AI for DevSecOps: A Landscape and Future Opportunities

MICHAEL FU, Monash University, Australia
JIRAT PASUKSMIT, Atlassian, Australia
CHAKKRIT TANTITHAMTHAVORN, Monash University, Australia

DevOps has emerged as one of the most rapidly evolving software development paradigms. With the growing concerns surrounding security in software systems, the DevSecOps paradigm has gained prominence, urging practitioners to incorporate security practices seamlessly into the DevOps workflow. However, integrating security into the DevOps workflow can impact agility and impede delivery speed. Recently, the advancement of artificial intelligence (AI) has revolutionized automation in various software domains, including software security. AI-driven security approaches, particularly those leveraging machine learning or deep learning, hold promise in automating security workflows. They reduce manual efforts, which can be integrated into DevOps to ensure uninterrupted delivery speed and align with the DevSecOps paradigm simultaneously. This paper seeks to contribute to the critical intersection of AI and DevSecOps by presenting a comprehensive landscape of AI-driven security techniques applicable to DevOps and identifying avenues for enhancing security, trust, and efficiency in software development processes. We analyzed 99 research papers spanning from 2017 to 2023. Specifically, we address two key research questions (RQs). In RQ1, we identified 12 security tasks associated with the DevOps process and reviewed existing AI-driven security approaches. In RQ2, we discovered 15 challenges encountered by existing AI-driven security approaches and derived future research opportunities. Drawing insights from our findings, we discussed the state-of-the-art AI-driven security approaches, highlighted challenges in existing research, and proposed avenues for future opportunities.

CCS Concepts: • **Software and its engineering → Software development techniques**; • **Security and privacy** → *Software security engineering*; • **Computing methodologies** → Artificial intelligence.

Additional Key Words and Phrases: DevOps, DevSecOps, Artificial Intelligence, Deep Learning, Machine Learning, AI Security, Vulnerability, Supply Chain Security

ACM Reference Format:
Michael Fu, Jirat Pasuksmit, and Chakkrit Tantithamthavorn. 2024. AI for DevSecOps: A Landscape and Future Opportunities. 1, 1 (April 2024), 45 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The traditional software development lifecycle (SDLC) adopts a sequential and siloed approach, with distinct phases executed in a linear fashion, resulting in limited collaboration, slow feedback loops, increased risk of defects, difficulty in managing changes, lack of agility, and increased costs and time-to-market. To address these limitations, DevOps emerged over a decade ago, combining development (Dev) and operations (Ops) to integrate individuals, processes, and technology across application planning, development, delivery, and operations. Moreover, DevOps facilitates coordination and collaboration among previously segregated roles like development, quality engineering,

Authors' addresses: Michael Fu, Monash University, Clayton, Australia, yeh.fu@monash.edu; Jirat Pasuksmit, Atlassian, Sydney, Australia, jpasuksmit@atlassian.com; Chakkrit Tantithamthavorn, Monash University, Clayton, Australia, chakkrit@monash.edu.

ACM XXXX-XXXX/2024/4-ART
https://doi.org/XXXXXXX.XXXXXXX

and security [127]. DevOps has been a widely adopted SDLC [117], with organizations generally expressing satisfaction and positivity regarding their transition to DevOps practices [49].

While DevOps improves collaboration, automation, and agility in software development and operations, it often overlooks security considerations until later stages of the development process [131]. This delayed focus on security can lead to vulnerabilities and risks being introduced into the software, potentially exposing organizations to cyber threats and compliance issues. To overcome these security limitations and ensure the secure delivery of software products while preserving DevOps agility, the concept of DevSecOps (Development, Security, and Operations) emerged. DevSecOps involves considering application and infrastructure security from the outset and automating certain security gates to prevent slowdowns in the DevOps workflow [222]. Incorporating security practices into DevOps to achieve DevSecOps poses several challenges. However, common security practices, such as security code review [80], often demand significant manual effort, potentially hindering the agility of DevOps. A recent study underscores the need for developer-centric application security testing tools tailored to the continuous practices within DevSecOps. Moreover, it advocates for further research into automating traditionally manual security practices to align with the rapid software deployment cycles [153].

In recent years, the rapid advancement of artificial intelligence (AI) technologies has transformed various domains, including software development and cybersecurity. As organizations increasingly adopt DevSecOps practices [65], integrating security into the software development lifecycle becomes paramount. Furthermore, AI-based security approaches are trending in research, and security awareness in the software industry is increasing. As highlighted in Google's 2023 State of DevOps Report, DevOps teams believe that AI will play a crucial role in data analysis, security tasks, and bug identification [62]. Notably, the Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence, issued by the White House [78], underscores the importance of safety, responsible innovation, equity, transparency, and international collaboration in AI development and deployment. In summary, there has been a growing focus on the trend of DevSecOps in software development, along with an increased emphasis on utilizing AI for enhancing security measures.

We observe that several systematic literature reviews (SLRs) have explored the DevSecOps domain from diverse perspectives. As presented in Table 1, one SLR focused on AI-driven approaches (i.e., machine learning and deep learning) specifically for the operation and monitoring step in DevSecOps [9], while other studies, not primarily focused on AI, covered all steps of DevSecOps [4, 100, 118, 131, 132, 145, 153, 187]. However, none of the previous studies reviewed AI-driven approaches for all steps in DevSecOps comprehensively. Despite the growing intersection of AI and DevSecOps, our analysis reveals a notable absence of an SLR specifically examining literature concerning AI-driven methodologies and tools aimed at automating and enhancing the security aspect of DevOps that helps achieve the DevSecOps paradigm. To bridge this gap, our research aims to contribute to this critical intersection by providing a comprehensive landscape of AI techniques applicable to DevSecOps and identifying future opportunities for enhancing security, trust, and efficiency in software development processes.

In this article, *we define AI approaches as those employing machine learning or deep learning algorithms.* We iteratively defined our search string and searched papers from top-tier software engineering and security conferences and journals, focusing on those published between 2017 and 2023. This timeframe was chosen due to the significant advancements in AI and Large Language Models (LLMs) since the proposal of the transformer architecture in 2017. Our automated literature-searching process yielded a collection of 1,683 papers, from which we manually reviewed and filtered out 1,595 papers based on our selection criteria. Moreover, our snowballing search identified

11 additional papers. We systematically analyzed the collected 99 papers and presented two key aspects: Firstly, we identified existing AI-based security methods that can be integrated into the DevOps workflow. Secondly, we delved into the challenges and future research opportunities arising from the current landscape of AI-based security methods. Based on our SLR, this article presents the following contributions:

- We presented the landscape of the existing AI-based security approaches that can be integrated into DevOps workflow to fulfil the DevSecOps paradigm.
- We identified the themes of challenges faced by state-of-the-art AI-based security approaches and derive future research opportunities.

2 BACKGROUND AND RELATED WORK

In this section, we define the DevOps process of this study and present an overview of each step in DevOps. We then compare our study with existing reviews on DevSecOps.

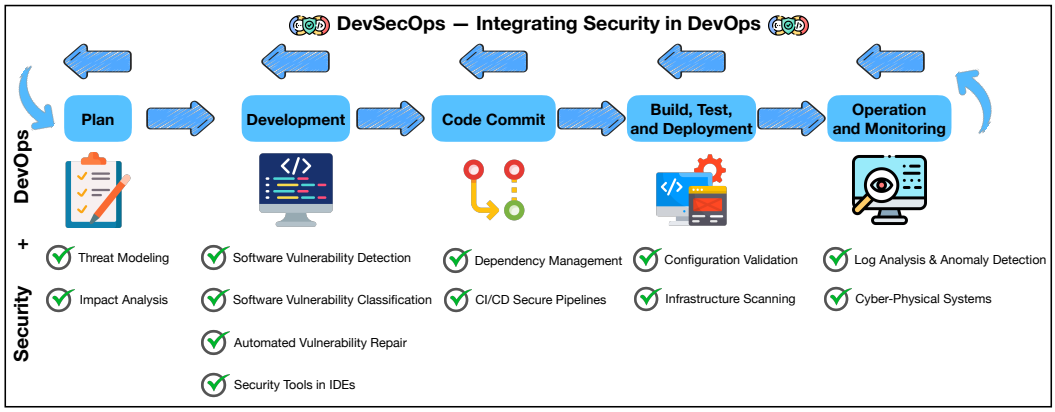

Fig. 1. The overview of the DevOps workflow and the identified security tasks relevant under each step in the DevOps process.

2.1 DevOps

For consistency, we followed the definition provided by Microsoft [125] and determined DevOps as a five-step workflow (depicted in Figure 1): (1) Plan, (2) Development, (3) Code Commit, (4) Build, Test and Deployment, and (5) Operation and Monitoring. Below, we introduce each step in detail.

Plan. In the "Plan" step of DevOps, teams define project goals, requirements, and timelines. This phase establishes the initial roadmap for the project, involving activities such as gathering user stories, prioritizing features, and assigning tasks. During this phase, the team identifies project objectives and security needs, engaging in threat modeling to grasp security vulnerabilities and plan security measures accordingly [166]. Furthermore, software impact analysis is conducted to identify entities directly or indirectly influenced by a change [8]. This involves the process of assessing and estimating the potential consequences before implementing a modification in the deployed product [93].

Development. In the "Development" step of DevOps, software engineers are responsible for implementing the features and functionalities outlined in the requirements and specifications established during the planning phase. In this step, developers operate within an Integrated Development

Environment (IDE) where they use static analysis tools (e.g., Checkmarx [22], Flawfinder [200], and Snyk [170]) to scan for potential errors and vulnerabilities before compiling the code.

Code Commit. In the "Code Commit" step of DevOps, developers use version control systems like Git to commit their code changes, facilitating collaboration and tracking changes. This step is integral to the Continuous Integration/Continuous Deployment (CI/CD) pipeline, where CI involves the regular integration of developers' work into the main branch of the version control system [173], while CD automates the deployment of software changes to production without human intervention [164]. Furthermore, dependency management plays a critical role in this step, as modern software often relies heavily on third-party code, such as external libraries, to streamline development processes [101]. However, this practice can introduce dependency vulnerabilities [144], underscoring the importance of effectively managing external libraries and dependencies to ensure the reliability and security of the software product. For instance, Dependabot [61] in GitHub helps users monitor their software dependencies and issues security alerts to users if finding vulnerable dependencies.

Build, Test, and Deployment. In the "Build and Test" step of DevOps, software code undergoes compilation and rigorous testing to ensure functionality and reliability. Depending on the organization's infrastructure and DevOps practices, these processes may occur either on-premises or in the cloud. In software systems, various configurations are used to control features, endpoints (e.g., cache server addresses), security, fault tolerance, tunable behaviors (e.g., timeouts, throttling limits) and so on [83]. Thus, configuration validation tools are used to ensure the proper configuration of cloud environments. Infrastructure as Code (IaC) simplifies infrastructure management by provisioning consistent environments through machine-readable code, eliminating the need for manual provisioning and management of servers and other components during application development and deployment [154]. Various tools and platforms support IaC, including Terraform, Cloudify, Docker Swarm, Kubernetes, Packer, and Chef, Ansible, Puppet for configuration management [66]. Then, the validated software code is deployed to the production environment to make it available for end-users, involving customization, configuration, and installation [20]. This phase involves automating the deployment process to ensure consistency and reliability [84].

Operation and Monitoring. In the "Operation and Monitoring" step of DevOps, the focus shifts to maintaining and monitoring the deployed software to ensure optimal performance and security. This phase involves leveraging actionable intelligence and employing data-driven, event-driven processes to promptly identify, evaluate, and respond to potential risks [125]. Log analysis is commonly used to detect and diagnose abnormal behavior, enhancing system reliability using data from application logs and runtime environments [18]. Additionally, anomaly detection involves identifying uncommon and unexpected occurrences over time [14]. Hagemann and Katsarou [68] categorize methods for detecting anomalies into three groups: machine learning, deep learning, and statistical approaches. Finally, feedback loops are established to gather insights from the aforementioned monitoring activities, enabling continuous improvement and refinement of the DevOps pipeline and security measures.

2.2 Other Reviews in DevSecOps

We are aware that there are existing literature reviews of the DevSecOps process, and we have identified 9 related reviews, which are summarized in Table 1. Myrbakken and Colomo-Palacios [131] conducted one of the early literature reviews of DevSecOps, focusing on providing an overview and defining the DevSecOps process, considering DevSecOps as a relatively new concept at the time. Prates et al. [145] reviewed DevSecOps metrics that can be used to monitor the process. For

Table 1. Comparison with other related reviews focusing on DevSecOps. **#P** - total number of reviewed papers, **SLR** - Is the study a systematic literature review?, **A** - Does the study focus on the machine learning and deep learning approaches for DevSecOps?, **B** - Does the study encompass all steps in DevSecOps?

Reference	Year	Focus	Time	#P	SLR	A	B
Myrbakken and Colomo-Palacios [131]	2017	This study explores the definition, characteristics, benefits, and evolution of DevSecOps while also pointing out the challenges in its adoption.	2014-2017	52			V
Prates et al. [145]	2019	This study reviews important metrics for monitoring the DevSecOps process.	2013-2019	13			
Mao et al. [118]	2020	This study reports the state-of-the-practice of DevSecOps and calls for academia to pay more attention to DevSecOps.	2013-2019	141			V
Bahaa et al. [9]	2021	This study reviews machine learning approaches on the detection of IoT attacks.	2016-2020	49	V	V	
Leppänen et al. [100]	2022	This study reviews security challenges and practices for DevOps software development.	2015-2019	18	V		V
Akbar et al. [4]	2022	This study identifies and prioritizes the challenges associated with implementing the DevSecOps process.	-2022	46			V
Naidoo and Möller [132]	2022	This study presents a socio-technical framework of DevSecOps based on a systematic literature review.	2016-2020	26	V		V
Rajapakse et al. [153]	2022	This study reviews the challenges faced by practitioners in DevSecOps adoption, assesses solutions in the literature, and highlights areas requiring future research.	2011-2020	54	V		V
Valdés-Rodríguez et al. [187]	2023	The study reviews methods or models suitable for integrating security into the agile software development life cycle.	2018-2023	39	V		V
This study	2024	Our study introduces a landscape of state-of-the-art AI-driven security methodologies and tools tailored for DevSecOps, while also pinpointing the challenges faced by these AI-driven approaches and deriving potential avenues for future research.	2017-2023	99	V	V	V

example, defect density and defect burn rate [29] can be employed in the "Development" step to monitor the quality of code being produced and the efficiency of defect resolution over time.

Recent studies have identified several challenges when implementing the DevSecOps paradigm [4, 100]. For instance, Akbar et al. [4] pointed out challenges such as the use of immature automated deployment tools and a lack of software security awareness. Furthermore, Rajapakse et al. [153] organized challenges and solutions in their SLR based on existing literature, presenting future research opportunities in DevSecOps for unresolved problems. Additionally, Valdés-Rodríguez et al. [187] discussed current trends in existing methods for software development involving security. On the other hand, Bahaa et al. [9] concentrated on machine learning/deep learning approaches for Internet of Things (IoT) attacks, while Naidoo and Möller [132] delved into the socio-technical perspective of DevSecOps.

In contrast to previous reviews, our SLR is driven by the emerging trend of AI in security research. AI-driven security approaches offer the potential to automate time-consuming security processes, thereby alleviating the time barrier associated with integrating security into DevOps. Our objective is to categorize AI-driven security approaches, pinpoint existing challenges, and uncover future research prospects for AI-based methods in DevSecOps. To our knowledge, this paper represents one of the first SLRs focusing on AI-driven security approaches, encompassing every step of the DevSecOps process.

