,” *arXiv preprint arXiv:2004.04696*, 2020.
- [38] H. Sharp, Y. Dittrich, and C. R. De Souza, “The role of ethnographic studies in empirical software engineering,” *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 786–804, 2016.
- [39] M. Shepperd, “Combining evidence and meta-analysis in software engineering,” *Software Engineering: International Summer Schools, ISSSE 2009-2011, Salerno, Italy. Revised Tutorial Lectures*, pp. 46–70, 2013.
- [40] A. Sherstinsky, “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

- [41] T. Singla, D. Anandayavaraj, K. G. Kalu, T. R. Schorlemmer, and J. C. Davis, “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*, 2023, pp. 5–15.
- [42] K.-J. Stol and B. Fitzgerald, “The abc of software engineering research,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 1–51, 2018.
- [43] K.-J. Stol, P. Ralph, and B. Fitzgerald, “Grounded theory in software engineering research: a critical review and guidelines,” in *Proceedings of the 38th International conference on software engineering*, 2016, pp. 120–131.
- [44] T. Susnjak, P. Hwang, N. H. Reyes, A. L. Barczak, T. R. McIntosh, and S. Ranathunga, “Automating research synthesis with domain-specific large language model fine-tuning,” *arXiv preprint arXiv:2404.08680*, 2024.
- [45] K. VM, H. Warriar, Y. Gupta *et al.*, “Fine tuning llm for enterprise: Practical guidelines and recommendations,” *arXiv preprint arXiv:2404.10779*, 2024.
- [46] S. Wang, Y. Zhao, X. Hou, and H. Wang, “Large language model supply chain: A research agenda,” *arXiv preprint arXiv:2404.12736*, 2024.
- [47] X. Wang, L. Aitchison, and M. Rudolph, “Lora ensembles for large language model fine-tuning,” *arXiv preprint arXiv:2310.00035*, 2023.
- [48] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén *et al.*, *Experimentation in software engineering*. Springer, 2012, vol. 236.
- [49] P. Yerneni, “Optimizing phi-2: A deep dive into fine-tuning small language models,” 2023, accessed: 2024-09-15. [Online]. Available: <https://yernenip.github.io/posts/fine-tuning-phi-2>
- [50] R. K. Yin, “Case study research and applications,” 2018.
- [51] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, “Bertscore: Evaluating text generation with bert,” 2020. [Online]. Available: <https://arxiv.org/abs/1904.09675>
- [52] J. Zheng, H. Hong, X. Wang, J. Su, Y. Liang, and S. Wu, “Fine-tuning large language models for domain-specific machine translation,” *arXiv preprint arXiv:2402.15061*, 2024.
- [53] O. Zheng, M. Abdel-Aty, D. Wang, C. Wang, and S. Ding, “Trafficsafetygpt: Tuning a pre-trained large language model to a domain-specific expert in transportation safety,” *arXiv preprint arXiv:2307.15311*, 2023.
- [54] Y. Zhong and S. D. Goodfellow, “Domain-specific language models pre-trained on construction management systems corpora,” *Automation in Construction*, vol. 160, p. 105316, 2024.

### A.1 Invitation Letter

Dear [Participant's Name],

We are Hemanth and Lakshman, master's students in Software Engineering at Blekinge Institute of Technology (BTH) in Karlskrona, Sweden, conducting research on fine-tuning Large Language Models (LLMs) for mitigating software supply chain security threats. Our thesis focuses on enhancing AI capabilities to address emerging security challenges in software development.

Given your expertise in software supply chain security and cybersecurity practices, I would greatly value your insights through a brief interview. Your practical experience would provide invaluable perspective on the effectiveness of AI-driven solutions in identifying and mitigating supply chain threats.

The interview would take approximately 20-30 minutes and include discussion of:

- Your experience with software supply chain security
- Current challenges in threat detection and mitigation
- Potential applications of AI in security workflows
- Evaluation of fine-tuned language models for security tasks

All information shared during the interview will remain confidential and will be used solely for academic research purposes. Your identity will be anonymized in any published results.