3 APPROACH

This systematic literature review (SLR) follows the principles outlined by Kitchenham Keele et al. [90], Kitchenham et al. [94], a framework widely adopted in the context of DevSecOps-related SLRs [4, 149, 153, 160]. Our methodology encompasses three stages: (1) formulation of the review plan, (2) execution of the review, and (3) comprehensive examination of the review outcomes. In the following sections, we introduce each of these steps in detail.

3.1 Research Questions

For a thorough understanding of AI-driven security approaches in DevSecOps, it is crucial to explore the existing AI techniques applicable to each DevSecOps step. Additionally, identifying challenges within these AI-based security techniques is essential for deriving future research directions aimed at further enhancing these techniques. Thus, this systematic literature review aims to address the following research questions:

Table 2. The overview of the selected software engineering and security conferences and journals.

Software Engineering Conference	Acronym	CORE Rank	#P
International Conference on Software Engineering	ICSE	**A***	12
Foundations of Software Engineering	FSE	**A***	12
Automated Software Engineering Conference	ASE	**A***	14
Mining Software Repositories	MSR	**A**	5
Software Analysis, Evolution and Reengineering	SANER	**A**	5
International Symposium on Software Testing and Analysis	ISSTA	**A**	1
International Conference on Software Maintenance and Evolution	ICSME	**A**	1
Evaluation and Assessment in Software Engineering	EASE	**A**	0
Empirical Software Engineering and Measurement	ESEM	**A**	1
Software Engineering Journal	**Acronym**	**Impact Factor**	**#P**
Transactions on Software Engineering	TSE	**7.4**	11
Transactions on Software Engineering and Methodology	TOSEM	**4.4**	4
Information and Software Technology	IST	**3.9**	5
Empirical Software Engineering	EMSE	**3.8**	5
Journal of Systems and Software	JSS	**3.5**	5
Security Conference	**Acronym**	**CORE Rank**	**#P**
Network and Distributed System Security Symposium	NDSS	**A***	2
Symposium on Security and Privacy	SP	**A***	1
Computer and Communications Security	CCS	**A***	5
USENIX Security Symposium	USENIX	**A***	0
Security Journal	**Acronym**	**Impact Factor**	**#P**
Transactions on Dependable and Secure Computing	TDSC	**7.3**	8
Transactions on Information Forensics and Security	TIFS	**7.2**	2

- **RQ1: What are the existing AI methods and tools employed at each stage of the DevSecOps process, and what specific security challenges do they address?**
- **RQ2: What challenges and future research opportunities exist for AI-driven DevSecOps?**

3.2 Literature Search Strategy

We follow the iterative approach outlined by Kitchenham et al. [94] to develop the search string for this study. For each of the five steps in DevOps, as defined in Section 2.1, we first identify the common security processes associated with that step. Next, we combine these security processes with AI-related terms to form the search string for AI-driven security approaches for that step. In particular, the full list of security tasks at each step of DevOps, which this study concentrates on, is presented in Figure 1. For instance, in the planning step, common security processes include threat modeling and software impact analysis. We extract the keywords "threat modeling" and "software impact analysis" with AI-related terms, resulting in a search string such as *(threat modeling OR software impact analysis) AND (AI-related terms)*. This iterative process is repeated for each step in the DevOps lifecycle. This process will result in a set of DevSecOps activity keywords and a set of AI-related keywords. The complete set of search keywords is as follows:

- *Keywords related to DevSecOps activities: Threat Modeling, Software Impact Analysis, Static Application Security Testing, SAST, Software Vulnerability Detection, Software Vulnerability Prediction, Software Vulnerability Classification, Automated Vulnerability Repair, Automated Program Repair, Dependency Management, Dependency Vulnerability, Package Management, CI/CD Secure Pipeline, Software Defect Prediction, Defect Prediction, SDP, Continuous Integration, Continuous Deployment, Configuration Validation, Infrastructure Scanning, Infrastructure as Code, IaC, Log Analysis, Anomaly Detection, Cyber-Physical Systems*

- *Keywords related to AI: Artificial Intelligence, AI, Deep Learning, DL, Machine Learning, ML, LLM, Large Language Model, Language Model, LM, Natural Language Processing, NLP, Transformer, Supervised Learning, Semi-supervised Learning, Unsupervised Learning*

After an iterative refinement process, our final search string is as follows:

> "([Threat Modeling] OR [Software Impact Analysis] OR [Static Application Security Testing] OR [SAST] OR [Software Vulnerability Detection] OR [Software Vulnerability Prediction] OR [Software Vulnerability Classification] OR [Automated Vulnerability Repair] OR [Automated Program Repair] OR [Dependency Management] OR [Dependency Vulnerability] OR [Package Management] OR [CI/CD Secure Pipeline] OR [Software Defect Prediction] OR [Defect Prediction] OR [SDP] OR [Continuous Integration] OR [Continuous Deployment] OR [Configuration Validation] OR [Infrastructure Scanning] OR [Infrastructure as Code] OR [IaC] OR [Log Analysis] OR [Anomaly Detection] OR [Cyber-Physical Systems]) **AND** ([Artificial Intelligence] OR [AI] OR [Deep Learning] OR [DL] OR [Machine Learning] OR [ML] OR [LLM] OR [Large Language Model] OR [Language Model] OR [LM] OR [Natural Language Processing] OR [NLP] OR [Transformer] OR [Supervised Learning] OR [Semi-supervised Learning] OR [Unsupervised Learning])"

We use Harzing's Publish or Perish software for our automated search process [75] with the Google Scholar search engine, aiming to gather high-quality and impactful research papers for our review. To achieve this, we target both top-tier software engineering (SE) conferences/journals and reputable software security venues. The targeted venues are summarized in Table 2. Specifically, we focus on 9 SE conferences, all of which are ranked either CORE A* or CORE A according to International CORE Conference Rankings (ICORE) [31], along with 5 SE journals with impact factors (IFs) ranging from 3.5 to 7.4. Additionally, we include 4 security conferences ranked CORE A* and 2 security journals with IFs of 7.2 and 7.3. We incorporate SE and security journals with IFs spanning from 3.5 to 7.4 to strike a balance between diversity and reputation. While journals with a higher IF enhance visibility and credibility, those with a lower IF broaden our literature search.

It is worth noting that we deliberately chose the top-tier SE and security conferences and journals to ensure the quality and impact of the studies included in our review. Our comprehensive search process led to a collection of 1,683 unique papers across 20 distinguished venues, ultimately identifying 88 studies that met our rigorous criteria and were deemed suitable for inclusion. This substantial number of papers demonstrates the thoroughness of our review and compares favorably with similar SLRs outlined in Table 1. While we acknowledge the potential for overlooking relevant studies by excluding lower-ranking venues, our deliberate emphasis on esteemed SE and security venues enables us to capture emerging trends and valuable insights from reputable sources. Notably, this approach aligns with the strategies adopted by other SLRs in the software engineering field such as [198].

This paper reviews AI-based security approaches that can be integrated into the DevOps process to achieve the DevSecOps paradigm. It is worth noting that the recent advancements of AI such as language models (LM) and large language models (LLM) stem from the transformer architecture published in 2017 by Vaswani et al. [189]. Thus, we focus our search on papers published from 2017 until the end of 2023 to examine the state-of-the-art AI-based security methods.

3.3 Literature Selection: Inclusion-Exclusion Criteria and Quality Assessment Criteria

As presented in Table 3, we have formulated three inclusion criteria (IC) and five exclusion criteria (EC) to ensure that the papers selected are qualified and highly relevant to this study. In addition, a

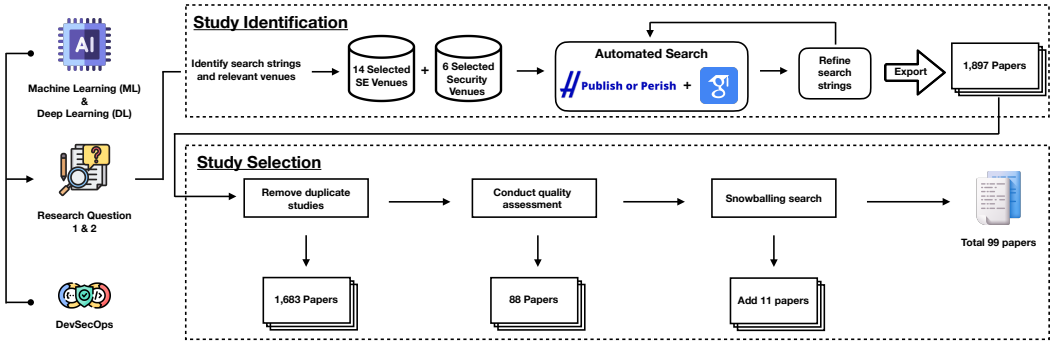

Fig. 2. The overview of our study identification and selection process.

Table 3. The inclusion-exclusion criteria and quality assessment criteria.

Inclusion Criteria	
IC-1	The paper must be peer-reviewed and published at a journal, conference, or workshop
IC-2	The paper focuses on machine learning/deep learning for security purposes for any of the step included in DevOps
IC-3	The paper with accessible full text
Exclusion Criteria	
EC-1	The paper is a duplicate or continuation of another study already included in the review
EC-2	The paper written in languages other than English
EC-3	The paper is a literature review
EC-4	The paper is a replication study
EC-5	Studies from sources such as books, theses, technical reports, monographs, keynotes, panels, or venues that do not undergo a full peer-review process
Quality Assessment Criteria	
QAC-1	The paper presents empirical results or case studies demonstrating the effectiveness of machine learning/deep learning techniques in improving security practices within DevSecOps
QAC-2	The research objective is described
QAC-3	The paper describes techniques or methodologies
QAC-4	The paper describes evaluation or validation methods
QAC-5	The paper presents the study results

well-crafted quality assessment can help prevent biases introduced by low-quality studies [217] and Kitchenham et al. [94] also suggested that such a process should be considered mandatory in any systematic review to avoid research bias. Thus, in Table 3, we further outline five quality assessment criteria (QAC) aimed at assessing the relevance, clarity, validity, and significance of included papers.

Specifically, we employ a binary scale (Yes/No) to evaluate each IC, EC, and QAC for every paper. Papers failing to meet any criteria are excluded from our study. We initially review the title and abstract of each of the 1,683 papers exported from our automated search process after deduplication. However, in some cases, we need to assess the full text to make a decision. Despite the presence of our search string terms in the title, abstract, or keywords of certain papers, it remains unclear how their content relates to the focus of our SLR. For example, while some papers appear to develop an automated security approach for a specific step in DevOps based on their title and abstract, a full-paper inspection reveals that the proposed approach is not relevant to either machine learning (ML) or deep learning (DL). Thus, to ensure that a paper is adequately aligned with the focus of our review (i.e., ML or DL-based security approaches for DevSecOps), we conducted a comprehensive full-text review following the initial assessment of titles and abstracts. By adhering to these steps,

we can effectively filter out papers that do not address ML and DL for DevSecOps. Specifically, we excluded 1,584 irrelevant papers based on our IC, EC, and QAC, resulting in a collection of 99 papers selected for this study.

3.4 Snowballing Search

To expand our search for potentially relevant primary studies, we employed a snowballing approach. This method involves not only examining the reference lists and citations of papers but also systematically tracking where papers are referenced and cited. This dual approach, known as backward and forward snowballing, allows us to thoroughly explore relevant literature beyond the initial set of papers.

Before initiating the snowballing procedure, it is essential to curate a collection of initial papers. In this study, the initial paper collection includes 99 papers after the quality assessment. We performed forward and backward snowballing with deduplication and the full study selection process. Consequently, we obtained an additional 11 papers via our snowballing search.

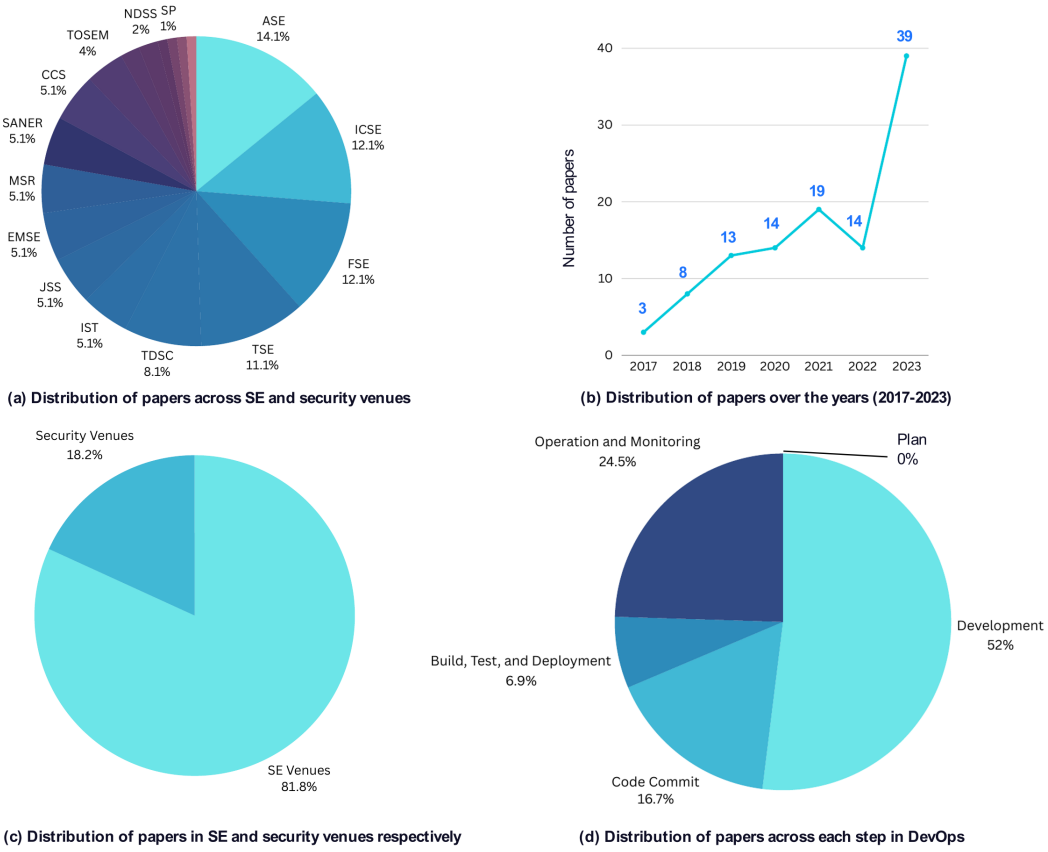

Fig. 3. The overview of the distributions of the selected 99 papers.

3.5 Data Extraction and Analysis

We obtained 99 relevant papers after searching, filtering, and snowballing. Figure 3 presents an overview of the distribution of our selected papers. All of the selected papers are peer-reviewed

by the venues listed in Table 2. Figure 3 (a) illustrates the distribution of papers across various venues, revealing ASE as the most prevalent venue with a contribution of 14% of the total, followed closely by ICSE and FSE, each accounting for 12%. TDSC follows with 8%, while IST, JSS, EMSE, MSR, SANER, and CCS each contribute 5% respectively. TOSEM contributes 4%, whereas NDSS and SP have the smallest contributions, at 2% and 1% respectively.

In Figure 3 (b), we present the paper trend over the years. Notably, the number of papers annually shows a steady linear increase from 2017 to 2022. However, there is a significant jump from 14 papers in 2022 to 39 papers in 2023. This sudden surge could be attributed to the recent advancements in generative AI and the escalating concerns surrounding software security. Figure 3 (c) presents the paper distributions of SE venues versus security venues where SE venues account for the majority of 82% of papers while security venues account for the remaining 18%.

Figure 3 (d) illustrates the distribution of papers across each step in DevOps. Notably, we found no relevant papers discussing an AI-driven security approach in the planning step of DevOps. This could be attributed to the nature of activities involved in this stage, such as threat modeling and impact analysis, which may require a higher degree of expertise and human intervention rather than relying on AI algorithms. The majority of studies, accounting for 52%, focus on the Development step in DevOps. In this step, AI-driven security approaches can directly assist software developers by detecting vulnerabilities in their source code, providing explanations, and suggesting repairs. Following this, 24% of the studies concentrate on the Operation and Monitoring step in DevOps. During this phase, AI-driven security approaches can learn from historical data and help monitor and detect anomalies occurring in software systems. Additionally, 17% of studies focus on developing AI-driven approaches for securing the Code Commit step in DevOps, while the remaining 7% concentrate on the Build, Test, and Deployment step in DevOps.

During the full-text review, we conducted data extraction to gather all relevant information necessary for a comprehensive and insightful response to our two research questions outlined in Section 3.1. This extraction phase involved collecting data on various AI-driven approaches proposed for security tasks associated with each step in DevOps, as outlined in Figure 1. With this compiled data, we systematically analyzed the relevant aspects of AI-driven approaches aimed at enhancing the security aspect of DevOps.

4 RQ1: AI METHODS OVERVIEW FOR DEVSECOPS

In this section, we introduce AI-driven security methodologies and tools for DevSecOps. We derived insights from 99 collected literature sources to address our RQ1. Each subsection focuses on AI-driven approaches for specific steps in DevOps. These steps may encompass multiple security-related tasks. Our answers to RQ1 are summarized in Table 4.

4.1 Plan

4.1.1 Threat Modeling. Threat modeling is an engineering technique employed to identify threats, attacks, vulnerabilities, and countermeasures that may impact an application. It aids in shaping the application's design, meeting the organization's security objectives, and minimizing risk [129]. In the planning step of DevSecOps, threat modeling proactively identifies security concerns and integrates security measures into the development process from the outset. Although attempts have been made to explore AI-driven approaches for threat modeling in the DevSecOps context, no relevant literature was found. Thus, in Section 6.1.1, we will introduce current threat modeling approaches such as STRIDE and DREAD and discuss their relevance to DevSecOps planning.

4.1.2 Software Impact Analysis. Impact analysis, also known as change analysis, is integral to the planning step of DevSecOps, allowing teams to assess the potential effects of proposed changes on

Table 4. (RQ1) The landscape of AI-driven approaches for security-related tasks in DevOps.

DevOps Step	Identified Security Task	AI Method	ML	DL	Reference
Plan	Threat Modeling	-	-	-	-
	Software Impact Analysis	-	-	-	-
Development	Software Vulnerability Detection	RNN		V	[37, 108, 109, 159, 197]
		TextCNN		V	[17]
		GNN		V	[19, 21, 42, 76, 107, 130, 192, 202, 223, 231, 235]
		Node2Vec		V	[220]
		LM for code		V	[40, 56, 225]
		LM + GNN		V	[40, 174]
	Software Vulnerability Classification	ML algorithms	V		[6, 194]
		RNN		V	[109, 236]
		TextRCNN		V	[41]
		Transformer		V	[191]
		LM		V	[38, 105]
		LM for code		V	[54, 57]
		LM for code + RNN		V	[137]
	Automated Vulnerability Repair	ML algorithms	V		[169]
		CNN		V	[115]
		RNN		V	[25, 133]
		Tree-based RNN		V	[106]
		GNN		V	[39]
		Transformer		V	[24, 28, 232, 233]
		LM for code		V	[13, 53, 58, 73, 86, 88, 120, 134, 140, 226, 234]
	Security Tools in IDEs	LM-based security tool		V	[57]
Code Commit	CI/CD Secure Pipelines	ML algorithms	V		[16, 23, 103, 138, 142, 177, 178, 183, 214]
		Explainable ML	V		[141]
		RNN		V	[143, 158]
		Tree-based RNN		V	[36]
		Transformer		V	[146]
		JIT-SDP tool	V		[91, 147]
		Change analysis tool	V		[121]
		LM		V	[96]
		LM for code		V	[224]
Build, Test, and Deployment	Configuration Validation	ML algorithms	V		[26, 74, 207]
		FFNN		V	[67]
		GAN		V	[10, 168]
	Infrastructure Scanning	ML algorithms	V		[35, 151, 152]
		Word2Vec-CBOW		V	[15]
Operation and Monitoring	Log Analysis and Anomaly Detection	ML algorithms	V		[30, 72, 92, 161, 204, 215]
		RNN		V	[44, 45, 104, 122, 176, 193, 204, 227, 230]
		RNN-based AE		V	[44, 221]
		GNN		V	[102]
		Transformer		V	[97]
		Explainable DL		V	[3, 71]
		Diffusion Model		V	[99]
	Cyber-Physical Systems	ML algorithms	V		[110, 205]
		RNN + GNN			[210]
		GAN		V	[209]
		VAE		V	[208]
		Transformer		V	[81]
		LM + RNN		V	[211]

Abbreviations: RNN - Recurrent Neural Network; TextCNN - Text Convolutional Neural Network; GNN - Graph Neural Network; LM - Language Model; ML - Machine Learning; TextRCNN - Text Recurrent Convolutional Neural Network; JIT-SDP - Just-in-Time Software Defect Prediction; FFNN - Feedforward Neural Network; GAN - Generative Adversarial Network; CBOW - Continuous Bag-of-Words; AE - Autoencoder; VAE - Variational Autoencoder; DL - Deep Learning.

software systems before implementation. This process involves identifying and evaluating ripple effects across various system components, including code, configuration, and dependencies [89]. Early impact analysis enables teams to anticipate and mitigate risks such as disruptions to system functionality, security vulnerabilities, and performance issues. Despite the critical role of impact analysis, no relevant literature on AI-driven security approaches was found during our literature search. Thus, in Section 6.1.2, we will introduce current impact analysis approaches and discuss their relevance to DevSecOps planning.

4.2 Development

4.2.1 Software Vulnerability Detection. Machine learning and deep learning-based vulnerability detection (VD) have been proposed to predict potential vulnerabilities in developers' source code. These detection approaches achieve improved accuracy from traditional static analysis methods without requiring the compilation of developers' code [33]. By leveraging these VD methods during development, developers can proactively identify vulnerabilities, facilitating a "shift-left" in security testing—from the testing/deployment phase to the development phase. This proactive integration aligns with the principles of DevSecOps, embodying the paradigm during the "Development" step. In what follows, we present current AI-driven VD methods, explore the challenges they have faced, and highlight potential avenues for future research.

Existing AI methods. Various AI-driven vulnerability detection (VD) methods have been proposed to predict vulnerabilities on different granularities (e.g., file, function, and line levels). Du et al. [46] used program metrics to detect vulnerabilities and compared their approach with machine learning models on the file level; Dam et al. [37] leveraged long short-term memory (LSTM) recurrent neural networks to learn both semantic and syntactic features of code and predict vulnerabilities at the file level.

On the other hand, most of the recent works focused on function-level vulnerability predictions. For instance, Russell et al. [159] developed an RNN-based representation learning approach that treated a code function as a sequence of tokens and input to sequential neural networks.

Other mainstream considered graph structures of source code such as data flow graph (DFG), control flow graph (CFG), or abstract syntax tree (AST). They treated a code function as a graph with nodes and edges and used graph neural networks (GNNs) to learn the graph representation to make vulnerability predictions. In particular, Zhou et al. [231] and Cao et al. [19] both proposed to incorporate DFG, CFG, and AST into GNNs while Chakraborty et al. [21] leveraged the code property graph (CPG) [213]. Mirsky et al. [130] proposed an enriched program dependency graph (ePDG) as the representation of their program and applied gated graph recurrent neural networks (GRNN). In addition, Zhang et al. [223] focused on the cross-project VD and built graph attention networks while Yuan et al. [220] proposed to extract the function's abstract behaviors as behavior graph and embedded such information using Node2Vec [64]. Wang et al. [192] explored the post-dominate tree and the exception flow graph. Zhang et al. [225] relied on source code pre-trained language model to embed their syntax-based CFG. Cai et al. [17] based their method on complex network analysis theory to convert the CPG into an image-like matrix and used the TextCNN model. Moreover, Wu et al. [202] employed vulnerability-specific inter-procedural slicing algorithms to capture the semantics of various types of vulnerabilities and used GNNs to learn and understand these vulnerability semantics. Finally, Steenhoek et al. [174] combined the recently advanced large language models (LLMs) with GNNs to further improve the prediction accuracy.

Nevertheless, these VD methods still operate on the function level, which may still consist of multiple lines of code that need to be manually inspected by developers. To address this issue, prior works have proposed various line-level VD approaches. It is worth noting that VulDeePecker, proposed by Li et al. [109], marks the initial stride towards fine-grained vulnerability detection. This approach introduces the concept of code gadgets, aiming to encompass a finer-grained code representation beyond program or function levels. Li et al. [107] proposed to leverage GNNs to predict on function level and used GNNExplainer [219] to interpret model predictions to locate fine-grained vulnerabilities. Wartschinski et al. [197] used Word2Vec embedding in conjunction with an RNN-based model to predict vulnerabilities within a few lines of code. In addition, Zou et al. [235] proposed a multi-granularity VD that can predict function and slice-level vulnerabilities.

Recent works have proposed AI-driven methods that can pinpoint vulnerable lines in source code. Li et al. [108] relied on intermediate code to accommodate extra semantic information with the BiRNN model to locate vulnerable lines. Fu and Tantithamthavorn [56] leveraged the self-attention mechanism in the transformer neural networks to locate line-level vulnerabilities while Hin et al. [76] leveraged GNNs and treat line-level VD as a node classification task. Similarly, Dong et al. [42] used GNNs with their subgraph embedding. Additionally, Ding et al. [40] combines the transformer model with GNNs to locate vulnerabilities on the line level. These line-level VD could save more manual effort for developers by pinpointing vulnerable lines within a code function.

4.2.2 Software Vulnerability Classification. AI-driven methods hold the promise of predicting vulnerability types by analyzing the given vulnerable source code. These predictions explain detected vulnerable source code, furnishing developers with valuable insights. By employing this approach, developers can prioritize addressing critical vulnerability types promptly. In practical terms, integrating these AI-driven automation methods directly into developers' Integrated Development Environments (IDEs) has the potential to furnish real-time vulnerability insights during the development stage. This integration aligns with the DevSecOps concept, reflecting the principle of incorporating security into the "Development" step.

Existing AI methods. We observed that a notable number of studies utilizing AI-driven methods concentrated on categorizing vulnerability types and characteristics by analyzing the input from vulnerability descriptions [6, 38, 41, 105, 191]. Nevertheless, these studies fall outside the scope of our review because information regarding vulnerability descriptions might not be accessible in the initial phases of software development.

Instead, our focus centers on AI-driven approaches that use plain source code as input to predict vulnerability types, which can explain vulnerability types by scanning developers' source code. Despite efforts, the data imbalance challenge in vulnerability classification persists. In particular, some vulnerabilities such as buffer-related errors are common while other vulnerabilities rarely occur. While Das et al. [38] incorporated data augmentation [199], the performance of their transformer model showed no significant improvement. Similarly, Wang et al. [194] addressed data imbalance by focusing on the top 10 frequency CWE-IDs, yet this approach limited the model's ability to identify rare vulnerability types.

To mitigate the data imbalance issue, Fu et al. [54] proposed a hierarchical knowledge distillation framework. The method involves grouping the imbalanced dataset into subsets based on CWE abstract types, creating more balanced subsets consisting of similar CWE-IDs. Separate TextCNN teacher models are trained for each subset, but they can only predict specific CWE-IDs within their subset. To address this limitation, a comprehensive transformer student model predicting all CWE-IDs is then developed through knowledge distillation.

On the other hand, Fu et al. [57] suggested utilizing the pre-trained language model CodeBERT [51] in conjunction with multi-objective optimization (MOO) to harness the advantages of multi-task learning for training a CWE-ID classification model. Their experiments demonstrated that incorporating multi-task learning, particularly with correlated tasks using the MOO method, can enhance the performance of vulnerability classification models.

Pan et al. [137] introduced TreeVul, an architecture designed with a hierarchical and chained structure. This design effectively uses the inherent tree structure of CWE-IDs as prior knowledge for the classification task. Notably, TreeVul provides developers with predictions for both high and low-level CWE-IDs. They leveraged CodeBERT to obtain source code representation and relied on multiple LSTM encoders to generate different levels of predictions.

Different from the approaches mentioned above, Zou et al. [236] combine vulnerability detection and type classification into an integrated multi-class detection task. Specifically, they proposed

μVulDeePecker which introduces the concept of code attention and considers control-dependence when extracting code gadget [109]. The bidirectional LSTM was trained to detect and classify 40 different vulnerability types.

4.2.3 Automated Vulnerability Repair. The progress in sequence-to-sequence (seq2seq) learning within the realm of deep learning has facilitated significant advancements, particularly in the development of AI-driven automated programs and vulnerability repair approaches. These innovative solutions now offer the capability to automatically recommend fixes for vulnerable or buggy programs, addressing the time-consuming and labor-intensive nature of manual code repair. Program repair models take source code as input, which allows for potential integration with developers' IDEs, providing near real-time code repair suggestions during the development phase. This integration automates the vulnerable code repair process and incorporates security into the "Development" step.

Existing AI methods. Several approaches have been suggested for program repair and vulnerability repair based on deep learning (DL), with program repair targeting general software defects and vulnerability repair addressing security-related weaknesses. Due to the inherent similarities between these tasks, transfer learning can be employed to enhance the model's ability to generalize across both as demonstrated by Chen et al. [24]. Consequently, the subsequent discussion encompasses a review of both DL-based program repair and vulnerability repair.

One of the initial sequence-to-sequence (seq2seq) program repair techniques is SequenceR [25], employing recurrent neural networks (RNNs) to generate repairs for defective programs. Subsequently, prior works [24, 28] proposed to use transformer architecture [189] for vulnerability repair. This architectural shift enhances repair accuracy by leveraging a global self-attention window, capturing semantic relationships within a software program more effectively than RNNs.

Recently, considerable research attention has been directed towards employing transformer-based pre-trained language models (LMs) for both program repair and vulnerability repair [13, 58, 73, 120, 140, 226, 234].

Mashhadi and Hemmati [120] focused on fixing Java programs using CodeBERT [51]. Berabi et al. [13] used T5 model pre-trained on natural language [148] to perform program repair in JavaScript while Fu et al. [58] leveraged T5 model pre-trained on code data [196] to repair C/C++ vulnerabilities and evaluated the effectiveness of LMs and different tokenizers. Hao et al. [73] and Zirak and Hemmati [234] further improved the fine-tuned model using the curricular fine-tuning strategy and deep domain adaption respectively. In addition, Pearce et al. [140] investigated the effectiveness of zero-shot large language models for vulnerability repair.