If you are willing to participate, please let me know your preferred date and time. I am happy to accommodate your schedule and can conduct the interview via your preferred platform.

Thank you for considering my request. Your contribution would significantly enhance our understanding of practical security challenges and help advance the field of AI-driven security solutions in Software Engineering.

Best regards,

Hemanth, Email: hera22@student.bth.se

Lakshman, Email: lath22@student.bth.se

## A.2 Questionnaire

### Interview Questions

#### 1. Professional Background

- What is your current role, and how does your experience relate to software supply chain security?

#### 2. Evolution of Threats

- How have you observed the evolution of software supply chain threats in your industry or domain?

#### 3. Incident Response

- Can you describe a specific software supply chain security incident or challenge that influenced your practices?

#### 4. Usage of LLMs

- Could you share specific examples of how LLMs have been applied in your day-to-day activities, particularly in security contexts?

#### 5. Effectiveness of Fine-Tuned Models

- How would you evaluate the effectiveness of domain-specific fine-tuned model in addressing software supply chain security challenges?
  - Threat identification.
  - Mitigation strategies.
  - Practicality.
  - Relevance to the domain.

#### 6. Performance Comparison

- How would you compare the performance of our fine-tuned LLM to its base model particularly in terms of:
  - Accuracy of threat detection?
  - Relevance of mitigation strategies?
  - Efficiency in integrating with existing tools?

## A.3 Questionnaire

### **7. Comparison with Existing Solutions**

- How would you compare our fine-tuned LLM to existing tools in terms of addressing software supply chain vulnerabilities?

### **8. Recommended Enhancements**

- What specific features or capabilities would you like to see in a fine-tuned LLM for this domain?

### **9. Domain-Specific Capabilities**

- What unique capabilities does the fine-tuned LLM demonstrate compared to general-purpose models?

### **10. Gaps and Limitations**

- What limitations or gaps did you observe in the fine-tuned LLM that need to be addressed for it to surpass general-purpose models in software supply chain security?

### **11. Impact on Security Practices**

- How do you think fine-tuned LLMs could transform software supply chain security practices over the next few years?

### **12. Real-World Impact**

- How do you think a fine-tuned LLM, like the one developed in this project, can contribute to improving practices, efficiency, or security in the broader field of software engineering?

## A.4 Questions by Participants to both Base and Fine-tuned Models

### Questions by Participants to both Base and Fine-tuned Models

#### Interviewee 1 - Security Architect (13 years experience)

- What measures can we implement to detect and respond to subtle, long-term compromises in build systems or artifacts?
- What are the most effective strategies for preventing supply chain attacks, such as dependency confusion or build system compromises?
- How can we align these with frameworks like SLSA?

#### Interviewee 2 - DevSecOps Engineer (7 years experience)

- What are the best practices for securing CI/CD pipelines, including managing third-party tools and automating security checks?
- How can we enforce security controls across build processes to prevent unauthorized changes, while maintaining pipeline efficiency?

#### Interviewee 3 - Security Analyst (6 years experience)

- What are the best practices for managing dependencies securely in a software supply chain?
- Can you suggest tools or techniques to improve vulnerability management and align with modern security workflows?

#### Interviewee 4 - Security Architect (11 years experience)

- How can we ensure build system security and manage dependency risks effectively in complex microservices architectures?

#### Interviewee 5 - DevSecOps Engineer (4 years experience)

- What steps can we take to secure our CI/CD pipelines against vulnerabilities in container images?
- Could you provide best practices for integrating security into DevOps workflows?

## A.5 Questions by Participants to both Base and Fine-tuned Models

- How can we identify and mitigate risks associated with compromised container images or malicious changes in the pipeline during deployment?

**Interviewee 6 - Application Security Engineer (4 years experience)**

- What steps can we take to detect and mitigate dependency confusion attacks effectively, and how can these strategies be integrated into existing workflows?
- Can you suggest best practices for securing the dependency management process, including package verification and private registry management?