Inspired by vision transformers, Fu et al. [53] introduced a vulnerability masking technique designed to guide the vulnerability repair model towards focusing on vulnerable code blocks during the decoding process of corresponding repairs. Another approach, suggested by Zhu et al. [232], involves a syntax-guided edit decoder for program repair. This method generates edits instead of modified code, providing an efficient representation of small modifications. Namavar et al. [133] also studied the effects of different code representations for program repair. Zhu et al. [233] proposed a type-aware program repair approach to mitigate the limitation of untypable patches generated by DL models. Dinella et al. [39] represented JavaScript programs as graphs and used GNNs to produce bug fixes.

Moreover, prior works have introduced context-aware program repair approaches. In particular, Li et al. [106] proposed to learn the surrounding code contexts of the fixes while Lutellier et al. [115] presented a context-aware neural machine translation (NMT) architecture that represents the buggy source code and its surrounding context separately. Jiang et al. [86] developed a code-aware

search strategy that finds correct fixes by searching for compilable patches that are close in length to the buggy code.

Certain studies concentrate on addressing particular categories of software bugs. For instance, Utture and Palsberg [186] focused on fixing resource leak warnings while Marcilio et al. [119] generated fix suggestions in response to static code analysis warnings, and Siddiq et al. [169] focused on fixing SQL injection vulnerabilities.

Finally, it is worth noting that some studies have developed end-to-end approaches for both vulnerability detection and repair [88, 124, 134]. Mesecan et al. [124] presented HyperGI which can detect, localize, and repair information leakage. Ni et al. [134] used multi-task learning to construct a comprehensive end-to-end model for both defect prediction and program repair. Jin et al. [88] relied on a static analysis tool, infer, to detect and locate vulnerabilities, then leveraged large language models (LLMs) to generate corresponding repairs.

4.2.4 Security Tools in IDEs. Static analysis tools rely on predefined patterns to assist developers in identifying potential vulnerabilities in source code. Similarly, integrating deep learning (DL)-based vulnerability prediction approaches into security analysis tools and deploying them to developers' Integrated Development Environments (IDEs) is becoming increasingly feasible. Previous research has shown that DL models can surpass static analysis tools in terms of accuracy in detecting and locating vulnerabilities. Moreover, they exhibit the capability to identify specific vulnerability types and propose corresponding repairs. The incorporation of AI-driven security tools into IDEs streamlines developers' workflows by automating the identification of security issues in their code. Such automation integrates security considerations into the development stage, aligning with the principles of DevSecOps. Below, we explore existing AI-driven security tools within IDEs, examine the challenges they have encountered, and discuss potential avenues for future research.

Existing AI methods. We encountered a limited number of search results while exploring AI-driven software security tools. Notably, although deep learning-based approaches have demonstrated superior effectiveness compared to static analysis tools [33], the predominant tools in the current landscape continue to rely on static analysis and pre-defined patterns. Despite this trend, our investigation uncovered open-source and commercial AI-driven security tools (AIBugHunter [57] and Snyk IDE [170]) designed to assist developers in identifying security issues during development.

Fu et al. [57] introduced AIBugHunter, a deep learning-based security tool designed for C/C++ to assist developers in automating security aspects of their development process. This tool boasts several key capabilities, including the detection and pinpointing of vulnerabilities at the line level, explanation of detected vulnerability types, estimation of vulnerability severity, and suggestions for corresponding repairs. Each of these functionalities is powered by a dedicated language model. AIBugHunter operates as an open-source tool and is accessible as a VSCode extension. According to the results of a user experiment conducted by Fu et al. [57], AIBugHunter can reduce the time spent on detecting, locating, estimating, explaining, and repairing vulnerabilities from 10-15 minutes to a mere 3-4 minutes.

Snyk IDE [170] is a commercial security tool supporting 11 programming languages. Through static code scans, it provide developers with crucial information, pinpointing vulnerabilities, offering explanations of their types, and suggesting actionable fixes. Snyk is powered by DeepCode AI, which consists of multiple AI models trained on security-specific data parsed from millions of open-source projects by security researchers. Moreover, Snyk IDE provides flexible plans, including both free and paid options. It seamlessly integrates as a security plugin across various popular IDEs like JetBrains, Visual Studio Code, Eclipse, and Visual Studio.

4.3 Code Commit

4.3.1 Dependency Management. Software dependencies play essential roles in software development, coming in two types: direct, which are libraries or packages directly called by developers' code, and transitive, which are dependencies of dependencies. In the DevOps process, developers can deliver software faster and on shorter release cycles using pre-built software dependencies. However, they also introduce potential security vulnerabilities, outdated software, bugs, and legal liabilities, impacting application performance and reliability. Thus, effective dependency management is critical to minimize these risks and ensure the security and reliability of software applications [171]. Despite the critical role of effective dependency management, our review of AI-driven security approaches encountered challenges in finding relevant literature. Thus, in Section 6.2.1, we will discuss current dependency management approaches. We will highlight their relevance to the DevSecOps code commit and discuss their strengths and limitations in addressing security concerns.

4.3.2 CI/CD Secure Pipelines. In the Code Commit step of DevSecOps, ensuring the security of the Continuous Integration/Continuous Deployment (CI/CD) pipeline is crucial. A Secure CI/CD Pipeline involves implementing security measures, starting when code is committed [155]. This may include integrating security checks, such as Just-in-Time (JIT) defect prediction approaches, to identify potential vulnerabilities in code changes as soon as they are submitted. Additionally, establishing robust issue-report-to-fix-commit links enhances security by facilitating the swift resolution of reported issues, ensuring that security fixes are promptly applied to the codebase. By leveraging AI-driven techniques in JIT defect prediction and issue-report-to-fix-commit links, organizations can proactively address security concerns at the code commit stage, aligning with the principles of DevSecOps.

Existing AI methods. With the increased interest in continuous deployment, a variant of software defect prediction called Just-in-Time (JIT) Software Defect Prediction (SDP) focuses on predicting whether each incremental software change (e.g., a commit) is defective [228]. By analyzing factors such as code complexity and historical data, teams can identify potential issues before they escalate. Integrating JIT-SDP into the CI/CD pipeline enables teams to proactively detect security vulnerabilities in code commits, such as improper input validation or insecure coding practices, allowing for timely mitigation and remediation.

A majority of studies have focused on building ML models for JIT-SDP. For instance, Chen et al. [23] proposed the supervised method MULTI, which aims to reduce manual effort from developers by maximizing the number of identified buggy changes while minimizing efforts in software quality assurance activities. Similarly, Li et al. [103] leveraged semi-supervised learning with a greedy strategy in unit code inspection effort to rank changes according to their tendency to be defect-prone.

Addressing challenges such as the class imbalance problem has also been a focus. Cabral et al. [16] proposed a class imbalance learning algorithm to improve existing ML models for JIT-SDP, while Tessema and Abebe [183] suggested using change request information to further enhance ML-based JIT-SDP. Additionally, Pornprasit et al. [141] explored the explainable AI perspective of defect prediction, using rule-based explanations to explain defective predictions. Specifically, they leveraged a rule-based logistic regression technique named RuleFit [52] to build a local explainable model to explain the blackbox ML models.

Fine-grained JIT-SDP models have been proposed to locate buggy files [138] or lines [142, 214] in a commit by ranking the prediction scores of ML models. On the other hand, Qiu et al. [146] and Pornprasit and Tantithamthavorn [143] leveraged Recurrent Neural Networks (RNN) to learn semantic information of source code and locate buggy files in a commit, then localized defective

lines by ranking prediction scores. In addition, Dam et al. [36] used an abstract syntax tree (AST) to represent source code and learned tree-based LSTM networks to capture the syntax and multiple levels of the semantics of source code.

Some prior works have developed JIT-SDP tools for integration into CI/CD pipelines. For example, Qiu et al. [147] proposed JITO, integrated into Intellij IDEA, to detect defective code changes and locate exact defect lines. Similarly, Khanan et al. [91] developed JITBot, integrated into CI/CD pipelines, providing defect prediction in GitHub commits along with explanations to clarify reasons and mitigation plans. In addition, Mehta et al. [121] introduced Rex, a change analysis tool that leverages machine learning (ML) models and program analysis. Rex learns change rules that capture dependencies between different regions of code or configuration, based on patterns observed in commit logs spanning several months. When an engineer modifies a subset of files within a change rule, Rex suggests additional changes to ensure consistency and completeness. The tool has been effectively implemented within services such as Office 365 and Azure, impacting over 5,000 changes in the system.

On the other hand, issue-report-to-fix-commit links are critical for security, aiding in understanding code changes and assessing security implications. In particular, traceability links between issues and commits (i.e., issue-commit links) play a key role in software maintenance tasks such as bug localization and prediction [96]. Manual maintenance of these links is error-prone, potentially leading to vulnerabilities. Automatic link recovery methods have been proposed, but traditional classifiers such as Relink [203] may struggle due to limited positive links and dependency on their number for generating negative links. To address this, Sun et al. [178] formulated the missing link problem as a model learning problem and trained a machine learning (ML) classifier. In contrast, Ruan et al. [158] proposed leveraging recurrent neural networks (RNNs) to learn the semantic representation of natural language descriptions and code in issues and commits, and the semantic correlation between issues and commits. Recently, Lan et al. [96] suggested using language models such as BERT to further enhance performance. However, Zhang et al. [224] argued that the size of language models is too large and proposed distilling knowledge from the CodeBERT model to build a smaller model while maintaining competitive performance.

Prior works have also proposed multiple AI-driven approaches to identify security-related commits automatically. For instance, Suh [177] used machine learning (ML) models such as Support Vector Machine (SVM) to predict whether a commit is likely to be reverted based on features extracted from the revision history of a codebase. Given the batch nature of continuous integration deployment at scale, developers can find time-sensitive bugs in production more quickly.

4.4 Build, Test, and Deployment

4.4.1 Configuration Validation. Configuration validation is a critical aspect of DevSecOps. It ensures that the configurations of software systems, including parameters and settings, are accurate, optimal, and secure. Common approaches to configuration validation include manual inspection, automated testing, and performance estimation models. The importance of configuration validation lies in its direct impact on system reliability, performance, and security. Misconfigurations can lead to vulnerabilities and system failures, hence making robust validation processes essential for safeguarding software systems against potential threats [212].

Existing AI methods. Manual configuration tuning in complex software systems presents a significant security challenge due to the extensive array of parameters for users to configure. However, understanding the intricacies of the software required for effective tuning often exceeds typical user capabilities [10]. This knowledge gap increases the risk of misconfigurations, leaving

systems vulnerable to security breaches. To address this challenge, leveraging AI-driven performance estimation can provide valuable insights into the impact of various configuration settings on system performance and security, empowering users to make informed decisions.

In particular, deep learning-based approaches have been proposed. Ha and Zhang [67] proposed DeepPerf, a simple feedforward neural network (FFNN) to predict performance values of highly configurable software systems. Shu et al. [168] suggested employing generative networks with adversarial learning, comprising a generator and a discriminator, which iteratively refine the prediction model through competition until its predicted values converge towards the ground truth distribution. Additionally, Cheng et al. [26] proposed to combine multi-objective optimization and a performance prediction model to search for an optimal configuration for Spark deployed in the public cloud. Recently, Xia et al. [207] introduced CoMSA, a Modeling-driven Sampling Approach based on XGBoost, which prioritizes configurations with uncertain performance predictions for further training. CoMSA is adaptable to scenarios with or without historical performance testing results because not all software projects maintain such records. Instead of solely focusing on improving the performance estimation models, Bao et al. [10] proposed ACTGAN which aims to capture hidden structures within good configurations and use this knowledge to generate potentially better configurations.

On the other hand, container orchestrators (CO) are crucial for managing container clusters in virtualized infrastructures. Securing CO is challenging due to numerous configurable options and manual configuration is prone to errors and time-consuming. Thus, Haque et al. [74] proposed KGSecConfig, a machine learning-based approach for automating the security configuration of Container Orchestrators (CO), such as Kubernetes, Docker, Azure, and VMWare. It leverages knowledge graphs to systematically capture, link, and correlate heterogeneous and multi-vendor configuration options into a unified structure. By employing keyword and learning models, KGSecConfig enables the automated extraction of secured configuration options and concepts, aiding in the mitigation of misconfigurations in CO environments.

4.4.2 Infrastructure Scanning. In DevSecOps, Infrastructure Scanning plays a crucial role in ensuring the security and compliance of software systems. With the rise of Infrastructure as Code (IaC) tools like Ansible, Chef, and Puppet, the process of provisioning and configuring infrastructure has become more automated and scalable. These tools allow developers and operations teams to define infrastructure configurations as machine-readable code, enabling consistent, repeatable deployments across environments. Thus, IaC is a key DevOps practice and a component of continuous delivery [126]. However, during the development of IaC scripts, practitioners might unknowingly introduce security smells (e.g., hard-coded passwords). These recurring coding patterns signal security weaknesses that could lead to security breaches [150]. By integrating AI-driven IaC methods, organizations may proactively identify and mitigate security risks in their IaC scripts. This saves developers' manual security inspection efforts and strengthens the overall security posture of their systems.

Existing AI methods. Similar to other source code artifacts, Infrastructure as Code (IaC) scripts may contain defects that hinder their proper functionality. To automate the defect prediction process and reduce manual inspection, Rahman and Williams [151] leveraged text-mining techniques, such as Bag-of-Words (BoW) and TF-IDF, to extract features from IaC scripts and predict defective ones using machine learning (ML) models. They evaluated their method on Puppet IaC scripts. Rahman and Williams [152] further conducted qualitative analysis on defect-related commits extracted from open-source software repositories to identify source code characteristics correlated to defective IaC scripts. They then surveyed practitioners to gauge their agreement with the identified characteristics, using them as features to construct their ML IaC defect prediction model. Similarly, Dalla Palma

et al. [35] also suggested an ML-based technique to predict defects in IaC scripts. Their models rely on various metrics, such as lines of code, IaC-specific metrics like the number of configuration tasks, and process metrics such as the number of commits to a file, computed from the collected IaC scripts to predict their proneness to failure. The study was particularly implemented and targeted for Ansible-based projects.

On the other hand, Borovits et al. [15] focused on the linguistic anti-patterns in IaC. Linguistic anti-patterns are recurring poor practices concerning inconsistencies in the naming, documentation, and implementation of an entity. They impede the readability, understandability, and maintainability of source code. In particular, they proposed FINDICI, a deep learning (DL)-based approach for anti-pattern detection in IaC. They build and use the abstract syntax tree of IaC code units to create code embeddings used by DL models to detect inconsistent IaC code units. They also evaluated their approach using Ansible-based projects.

4.5 Operation and Monitoring

4.5.1 Log Analysis and Anomaly Detection. In this context, AI-driven approaches play a crucial role. Organizations leverage AI techniques to detect and mitigate anomalies in system logs effectively. Furthermore, explainable AI (XAI) enhances this capability by providing insights into the root causes of anomalies and facilitating informed decision-making. The application of AI-driven anomaly detection extends beyond system logs to encompass cloud services, thereby strengthening operations and aligning seamlessly with the principles of the DevSecOps paradigm. In the following sections, we present our literature review focusing on these tasks.

Existing AI methods. Some studies have concentrated on machine learning (ML) models such as support vector machines (SVM) due to their lower computation and time requirements compared to deep learning (DL) models. For instance, Khreich et al. [92] integrated frequency and temporal information from system call traces using a one-class SVM, which preserves temporal dependencies among these events. Moreover, Cid-Fuentes et al. [30] observed that certain anomaly detectors rely on historical failure data and cannot adapt to changes in system behavior at runtime. To address this limitation, they developed a model of system behavior at runtime using SVM, thereby eliminating the need for historical failure data and enabling adaptation to behavior changes. Similarly, Han et al. [72] employed online SVM to adapt to noise in system log data. More recently, Yang et al. [215] used traditional Principal Component Analysis (PCA) and achieved comparable effectiveness to advanced supervised/semi-supervised DL-based techniques while demonstrating better stability under insufficient training data.

Due to recent advancements in deep learning (DL), numerous studies on log-based anomaly detection have introduced various DL-based approaches. For instance, Du et al. [45] proposed DeepLog, one of the pioneering Recurrent Neural Network (RNN) models that treat system logs as natural language sequences to autonomously learn patterns from normal execution log files. Following this, several other RNN-based detectors emerged, including those by Zhang et al. [227], who leveraged attention-based Bi-LSTM, and Meng et al. [122], who relied on the same architecture with their proposed template2Vec to extract the semantic and syntax information from log templates. Studiawan et al. [176] employed Gated Recurrent Units (GRU) alongside sentiment analysis to identify negative sentiment indicative of anomalous activities in operating system (OS) logs. Furthermore, Zhou et al. [230] delved into sentence embedding with event metadata to glean syntactic information from log data, while Wang et al. [193] introduced an online learning paradigm and used LSTM to handle incoming new and unstable log data. In contrast, Yuan et al. [221] leveraged LSTM-based autoencoder to reconstruct discrete event logs and showed that their

approach can detect not only sequences that include unseen or rare events, but also structurally abnormal sequences.

Du et al. [44] focused on the challenge of continuously updating anomaly detection systems with new information over time. They introduced a lifelong learning framework called unlearning, which adjusts the model upon labeling false negatives or false positives post-deployment. This framework addressed both the challenge of exploding loss in anomaly detection and catastrophic forgetting in lifelong learning. Furthermore, Li et al. [104] identified challenges in analyzing interleaved logs in modern distributed systems. They propose SwissLog as a solution to these challenges. The issues include the absence of log dependency mining, variability in log formats, and difficulty in non-intrusive performance issue detection. SwissLog tackles these by constructing ID relation graphs, grouping log messages by IDs, using an online data-driven log parser, and applying an attention-based Bi-LSTM model and heuristic searching algorithm for anomaly detection and localization.

Some existing approaches, as discovered by Le and Zhang [97], rely on a log parser to convert log messages into log events, which are then used to create log sequences. However, errors in log parsing can negatively impact the performance of anomaly detectors based on unsupervised or supervised machine learning models. Thus, they introduced NeuralLog, which employs a transformer architecture and eliminates the need for log parsing. They used subword tokenization to address the out-of-vocabulary (OOV) issue. Instead of treating system logs as natural language sequences which might reduce anomaly detectors' sensitivity to the log flaws and the concurrency of multiple anomalies, Li et al. [102] transformed log record sequences into log event graphs using event semantic embedding and event adjacency matrix. An attention-based Gated Graph Neural Network (GGNN) model was then used to capture semantic information for anomaly identification. Finally, Wu et al. [204] presented a comprehensive study investigating the effectiveness of different representations used in machine learning and deep learning models for log-based anomaly detection.

Some studies have delved into explainable AI (XAI) for anomaly detection in securing software systems. For instance, Han et al. [71] introduced an interpretation methodology tailored for unsupervised deep learning models specifically designed for security systems. Their approach formulates anomaly interpretation as an optimization problem, seeking to identify the most significant differences between anomalies and a normal reference. Furthermore, the interpretations underwent validation through feedback from human security experts. Additionally, Aguilar et al. [3] proposed a decision tree-based autoencoder aimed at anomaly detection, which offers insights into its decisions by exploring correlations among various attribute values.

Given the complexity of managing diverse services in cloud environments, there is a need for automated anomaly detection mechanisms that are easy to set up and operate without requiring extensive knowledge of individual services [161]. For instance, Sauvanaud et al. [161] used machine learning models to aid providers in diagnosing anomalous virtual machines (VMs). Recently, Lee et al. [99] proposed Maat, a framework for anticipating cloud service performance anomalies based on a conditional diffusion model [77]. Maat adopts a two-stage paradigm for anomaly anticipation, consisting of metric forecasting and anomaly detection on forecasts. It employs a conditional denoising diffusion model for multi-step forecasting and extracts anomaly-indicating features based on domain knowledge, followed by the application of isolation forest [112] with incremental learning to detect upcoming anomalies, thus uncovering anomalies that better conform to human expertise.

4.5.2 Cyber-Physical Systems. Cyber-physical systems (CPS) integrate sensing, computation, control, and networking into physical objects and infrastructure, establishing connections among them

and with the Internet to facilitate seamless interaction and automation [135]. Given the critical need for high security in CPS to ensure safe operation, anomaly detection, relying on data analysis and learning, emerges as a key security technology. The principles of DevSecOps, prioritizing security at every stage of software development, intersect with CPS, particularly when software interacts with physical systems or infrastructures. In such scenarios, the integration of AI-driven approaches for anomaly detection and security enhancement in CPS aligns with DevSecOps goals, aiming to seamlessly integrate security into the development and deployment processes.

Existing AI methods. Prior works have proposed leveraging Digital Twins, which are digital replicas of physical entities [48], to train ML and DL models for anomaly detection. Digital twins are particularly useful for anomaly detection in cyber-physical systems (CPS) due to their ability to create virtual replicas of physical systems. For example, LATTICE [210] is a digital twin-based anomaly detection method employing deep curriculum learning. It assigns difficulty scores to each sample and uses a training scheduler to sample batches of training data based on these scores, facilitating learning from easy to difficult data. This approach has shown to be more effective than their previous DL-based proposal, ATTAIN [209]. Additionally, Xu et al. [211] identified challenges related to data complexity and insufficiency within CPS, particularly in train control and management systems. Consequently, they proposed employing a language model (LM) with an LSTM architecture to understand complex data, supplemented by a knowledge distillation technique to learn from out-of-domain datasets, addressing the issue of data insufficiency. On the other hand, Xi et al. [205] found that existing anomaly detection methods in CPS, such as AutoEncoder (AE) [11] or Generative Adversarial Network (GAN) [162], often overlook implicit correlations between data points, like the relationship between vehicle speed and obstacle position in the Intelligent Cruise Control System (ICCS), resulting in suboptimal performance. Hence, they proposed an adaptive unsupervised learning method incorporating a Gaussian Mixture Model (GMM), dynamically constructing and updating data correlations via KNN and dynamic graph techniques. Lin et al. [110] focused on Industrial Control Systems (ICS) such as water and power and leveraged the Bayesian network to discover dependencies between sensors and actuators and recognize irregular dependencies.

On the other hand, microservice anomaly detection is vital for system reliability in a microservice architecture. Xie et al. [208] focused on anomaly detection in traces within a microservice architecture. Traces record inter-microservice invocations and are essential for diagnosing system failures. They suggested a group-wise trace anomaly detection algorithm, which categorizes traces based on shared sub-structures and employs a group-wise variational autoencoder (VAE) to obtain structural representations, effectively reducing system detection overhead and outperforming existing methods that analyze each trace individually without considering the structural relationships between them. Huang et al. [81] claimed that the main challenge arises from integrating multiple data modalities (e.g., metrics, logs, and traces) effectively. To address this, they proposed extracting and normalizing features from metrics, logs, and traces, integrating them using a graph representation called MST (Microservice System Twin) graph. A transformer architecture with spatial and temporal attention mechanisms is then employed to model inter-correlations and temporal dependencies, enabling accurate anomaly detection.

5 RQ2: CHALLENGES AND RESEARCH OPPORTUNITIES IN AI-DRIVEN DEVSECOPS

In the previous section, we reviewed existing AI-driven security methodologies and tools for DevSecOps to address our RQ1. Now, we will introduce themes of challenges encountered by prior studies and derive future research opportunities to answer our RQ2. To begin, we found that some of these challenges are shared across multiple security tasks related to the DevOps process. Thus,

Table 5. (RQ2) The overview of the 15 challenges and future research opportunities derived from previous studies.

DevOps Step	Identified Security Task	Themes of Challenges	Research Opportunity
Plan	Threat Modeling	-	-
	Software Impact Analysis	-	-
Development	Software Vulnerability Detection	C1-1 - Data Imbalance	R1-1 - Data augmentation and logit adjustment
		C4 - Cross Project	R4 - Evaluate cross-project SVD with diverse CWE-IDs
		C5 - MBU Vulnerabilities	R5 - Evaluate SVD on MBU vulnerabilities
		C6 - Data Quality	R6 - Address data inaccuracy from automatic data collection.
	Software Vulnerability Classification	C1-2 - Data Imbalance	R1-2 - Meta-learning and LLMs
		C7 - Incompleted CWE Tree	R7 - Develop advanced tree-based SVC
	Automated Vulnerability Repair	C2-1 - Model Explainability	R2-1 - Evidence-based explainable AI (XAI)
		C8 - Sequence Length and Computing Resource	R8 - Explore transformer variants that can process longer sequences
		C9 - Loss of Pre-Trained Knowledge	R9 - Explore different training paradigms during fine-tuning
		C10 - Automated Repair on Real-World Scenarios	R10 - Address limitations of LLMs
	Security Tools in IDEs	C3-1 - Lack of AI Security Tooling in IDEs	R3-1 - AI tool deployment and comprehensive tool evaluation
Code Commit	CI/CD Secure Pipelines	C2-2 - Model Explainability	R2-2 - Explainable AI (XAI) for DL Models
		C3-2 - Lack of AI Security Tooling in CI/CD	R3-2 - AI tool deployment in CI/CD pipelines
		C11 - The Use of RNNs	R11 - Explore LMs and LLMs
Build, Test, and Deployment	Configuration Validation	C12 - Complex Feature Space	R12 - Explore transformers for tabular data
	Infrastructure Scanning	C3-3 - Lack of AI Security Tooling for Infrastructure Scanning	R3-3 - AI tool deployment and post-deployment evaluation
		C13 - Manual Feature Engineering	R13 - Explore DL-based techniques
Operation and Monitoring	Log Analysis and Anomaly Detection	C2-3 - Model Explainability	R2-3 - Explainable AI (XAI) for ML Models
		C14 - Normality Drift for Zero-Positive Anomaly Detection	R14 - Enhance normality drift detection
	Cyber-Physical Systems	C15 - Monitoring Multiple Cyber-Attacks Simultaneously	R15 - Distributed anomaly detection and multi-agent systems

we will first illustrate these common challenges along with their associated research opportunities in the following section. After that, we will proceed to introduce challenges and opportunities specific to each security task in the DevOps process. For clarity, we will use "**C**" followed by a number to represent a challenge and "**R**" followed by a number to represent the corresponding research opportunity. Our answers to RQ2 are summarized in Table 5.

5.1 Common Challenges

We identified three common challenges: (C1) Data Imbalance, (C2) Model Explainability, and (C3) Lack of AI Security Tooling. Based on our investigation of previous literature, we found that these challenges are shared by multiple security tasks related to the DevOps process. In what follows, we introduce these common challenges and discuss the associated research opportunities.

C1-1 - Data Imbalance in Software Vulnerability Detection (In DevOps Development). Chakraborty et al. [21] observed that the performance of deep learning (DL)-based VD approaches could drop 73% of the F1-score due to the data imbalance issue. Thus, Yang et al. [218] further investigated the impact of data sampling on the effectiveness of existing state-of-the-art (SOTA) DL-based VD approaches. Their discovery revealed that, in DL-based VD, employing over-sampling proves more beneficial than under-sampling. Despite this observation, their experimental findings indicate a persistent challenge: a notable proportion of cases (ranging from 33% to 58%) where decisions were not determined by the presence of vulnerable statements. Consequently, the issue of data imbalance persists, with models continuing to prioritize non-vulnerable code statements when arriving at a vulnerable decision.

R1-1 - Data Augmentation and Logit Adjustment. Regarding the research opportunities of data imbalance, Yang et al. [218] suggested that future research explore data augmentation, emphasizing its potential value. Their results indicated that employing a straightforward repetition strategy could enhance the performance of models. Furthermore, the issue of data imbalance is also recognized in the computer vision domain, and proven methods like logit adjustment [123] have demonstrated effectiveness in addressing imbalances in image classification. The application of such methods holds the potential to improve the performance of vulnerability detection.

C1-2 - Data Imbalance in Software Vulnerability Classification (In DevOps Development). We found that current vulnerability classification methods still suffer from the data imbalance challenge where models have limited performance on the vulnerability types that infrequently occur. For instance, VulExplainer [54] can correctly identify 67%-69% for common CWE-IDs while the performance drops to 49%-56% for rare CWE-IDs. Moreover, the MOO-based vulnerability

classification [57] could not correctly identify some of the infrequent vulnerabilities such as CWE-94 (Improper Control of Generation of Code).

R1-2 - Meta-Learning and LLMs. Fu et al. [54] showed that their VulExplainer approach outperformed the commonly used data imbalance techniques such as focal loss [111] and logit adjustment [123]. However, the performance on infrequent vulnerability types still has plenty of room to improve. Thus, future research should explore other techniques to address the data imbalance issue. For instance, meta-learning involves training models to learn a higher-level strategy or set of parameters that enable them to quickly adapt to new, unseen tasks with limited data. It might be suitable for imbalanced data because the exposure to diverse tasks during meta-training helps the model generalize effectively, allowing it to adapt to tasks with imbalanced class distributions. The transfer of knowledge across tasks and the few-shot learning nature of meta-learning contribute to improved performance in scenarios where certain classes have limited samples. Such an approach could also be integrated with few-shot learning of large language models (LLMs).

C2-1 - Model Explainability in Automated Vulnerability Repair (In DevOps Development). The advancement of language models has dramatically improved the accuracy of programs and vulnerability repair due to the substantial model size and training data. While recent studies focus on performance improvement, the predictions offered by those models are not explainable, posing challenges in establishing trust between the models and users. As highlighted by Winter et al. [201], trust in program repair is a crucial problem, in their empirical study involving Bloomberg developers, the sentiment conveyed was that an automated repair tool should demonstrate its reliability and foster trust with developers.

R2-1 - Evidence-based XAI. Given the complex structure of Large Language Models (LLMs), characterized by multiple hidden layers and a large number of parameters, developing intrinsically explainable AI to explain their repair predictions poses a significant challenge. Nonetheless, an avenue worth exploring involves leveraging the model's self-attention mechanism to ascertain if it can offer meaningful explanations. In addition, future studies could delve into evidence-based explainable AI (XAI), wherein the repair model not only presents its generated fix to end users but also showcases a similar repair case from its training data. This approach aims to establish trust with users, drawing inspiration from its successful application in explaining the story point estimation model in agile software development [55].

C2-2 - Model Explainability in Software Defect Prediction (In DevOps Code Commit). Our investigation revealed that the majority of just-in-time (JIT) software defect prediction (SDP) methods based on deep learning (DL) primarily emphasize enhancing performance and granularity [138, 142, 143, 214]. While more accurate and finer-grained SDP methods are beneficial for constructing robust and cost-effective DL-based SDP solutions, there is a noticeable lack of attention to the explainability of these DL-based SDPs. This lack of focus on explainability presents a challenge to the trustworthiness of DL-based SDPs.

R2-2 - XAI for DL Models. Given the complexity of DL models, explaining them is more challenging compared to machine learning (ML) models. Nonetheless, future studies could explore the use of extrinsic explanations such as Layer Integrated Gradient (LIG) [180], DeepLift [5, 167], DeepLiftSHAP [114], and GradientSHAP [114], which rely on gradients or their approximations to assess feature importance. Additionally, intrinsic explanations, such as the self-attention outputs from transformer architectures, could also highlight significant features contributing to model predictions. Finally, exploring case-based reasoning [1], which uses similar predictions retrieved from the training data as supporting evidence to explain model predictions, is another promising avenue to consider.

C2-3 - Model Explainability in Log Analysis and Anomaly Detection (In DevOps Operation and Monitoring). Our investigation reveals that while the majority of AI-driven anomaly detection approaches focused on performance improvement, only a few studies [3, 71] prioritized model explainability. Explainable AI (XAI) is crucial for anomaly detection models due to several reasons according to Han et al. [71]. Firstly, without detailed explanations for system decisions, security operators struggle to establish trust, leading to unreliable outputs. Secondly, the black-box nature of deep learning models makes it difficult to diagnose and address system mistakes, hindering effective troubleshooting. Additionally, integrating human expertise into security systems is challenging with opaque models, limiting human-in-the-loop capabilities and feedback incorporation. Thus, the lack of focus on explainability presents a significant challenge for AI-driven anomaly detection systems.

R2-3 - XAI for ML Models. To enhance the explainability of machine learning (ML) models in anomaly detection, future research can explore various interpretability methods. Approaches such as SHAP [114], LIME [156], and Breakdown [63] have demonstrated success in tasks like software defect prediction for understanding model decisions [87]. Additionally, recent advancements in explainable AI, such as the AIM framework proposed by Vo et al. [190], show potential for outperforming traditional XAI approaches like LIME and L2X, particularly in tasks like sentiment analysis. On the other hand, causal inference holds the potential to make machine learning models more interpretable [95]. By determining cause-and-effect relationships between variables, causal inference techniques offer insights into AI-driven anomaly detection. Overall, integrating causal inference methods into anomaly detection frameworks holds promise for enhancing model interpretability. By uncovering causal relationships between input features and detected anomalies, these techniques offer deeper insights into system behavior, enabling more informed decision-making by software developers and security analysts.

C3-1 - Lack of AI Security Tooling in IDEs (In DevOps Development). Challenge - Lack of AI-driven Tools in IDEs A significant hurdle in the current landscape is the limited availability of AI-driven security tools in developers' IDEs, posing a challenge to the widespread adoption of advanced security measures in software development. The scarcity of such tools restricts developers from harnessing the full potential of artificial intelligence in enhancing security practices. Additionally, there is a notable absence of a comprehensive user study that thoroughly evaluates the performance and effectiveness of existing AI-driven security tools, including prominent ones like AIBugHunter [57] and Snyk [170]. The lack of a comprehensive user study hinders a nuanced understanding of the strengths, weaknesses, and overall impact of these tools, impeding efforts to establish robust security practices in the evolving landscape of software development.

R3-1 - AI Tool Deployment and Comprehensive Tool Evaluation. We outline potential avenues for research in the domain of AI-driven security tools. Considering the extensive literature and varied techniques available for vulnerability detection, classification, and repair, future studies could focus on seamlessly integrating existing methods into developers' IDEs to enhance their practical applicability. For instance, commercial tools such as Code Scanning Autofix powered by GitHub Copilot are available in GitHub to help developers automatically fix vulnerable code in their pull requests. This tool is an expansion of Code Scanning that provides users with targeted recommendations to help them fix code scanning alerts in pull requests and avoid introducing new security vulnerabilities. The potential fixes are generated automatically by large language models (LLMs) using data from the codebase, the pull request, and from CodeQL analysis [182]. In particular, when a vulnerability is discovered in supported languages (i.e., JavaScript, TypeScript, Python, and Java), Autofix will generate a natural language explanation of the suggested fix, along with a preview of the code suggestion that the developer can accept, edit, or dismiss. Moreover, these code suggestions can include changes across multiple files and the dependencies that should be added

to the project [60]. Furthermore, a comprehensive evaluation of these AI-driven security tools is imperative. This involves soliciting and analyzing user feedback, particularly the scenarios where the tools generate inaccurate predictions. Furthermore, it is crucial to investigate the robustness of AI-driven security tools. For example, a robust security tool should ideally predict the repaired program as benign. However, the existing literature does not definitively address whether a program continues to be predicted as vulnerable even after undergoing a successful repair. Exploring this aspect would contribute valuable insights into the efficacy of AI-driven security tools in practical scenarios.

C3-2 - Lack of AI Security Tooling in CI/CD (In DevOps Code Commit). Our investigation revealed that the majority of just-in-time (JIT) SDP tools used in CI/CD environments, such as JITO [147] and JITBot [91], primarily rely on machine learning models. Despite proposed deep learning (DL)-based SDP methodologies, there is a notable absence of DL-based tools integrated into CI/CD pipelines. This absence impedes the practical adoption of DL-based approaches.

R3-2 - AI Tool Deployment in CI/CD Pipelines. In contrast to machine learning (ML) models, which depend on manually predefined metrics for predicting software defects, DL models can learn source code representations directly from developers' code without the need for extensive feature engineering efforts. To enhance their practical adoption, future research should focus on integrating existing DL-based approaches for JIT SDP into CI/CD pipelines. Additionally, conducting a comprehensive evaluation of these DL-based tools is essential. This evaluation should encompass not only performance assessment post-deployment but also a large-scale user study to gather feedback from developers.

C3-3 - Lack of AI Security Tooling for Infrastructure Scanning (In DevOps Build, Test, and Deployment). Our investigation has uncovered a gap between AI-driven infrastructure scanning approaches and their practical adoption. Although several machine learning (ML)-based methods have been proposed to detect defective Infrastructure-as-Code (IaC) scripts [15, 35, 151, 152], they have yet to be deployed as software tools for developers. The lack of deployment hinders their practical adoption. Furthermore, the precision and practicality of these tools post-deployment remain unexplored.

R3-3 - AI Tool Deployment and Post-Deployment Evaluation. Future research should focus on bridging the gap between proposed AI-driven methods for detecting defective Infrastructure-as-Code (IaC) scripts and their practical deployment as software tools for developers. Exploring strategies to facilitate the deployment of these tools, such as developing user-friendly interfaces and integration into existing development workflows, could enhance their practical adoption. Additionally, investigating the precision and practicality of these tools post-deployment is crucial to assess their effectiveness in real-world scenarios. By addressing these research opportunities, advancements can be made towards empowering developers with effective and efficient tools for enhancing the security of Infrastructure-as-Code.

Below, we begin introducing the challenges and research opportunities specific to each security task in the DevOps process. Since we found no relevant literature discussing AI-driven approaches in the planning step of DevOps, we will start with the development step.

5.2 Development

5.2.1 Software Vulnerability Detection. While the progress in code pre-trained language models enhances the F1-score of function-level vulnerability detection, reaching up to 96.5% [174], there remain several challenges that must be addressed in the current landscape of AI-driven vulnerability detection. We describe challenges and potential research directions in the following.

C4 - Cross Project. Most of the VD studies considered the mixed project (models are trained and tested on combined projects) scenario when evaluating their proposed approaches. However,

in an empirical study evaluating SOTA DL-based VD approaches, Steenhoek et al. [175] observed a decline in model performance during detection under the cross-project scenario (models are trained on one set of projects and tested on completely different, non-overlapping projects). Specifically, the F1-score of function-level detection models experienced a reduction ranging from 11% to 32%. This underscores the challenge posed by cross-project vulnerability detection and emphasizes the limited generalizability of current methodologies. The findings emphasize the need for advancements in methodologies capable of generalizing effectively across various projects, especially in cross-project scenarios.

R4 - Cross-Project SVD with Diverse CWE-IDs. Zhang et al. [223] and Liu et al. [113] both proposed to use deep domain adaptation to address cross-project VD in C languages. Subsequent research avenues could delve into cross-programming language VD and explore cross-project VD for other programming languages. In particular, Liu et al. [113] mainly focused on CWE-119 and CWE-399 while other dangerous CWE-IDs [34] should be considered in future studies. Furthermore, the study concentrated on graph attention networks, yet there is potential for exploration of other graph neural networks (GNNs) and large language models (LLMs) for effective cross-project VD.

C5 - MBU Vulnerabilities. Sejfia et al. [163] pointed out that state-of-the-art deep learning-based vulnerability detection (VD) approaches concentrate on individual base units, assuming that vulnerabilities are confined to a single function. However, vulnerabilities may extend across multiple base units (MBU). They found that existing DL-based detectors do not work as well in detecting all comprising parts of MBU, which poses a challenge in predicting MBU vulnerabilities. Thus, future research should contemplate adapting their evaluation methods to incorporate the presence of MBU vulnerabilities.

R5 - Evaluate SVD on MBU Vulnerabilities. The majority of studies on function-level VD [21, 27, 56] have typically assessed performance based on the number of correctly detected vulnerable functions. However, it is important to note that a single vulnerability might encompass multiple vulnerable functions. Therefore, merely identifying one vulnerable function does not necessarily equate to the comprehensive detection of the entire vulnerability. Acknowledging this, Sejfia et al. [163] emphasized the need for future research to account for the scenario of Multiple Base Unit (MBU) vulnerabilities during the training and testing phases of DL-based VD. This requires refining evaluation metrics and methodologies to align with MBU vulnerabilities. Consequently, there is a critical call for an enhanced understanding and consideration of MBU vulnerabilities to further advance the field of vulnerability detection.

C6 - Data Quality. Finally, Croft et al. [32] noted data quality concerns related to accuracy, uniqueness, and consistency in widely used vulnerability datasets. Their findings revealed that a substantial portion, ranging from 20% to 71%, of labels in real-world datasets were inaccurately assigned. This inaccuracy had the potential to significantly impact the performance of resulting models by up to 65%.

R6 - Addressing Data Inaccuracy from Automatic Data Collection. Croft et al. [32] pointed out that automatic data collection often leads to data inaccuracy. Common vulnerability datasets [50, 229, 229] were constructed by using the code changes information to recover the vulnerable version of a method. However, the vulnerability fixing commits or vulnerable lines could be incorrectly selected. Thus, it is important for future research to devise semantic filters or heuristics—methods or rules designed to analyze and interpret the meaning of code changes. This is crucial for precisely pinpointing lines that signify vulnerability fixes, and addressing the data inaccuracies stemming from the automatic data collection process.

5.2.2 Software Vulnerability Classification. In our examination of AI-driven methods explaining vulnerability types to developers, we found that a significant amount of studies focus on vulnerability

detection, with limited attention given to vulnerability classification. Below, we outline the challenge identified in current AI-driven vulnerability classification approaches followed by future research opportunities.

C7 - Incompleted CWE Tree. The TreeVul approach [137] currently relies exclusively on parent-child relations within the Common Weakness Enumeration (CWE) hierarchy for conducting top-down searches. This approach excludes the consideration of other potentially valuable relations, such as PeerOf, which could enhance the model's effectiveness. In addition, TreeVul only considers CWE categories with depth<=3 while some important categories may be located at depth>3. These incomplete considerations of CWE tree structure may hinder TreeVul's ability to generalize to additional CWE-IDs.

R7 - Advanced Tree-based SVC. According to Pan et al. [137], TreeVul did not encompass the entire CWE tree. Consequently, future investigations may broaden the TreeVul approach by incorporating extra relations, such as PeerOf. Nevertheless, this procedure views the CWE structure as a graph, introducing a higher level of complexity compared to the hierarchical tree structure considered by Pan et al. [137]. Thus, emphasis could be placed on streamlining the transformation process to reduce the complexities of converting the CWE tree into a graph. Furthermore, optimizing the TreeVul approach to dynamically determine the appropriate level for concluding top-down searches beyond the predefined depth-3 CWE categories offers an opportunity to enhance the model's adaptability. This optimization could involve developing a mechanism to assess confidence levels, allowing the model to automatically adjust its search depth based on contextual factors for each specific input. Such advancements could improve the precision and flexibility of AI-driven vulnerability classification.

5.2.3 Automated Vulnerability Repair. Recent advancements in transformer architecture and pre-trained language models for code have shown improvements over RNN-based models in vulnerability repair and program repair tasks. Nevertheless, beyond model architecture and performance, there are additional aspects that demand attention and further improvement, especially when considering their integration into real-world projects. Below, we present challenges derived from previous studies with potential research opportunities.

C8 - Sequence Length and Computing Resource. Most code pre-trained language models (LMs) have an input length limit of 512 subword tokens for base-size models and 1024 for large-size models. Thus, LMs may not fully comprehend long code programs due to the input length limit of LMs, which constrains the repair capability. Furthermore, the increase in output length not only impacts the performance of repair capability but also introduces a well-known challenge associated with long sequences in the NMT model. This difficulty arises from the limitations imposed by Markov chain assumptions and probabilistic constraints, where the model encounters challenges in maintaining coherence and capturing intricate dependencies over extended sequences. In particular, Fu et al. [58] observed that the repair model demonstrated a repair accuracy of 77% when both input and output lengths were below 100 and 10, respectively. However, the model's accuracy drastically dropped to 7% when the input and output lengths exceeded 500 and 50, respectively. In addition, Huang et al. [82] emphasized that the substantial size of language models can place a burden on computing resources, which hinders the generation of candidate patches. For example, during patch synthesis on the Defects4J dataset, the maximum beam size is set at 200, limiting the generation to only 200 patches for each bug.

R8 - Transformer Variants That Process Longer Sequence. Transformer architectures have led to remarkable progress in automated repair applications. However, despite their successes, modern transformers rely on the self-attention mechanism, whose time- and space-complexity is

quadratic in the length of the input that requires substantial computing resources when encountering long sequences. Recently, many alternative architectures have been proposed to mitigate the long sequence challenge and computing burden of transformers. For example, Beltagy et al. [12] presented Longformer which uses global and local attention windows to mitigate the memory and time bottleneck of the self-attention mechanism from O(ns×ns) to O(ns×w), with ns being the sequence length and w being the average window size. In particular, the base-size Longformer can process up to 4,096 subword tokens. On the other hand, Sun et al. [179] proposed Retentive Network (RetNet) as a new foundation architecture for large language models. The chunkwise recurrent representation of RetNet facilitates efficient long-sequence modeling with linear complexity. Thus, future studies may explore their efficacy in the context of program and vulnerability repair.

C9 - Loss of Pre-Trained Knowledge. Most language model-based repair approaches follow a paradigm of taking the pre-trained checkpoint and further fine-tuning the model to fit the downstream program and vulnerability repair tasks. Nevertheless, according to Huang et al. [82], after fine-tuning, pre-trained models may experience a reduction in the knowledge gained during pre-training when compared to zero-shot learning (i.e., without fine-tuning). This could be attributed to conflicting training objectives between the unsupervised masked language modeling (MLM) of pre-training and the supervised neural machine translation (NMT) of fine-tuning.

R9 - Explore Different Training Paradigms. Huang et al. [82] suggested two research directions to address this challenge. Future research could investigate strategies to alleviate catastrophic forgetting [165] for program and vulnerability repair tasks. Additionally, exploring both neural machine translation (NMT) and masked language modeling (MLM) training paradigms during the fine-tuning stage is a potential direction. Notably, AlphaRepair [206] adopts a cloze task (MLM) approach instead of a translation task (NMT), predicting the token at the mask location based on contextual tokens. While employing MLM during fine-tuning could be advantageous as it aligns with the training objective of the model's pre-training stage, the differentiation in repair efficacy between the two paradigms (NMT and MLM) remains unclear.

C10 - Automated Repair on Real-World Scenarios. Pearce et al. [140] found that large language models (LLMs) can perfectly repair all of their synthetic and hand-crafted vulnerability scenarios. However, LLMs were not sufficiently reliable when producing automatic fixes for the real-world data in their qualitative analysis. Furthermore, they underscored the limitation of the current repair approach, which is confined to addressing issues within a single location in a single file.

R10 - Addressing Limitations of LLMs. Addressing the limitations of large language models (LLMs) in real-world vulnerability scenarios presents promising avenues for future research. Pearce et al. [140] identified the significant performance gap of LLMs in repairing synthetic and real-world vulnerability scenarios. Thus, understanding and mitigating the factors contributing to the discrepancy in performance between synthetic and real-world scenarios would be a valuable direction for further investigation, enabling the development of more robust automatic fixes in practical cybersecurity contexts. Furthermore, exploring multi-location and multi-file repair strategies and techniques could offer valuable insights.

5.3 Code Commit

5.3.1 CI/CD Secure Pipelines. AI-driven just-in-time (JIT) software defect prediction (SDP) approaches have been proposed and deployed into CI/CD pipelines for adoption [91, 147]. However, our investigation uncovers several challenges that must be addressed to further enhance these AI-driven approaches. Below, we present the challenge derived from previous JIT SDP studies with potential research opportunities.

C11 - The Use of RNNs. We discovered that many DL-based JIT SDP methods primarily rely on RNNs [36, 143]. However, pre-trained transformer architectures and language models (LMs) demonstrate promise for SDP, building on their successful track record in vulnerability detection [175], a closely related domain. RNNs excel at learning source code representations and predicting defects without relying on predefined metrics. Nonetheless, RNNs process input sequentially and struggle with long sequences, which can lead to suboptimal results compared to transformer-based models.

R11 - Explore LMs and LLMs. In light of the successful application of language models (LMs) for software vulnerability detection [56, 175], future research could explore their potential for JIT SDP. LMs like CodeBERT [51] and CodeT5 [195, 196] are transformer architectures pre-trained on source code, featuring self-attention mechanisms that excel in capturing semantic nuances and handling longer sequences compared to RNNs. Moreover, these LMs are pre-trained on extensive source code datasets encompassing various programming languages, enabling them to generate better source code representations compared to RNNs and enhancing SDP effectiveness. Furthermore, investigating the efficacy of large language models (LLMs) such as GPT-4 [2] and Code Llama [157] for SDP is a valuable research direction. These models are pre-trained extensively to understand source code. Thus, future research could explore strategies for leveraging these LLMs and deploying them to build secure CI/CD pipelines.

5.4 Build, Test, and Deployment

5.4.1 Configuration Validation. AI-driven methodologies have been suggested for automated verification of optimal and secure system configurations. Nevertheless, the complexity of software systems, characterized by an extensive array of configuration options, poses a challenge for machine learning (ML) and deep learning (DL) models. Below, we present the challenge derived from previous studies with potential research opportunities.

C12 - Complex Feature Space. Machine learning (ML) models for extracting insights from numeric and categorical features is a widely adopted strategy in configuration validation processes. Contemporary AI-driven performance prediction models frequently rely on conventional machine learning models like XGBoost [207] or basic feed-forward neural networks such as Deep-Perf [67] to analyze tabular configuration data and predict system performance. Nonetheless, software systems often encompass an extensive array of configuration options. For instance, one of the datasets used to evaluate Deep-Perf consists of 13,485 valid configurations [67]. The extensive configuration size results in intricate relationships among features and an expansive feature space. Consequently, traditional ML models and simple feed-forward neural networks may struggle to generalize to unseen configurations or capture subtle patterns and dependencies within the data, especially in high-dimensional spaces such as configuration datasets.

R12 - Transformers for Tabular Data. Addressing the challenge of capturing intricate relationships and dependencies within high-dimensional configuration datasets presents numerous future research opportunities in performance prediction. One promising avenue for exploration involves leveraging more advanced model architectures, such as transformer-based models [189], to enhance the predictive capabilities of performance prediction models. Transformer architectures, renowned for their success in natural language processing tasks, offer the potential to effectively capture complex patterns and dependencies within tabular data like configuration datasets. By adapting transformer models to handle tabular data, researchers can explore their ability to learn from the sequential and relational information present in configuration parameters and accurately predict system performance. In addition, researchers can explore semi-supervised learning techniques [136]. These methods leverage both labeled and unlabeled data, helping overcome the scarcity of labeled training instances in performance prediction. In summary, these research directions could

improve the effectiveness of AI-driven configuration validation processes in complex software systems.

5.4.2 Infrastructure Scanning. AI-driven infrastructure scanning approaches have been proposed for detecting insecure Infrastructure-as-Code (IaC) scripts during the deployment phase of DevOps. Nevertheless, there remains room for improvement and identified gaps. Below, we outline the challenge derived from previous studies with potential research opportunities.

C13 - Manual Feature Engineering. We found that some AI-driven approaches still rely on manual feature engineering with machine learning (ML) models to predict insecure IaC scripts [35, 151, 152], which can demand substantial effort, especially as the project scales up. It is necessary to prepare a predetermined set of features before model training; for instance, Dalla Palma et al. [35] prepared a set of 108 features. Subsequently, the feature selection process is essential to filter out irrelevant features, and techniques such as normalization will be adopted to optimize ML model performance. This time-consuming process could hinder the practical adoption of AI-driven infrastructure scanning approaches.

R13 - Explore DL-based Techniques. Considering the recent advancement of deep learning (DL) and language models (LMs), it is feasible to use DL models for detecting defective IaC scripts. Unlike traditional machine learning (ML) models, DL models allow direct input of IaC scripts, thereby eliminating the need for manual feature engineering. DL models are capable of learning both semantic and syntactic features of IaC scripts. They can identify insecure patterns within IaC scripts based on historical data and predict defective scripts in future instances. Additionally, LMs have proven successful in various software security tasks, such as vulnerability detection [56, 175] and repairs [13, 58, 73]. Given that LMs are pre-trained on vast amounts of source code data, it is viable to fine-tune them for accurately detecting insecure IaC scripts. Hence, future research could explore the application of DL models to address the time-consuming challenges associated with manual feature engineering in detecting insecure IaC scripts.

5.5 Operation and Monitoring

5.5.1 Log Analysis and Anomaly Detection. Several AI-driven anomaly detection approaches have been developed to detect anomalies in software systems. However, our investigation reveals the challenge that requires attention and resolution. Below, we introduce the challenge derived from previous studies with potential research opportunities.

C14 - Normality Drift for Zero-Positive Anomaly Detection. In anomaly detection, zero-positive classification is commonly used because anomalies are typically rare and undefined. Zero-positive classification trains models on normal behavior, allowing them to generalize to unseen anomalies and adapt to changing data distributions. However, Han et al. [70] recognized a normality drift problem for the zero-positive classification used in anomaly detection. Normality drift refers to the phenomenon where the underlying distribution of normal (non-anomalous) data changes over time in a dataset used for training AI-driven anomaly detections. In other words, the characteristics of normal behavior exhibited by the system or environment being monitored may evolve or shift gradually or abruptly, leading to discrepancies between the training data and the data encountered during deployment or inference. This drift can adversely affect the performance of anomaly detection systems, as models trained on historical data may become less effective at identifying anomalies from the new normal behavior. Thus, addressing normality drift is crucial for maintaining the effectiveness and reliability of anomaly detection systems.

R14 - Enhance Normality Drift Detection. Han et al. [70] introduced OWAD as a solution to combat the challenge of normality drift encountered in deep learning-based anomaly detection for security applications. OWAD serves as a framework designed to detect, explain, and adapt

to normality shifts at the distribution level, departing from sample-level explanations like CADE [216] which fail to comprehensively address the holistic shift in distribution, thereby limiting their applicability in understanding the overall normality drift. This development paves the way for numerous future research opportunities in anomaly detection. Particularly, there exists potential for exploring innovative adaptation strategies within OWAD aimed at enhancing its efficacy in adapting to evolving data distributions. This could entail investigating reinforcement learning approaches or meta-learning techniques to dynamically adjust model parameters in response to shifting distributions. Future investigations can also concentrate on refining OWAD's adaptation mechanisms to ensure effectiveness across diverse security environments.

5.5.2 Cyber-Physical Systems. AI-driven approaches have been proposed to address security concerns in cyber-physical systems (CPS) [11, 110, 162, 205, 209–211]. However, our investigation reveals that current approaches to CPS security primarily target individual cyber-attacks, neglecting the complex scenario of encountering multiple simultaneous threats across diverse CPS layers. Below, we describe this challenge and highlight potential avenues for future research.

C15 - Monitoring Multiple Cyber-Attacks Simultaneously. CPS consists of multiple layers representing distinct components and functionalities crucial for system operation. These layers typically include the physical layer encompassing hardware infrastructure, the communication layer facilitating data exchange between components, the control layer governing system behavior, and the data processing layer analyzing collected data. However, current approaches in CPS security often focus on addressing individual cyber-attacks, overlooking the reality of facing multiple simultaneous threats across various CPS layers [11, 110, 162, 205, 209–211]. For instance, attackers may simultaneously disrupt communication networks, manipulate control algorithms, and tamper with data analytics results in the smart grid CPS, posing complex challenges for system security and resilience. As cyber threats grow in sophistication and diversity, the need to coordinate responses and monitor ongoing attacks concurrently becomes paramount. However, achieving this in real time presents substantial challenges due to the intricate and interconnected nature of CPS environments.

R15 - Distributed Anomaly Detection and Multi-Agent Systems. Future research could focus on developing distributed detection and response mechanisms within CPS environments, empowering individual components to autonomously detect and respond to cyber threats in real time. This approach decentralizes the security architecture and reduces reliance on centralized control systems. By integrating AI-based anomaly detections, these mechanisms could effectively identify unusual patterns or behaviors indicative of cyber attacks across various CPS layers. Multi-agent systems (MAS) hold promise for addressing CPS attacks across different layers due to their decentralized and collaborative nature. MAS consists of multiple autonomous agents, each capable of independent decision-making and action. For example, Lyu and Brennan [116] introduced a two-layer architecture modeling framework for CPS, demonstrating how it enables real-time adaptation in dynamic industrial automation environments. Moreover, integrating AI techniques with MAS could facilitate collaborative decision-making and resource allocation among autonomous agents distributed across different CPS components or layers. By leveraging reinforcement learning (RL), agents can dynamically adapt their strategies based on feedback from the environment, allowing them to respond to emerging cyber threats with agility and precision. For instance, Ibrahim and Elhafiz [85] presented a case study of a smart grid and investigated CPS security using RL. In summary, AI-driven MAS could adaptively adjust defense strategies based on evolving threat landscapes and system conditions, enhancing the effectiveness of cyber defense operations within CPS environments.

6 SUPPLEMENTARY DISCUSSION: SECURITY TASKS IN DEVSECOPS

We have addressed RQ1 and RQ2 in Section 4 and Section 5, respectively. However, from our selected publication venues, we found no associated literature discussing AI-driven security approaches for threat modeling and impact analysis in the planning step in DevOps. Similarly, we found no literature addressing dependency management in the code commit step in DevOps. Thus, in this section, we discuss common approaches for threat modeling, impact analysis, and dependency management, and their relevance to the DevSecOps principle. We also introduce potential AI applications that can be used to facilitate these processes and enhance security.

6.1 Plan

6.1.1 Threat Modeling. Threat modeling is a systematic approach used in software development and cybersecurity to identify and mitigate potential security threats and ensure secure DevOps planning [43]. This process involves analyzing the system's architecture, components, data flow, and potential attack vectors to understand where security weaknesses may exist. One widely adopted framework for threat modeling is STRIDE proposed by Howard and Lipner [79], which categorizes threats into six main types: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of Privilege.

On the other hand, the DREAD framework proposed by [98] provides a structured approach to assess the severity of identified threats. DREAD stands for Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability. This framework allows teams to assign scores to each criterion, typically on a scale from 0 to 10, to evaluate the potential impact of a threat. By considering factors such as the potential damage caused, the ease of exploitation, and the number of users affected, teams can prioritize their security efforts and focus on addressing the most critical vulnerabilities first.

Both STRIDE and DREAD threat modeling is typically conducted by human security experts, often in collaboration with developers, architects, and other stakeholders involved in the software development process. These collaborative processes ensure that security considerations are integrated into DevOps planning throughout the development lifecycle and that potential threats are identified, assessed, and mitigated at an early stage of development.

6.1.2 Impact Analysis. Software impact analysis is a crucial process that analyzes, predicts, and estimates the potential consequences before a change in the deployed product [93]. Impact analysis integrates security considerations into the planning phase of DevOps by analyzing potential unexpected side effects of decisions or changes within a system and identifying potentially affected areas. This process starts by identifying impacted modules and functionality, describing proposed changes, and delineating affected areas. Risk assessment is used to evaluate potential risks associated with each change, such as performance changes, security vulnerabilities, and compatibility issues, often using a qualitative scale or numerical scoring system [184].

In particular, Turver and Munro [185] introduced a technique for the early detection of ripple effects based on a simple graph-theoretic model of documentation and the themes within the documentation. This technique aims to provide a more accessible and effective approach to assessing the impact of changes, particularly during the early stages of a project when source code understanding may be limited. Gethers et al. [59] introduced an adaptive approach for conducting impact analysis from a change request to the source code. The approach begins with a textual change request, such as a bug report. It uses a single snapshot (release) of the source code which is indexed using Latent Semantic Indexing to estimate the impact set. In particular, the analysis encompasses information retrieval, dynamic analysis, and data mining of previous source code commits. They evaluated

their approaches using open-source software systems and showed significant improvement in their combined approach over standalone approaches.

Potential AI Application. The recent advancement of AI models presents promising opportunities for facilitating threat modeling and impact analysis in cybersecurity. For instance, an AI-driven commercial tool named Aribot for threat modeling has been introduced by Aristiun [7]. Aribot automates the threat modeling process through various functionalities. It automatically generates Infrastructure-as-Code templates, addressing public cloud-specific threats effectively. It ensures traceable security requirements throughout the lifecycle, facilitating comprehensive security coverage. Aribot also simplifies compliance adherence by mapping security requirements to frameworks such as the National Institute of Standards and Technology (NIST). Additionally, it enhances transparency and accountability by reporting and tracking records remediation efforts by development teams, providing real-time updates on implementation status without requiring manual intervention.

Another AI-driven commercial tool for impact analysis has been introduced by Validata [188]. This AI-based solution automatically delivers crucial information for applications, expediting upgrades and patches while predicting the impact of new product versions before release. It empowers users to effortlessly analyze change impact on quality, performance, resource capacity, and costs within hours, automatically applying recommended changes through a corrective action plan.

Moreover, Microsoft [128] has recently introduced Copilot for Security. Informed by large-scale data and threat intelligence, including more than 78 trillion security signals processed by Microsoft each day. Copilot is coupled with large language models (LLMs) to deliver tailored security insights. It offers an interactive interface to support security practitioners during impact analysis and threat modeling with relevant security knowledge. Their randomized controlled trial indicated that experienced security analysts were 22% faster with Copilot, they were 7% more accurate across all tasks when using Copilot, and 97% said they want to use Copilot the next time they do the same task [47]. Copilot serves as the first critical step in leveraging generative AI to support security practitioners in their workflow.

In summary, the adoption of AI in threat modeling and impact analysis presents significant advantages for the planning step in DevSecOps. Tools like Aribot, Validata, and Copilot for Security showcase the potential of AI to automate these processes effectively. By leveraging the capabilities of AI to assess risks, identify vulnerabilities, and prioritize security measures, organizations could make informed decisions and allocate resources more efficiently.

6.2 Code Commit

6.2.1 Dependency Management. The issue of vulnerable dependencies is widely recognized in software ecosystems due to the extensive interconnection of free and open-source software (FOSS) libraries [139]. Thus, dependency management plays a crucial role in assisting developers to handle and organize the various external components or libraries that a software project relies on to function correctly. This process involves identifying, tracking, and managing the dependencies between different modules, libraries, or packages within a software application. Effective dependency management ensures secure and resilient software development practices in DevSecOps, mitigating the risks associated with vulnerable dependencies.

Dependency management includes package managers like npm (Node.js), pip (Python), Maven (Java), and NuGet (.NET), which automate the installation, configuration, and versioning of dependencies. In addition, there exist commercial tools that provide visibility into dependencies and help identify and mitigate security vulnerabilities. For instance, Sonatype Nexus [172] is a repository

manager used for managing software components. This tool allows organizations to store and retrieve artifacts securely and efficiently. Nexus supports various package formats and integrates with popular build tools and CI/CD pipelines, providing dependency management, security scanning, and artifact promotion capabilities. Black Duck [181] is a software composition analysis (SCA) platform designed to manage risks linked to open-source and third-party software components. Black Duck scans codebases, detects open-source components, and assesses their compliance with licenses, security vulnerabilities, and overall quality. It integrates with development tools for proactive dependency management. Snyk [171] also offers a dependency management tool, which is a developer-centric approach to application security, seamlessly integrating into existing DevOps workflows. This tool offers a Dependency Tree View for identifying dependencies and their vulnerabilities, automatically updating them as they evolve. With features like automated scanning within IDEs, comprehensive vulnerability databases, and prioritized remediation, Snyk empowers developers to proactively manage their risk exposure and maintain the integrity of their software projects.

Potential AI Security Applications. Recently, a startup company Infield introduced an AI-driven commercial tool designed to automate software dependency management and strengthen DevOps security [69]. The tool continuously monitors recommended updates of open-source components, provides step-by-step guidance for achieving the ideal status, and gathers unstructured information about open-source dependencies and their upgrades. This data is then structured to help users manage their backlog of upgrades efficiently, allowing them to prioritize upgrades based on risk and effort. For instance, the tool can be connected with users' codebase in GitHub, scans their code to determine the underlying dependencies, and recommends the steps needed to upgrade safely for their codebase. The tool offers a human-assisted approach to dependency management, helping organizations overcome the challenges of maintaining dependencies in a rapidly evolving ecosystem.

7 THREATS TO VALIDITY

Our systematic literature review (SLR) was conducted in alignment with the guidelines outlined by Keele et al. [90], Kitchenham et al. [94]. However, like any SLR, our review also has certain limitations. Below, we provide a discussion of the external, internal, and construct threats to the validity of our SLR, along with corresponding mitigation strategies.

Threats to external validity relate to the search string, the filtering process, and the selection of venues in this SLR aimed at identifying literature related to AI-driven methodologies and tools for DevSecOps. It is possible that our search string missed studies that should have been included in our review, potentially due to a missed term or a combination of terms that may have returned more significant results. Given that our study focuses on two areas—AI (specifically, machine learning and deep learning) and security methodologies and tools for integration into DevOps—we employ a systematic approach to testing different combinations of AI-related terms and security tasks as presented in Section 3.2. This involves experimenting with variations in search strings, including synonyms, related terms, and alternative phrasings. By doing so, we aim to increase the likelihood of identifying relevant studies that may have been overlooked initially. Furthermore, we leverage our understanding of the domain to refine and optimize our search strategy iteratively to mitigate this threat.

While conducting this systematic literature review (SLR), we acknowledge the potential for selection bias in the studies included. To mitigate this threat, we employ rigorous inclusion and

exclusion criteria predefined before initiating the filtering process as illustrated in Table 3. Furthermore, we implement a snowballing search strategy to supplement our initial searches as presented in Section 3.4, thereby capturing papers that may have been overlooked initially.

The selection of venues for this SLR plays a crucial role in ensuring the reliability of our findings. In our approach, we deliberately include top-tier SE and security conferences and journals, focusing on venues with CORE A or CORE A* rankings. By prioritizing these esteemed venues, known for their rigorous peer-review processes and high academic standards, we aim to uphold the quality and credibility of the literature surveyed. While we acknowledge the possibility that our exclusion of lower-ranking venues may have resulted in the omission of relevant studies, our deliberate focus on top SE and security venues allows us to capture emerging trends and insights from reputable sources. To provide transparency and accountability in our venue selection process, we disclose the selected venues in Table 2, enabling readers to assess our review methodology.

Threats to internal validity relate to the potential absence of literature discussing AI-driven methods or tools for specific security tasks within the DevSecOps process. Notably, despite our comprehensive search efforts, we encountered a scarcity of studies addressing AI applications for threat modeling and impact analysis in the plan step, as well as dependency management in the code commit step. This gap in the literature poses a potential limitation to the comprehensiveness of our review. We have devised a mitigation strategy to address this limitation. In addition to our primary research questions (RQ1 and RQ2), we include a supplementary Section 6 following our RQ discussion to examine existing approaches for threat modeling, impact analysis, and dependency management. By scrutinizing relevant literature outside the scope of AI-driven methods, we aim to offer a comprehensive overview of current practices and their relevance to DevSecOps.

8 CONCLUSION

AI-driven security approaches are revolutionizing the automation of software security, presenting opportunities to seamlessly integrate security practices into the DevOps workflow and realize the DevSecOps paradigm efficiently. Throughout this systematic literature review, we collected papers from high-impact software engineering and software security venues, analyzing 99 papers focusing on AI-driven security approaches tailored for DevSecOps. We unveiled 12 security tasks critical to DevOps and examined various AI-driven security approaches and tools (RQ1). Subsequently, we identified 19 significant challenges confronting state-of-the-art AI-driven security methodologies and derived promising avenues for future research (RQ2). In conclusion, this paper sheds light on the transformative potential of AI-driven security techniques in realizing the DevSecOps paradigm, while identifying critical challenges and emphasizing the need for future endeavors to address them in this critical intersection of AI and DevSecOps.

9 ACKNOWLEDGMENTS

C. Tantithamthavorn was partially supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100941).

REFERENCES

[1] Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications* 7, 1 (1994), 39–59.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Diana Laura Aguilar, Miguel Angel Medina-Perez, Octavio Loyola-Gonzalez, Kim-Kwang Raymond Choo, and Edoardo Bucheli-Susarrey. 2022. Towards an interpretable autoencoder: A decision-tree-based autoencoder and its application in anomaly detection. *IEEE transactions on dependable and secure computing* 20, 2 (2022), 1048–1059.

[4] Muhammad Azeem Akbar, Kari Smolander, Sajjad Mahmood, and Ahmed Alsanad. 2022. Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology* 147 (2022), 106894.
[5] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. 2018. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. In *6th International Conference on Learning Representations (ICLR)*. Arxiv-Computer Science.
[6] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. 2020. Automation of vulnerability classification from its description using machine learning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–7.
[7] Aristiun. 2024. Automated Threat Modeling using AI. https://www.aristiun.com/automated-threat-modeling-using-ai.
[8] Robert S Arnold and Shawn A Bohner. 1993. Impact analysis-towards a framework for comparison. In *1993 conference on software maintenance*. IEEE, 292–301.
[9] Ahmed Bahaa, Ahmed Abdelaziz, Abdalla Sayed, Laila Elfangary, and Hanan Fahmy. 2021. Monitoring real time security attacks for IoT systems using DevSecOps: a systematic literature review. *Information* 12, 4 (2021), 154.
[10] Liang Bao, Xin Liu, Fangzheng Wang, and Baoyin Fang. 2019. ACTGAN: automatic configuration tuning for software systems with generative adversarial networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 465–476.
[11] Yuequan Bao, Zhiyi Tang, Hui Li, and Yufeng Zhang. 2019. Computer vision and deep learning–based data anomaly detection method for structural health monitoring. *Structural Health Monitoring* 18, 2 (2019), 401–421.
[12] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).
[13] Berkay Berabi, Jingxuan He, Veselin Raychev, and Martin Vechev. 2021. Tfix: Learning to fix coding errors with a text-to-text transformer. In *International Conference on Machine Learning*. PMLR, 780–791.
[14] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. 2021. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)* 54, 3 (2021), 1–33.
[15] Nemania Borovits, Indika Kumara, Dario Di Nucci, Parvathy Krishnan, Stefano Dalla Palma, Fabio Palomba, Damian A Tamburri, and Willem-Jan van den Heuvel. 2022. FindICI: Using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code. *Empirical Software Engineering* 27, 7 (2022), 178.
[16] George G Cabral, Leandro L Minku, Emad Shihab, and Suhaib Mujahid. 2019. Class imbalance evolution and verification latency in just-in-time software defect prediction. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 666–676.
[17] Wenjing Cai, Junlin Chen, Jiaping Yu, and Lipeng Gao. 2023. A software vulnerability detection method based on deep learning with complex network analysis and subgraph partition. *Information and Software Technology* 164 (2023), 107328.
[18] Jeanderson Cândido, Maurício Aniche, and Arie van Deursen. 2021. Log-based software monitoring: a systematic mapping study. *PeerJ Computer Science* 7 (2021), e489.
[19] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei, and Bin Li. 2021. Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection. *Information and Software Technology* 136 (2021), 106576.
[20] Antonio Carzaniga, Alfonso Fuggetta, Richard S Hall, Dennis Heimbigner, André Van Der Hoek, and Alexander L Wolf. 1998. *A characterization framework for software deployment technologies*. Technical Report. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado.
[21] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2021. Deep learning based vulnerability detection: Are we there yet. *IEEE Transactions on Software Engineering* (2021).
[22] Checkmarx. 2006. Checkmarx. https://checkmarx.com/.
[23] Xiang Chen, Yingquan Zhao, Qiuping Wang, and Zhidan Yuan. 2018. MULTI: Multi-objective effort-aware just-in-time software defect prediction. *Information and Software Technology* 93 (2018), 1–13.
[24] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2022. Neural transfer learning for repairing security vulnerabilities in c code. *IEEE Transactions on Software Engineering* 49, 1 (2022), 147–165.
[25] Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2019. Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering* 47, 9 (2019), 1943–1959.
[26] Guoli Cheng, Shi Ying, and Bingming Wang. 2021. Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model. *Journal of Systems and Software* 180 (2021), 111028.

[27] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. Deepwukong: Statically detecting software vulnerabilities using deep graph neural network. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–33.
[28] Jianlei Chi, Yu Qu, Ting Liu, Qinghua Zheng, and Heng Yin. 2022. Seqtrans: automatic vulnerability fix via sequence to sequence learning. *IEEE Transactions on Software Engineering* 49, 2 (2022), 564–585.
[29] E Chickowski. 2018. Seven Winning DevSecOps Metrics Security Should Track. *Bitdefender) Retrieved March* 25 (2018), 2019.
[30] Javier Alvarez Cid-Fuentes, Claudia Szabo, and Katrina Falkner. 2018. Adaptive performance anomaly detection in distributed systems using online svms. *IEEE Transactions on Dependable and Secure Computing* 17, 5 (2018), 928–941.
[31] CORE. 2023. International CORE Conference Rankings: ICORE. https://www.core.edu.au/icore-portal.
[32] Roland Croft, M Ali Babar, and M Mehdi Kholoosi. 2023. Data quality for software vulnerability datasets. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 121–133.
[33] Roland Croft, Dominic Newlands, Ziyu Chen, and M Ali Babar. 2021. An empirical study of rule-based and learning-based approaches for static application security testing. In *Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*. 1–12.
[34] CWE. 2023. 2023 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html.
[35] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian A Tamburri. 2021. Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering* 48, 6 (2021), 2086–2104.
[36] Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul-Joo Kim. 2019. Lessons learned from using a deep tree-based model for software defect prediction in practice. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*. IEEE, 46–57.
[37] Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. 2018. Automatic feature learning for predicting vulnerable software components. *IEEE Transactions on Software Engineering* 47, 1 (2018), 67–85.
[38] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Pothen, and Ehab Al-Shaer. 2021. V2w-bert: A framework for effective hierarchical multiclass classification of software vulnerabilities. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–12.
[39] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. 2020. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *International Conference on Learning Representations (ICLR)*.
[40] Yangruibo Ding, Sahil Suneja, Yunhui Zheng, Jim Laredo, Alessandro Morari, Gail Kaiser, and Baishakhi Ray. 2022. VELVET: a noVel Ensemble Learning approach to automatically locate VulnErable sTatements. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 959–970.
[41] Yukun Dong, Yeer Tang, Xiaotong Cheng, and Yufei Yang. 2023. DeKeDVer: A deep learning-based multi-type software vulnerability classification framework using vulnerability description and source code. *Information and Software Technology* (2023), 107290.
[42] Yukun Dong, Yeer Tang, Xiaotong Cheng, Yufei Yang, and Shuqi Wang. 2023. SedSVD: Statement-level software vulnerability detection based on Relational Graph Convolutional Network with subgraph embedding. *Information and Software Technology* 158 (2023), 107168.
[43] Victoria Drake. 2024. Threat Modeling. https://owasp.org/www-community/Threat_Modeling.
[44] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1283–1297.
[45] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.
[46] Xiaoning Du, Bihuan Chen, Yuekang Li, Jianmin Guo, Yaqin Zhou, Yang Liu, and Yu Jiang. 2019. Leopard: Identifying vulnerable code for vulnerability assessment through program metrics. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 60–71.
[47] Benjamin G Edelman, James Bono, Sida Peng, Roberto Rodriguez, and Sandra Ho. 2023. Randomized Controlled Trial for Microsoft Security Copilot. *Available at SSRN 4648700* (2023).
[48] Abdulmotaleb El Saddik. 2018. Digital twins: The convergence of multimedia technologies. *IEEE multimedia* 25, 2 (2018), 87–92.
[49] Floris MA Erich, Chintan Amrit, and Maya Daneva. 2017. A qualitative study of DevOps usage in practice. *Journal of software: Evolution and Process* 29, 6 (2017), e1885.
[50] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. AC/C++ code vulnerability dataset with code changes and CVE summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 508–512.

[51] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 1536–1547.

[52] Jerome H Friedman and Bogdan E Popescu. 2008. Predictive learning via rule ensembles. (2008).

[53] Michael Fu, Van Nguyen, Chakkrit Tantithamthavorn, Dinh Phung, and Trung Le. 2023. Vision Transformer-Inspired Automated Vulnerability Repair. *ACM Transactions on Software Engineering and Methodology* (2023).

[54] Michael Fu, Van Nguyen, Chakkrit Kla Tantithamthavorn, Trung Le, and Dinh Phung. 2023. VulExplainer: A Transformer-based Hierarchical Distillation for Explaining Vulnerability Types. *IEEE Transactions on Software Engineering* (2023).

[55] Michael Fu and Chakkrit Tantithamthavorn. 2022. GPT2SP: A transformer-based agile story point estimation approach. *IEEE Transactions on Software Engineering* 49, 2 (2022), 611–625.

[56] Michael Fu and Chakkrit Tantithamthavorn. 2022. Linevul: A transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 608–620.

[57] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Yuki Kume, Van Nguyen, Dinh Phung, and John Grundy. 2024. AIBugHunter: A Practical tool for predicting, classifying and repairing software vulnerabilities. *Empirical Software Engineering* 29, 1 (2024), 4.

[58] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. 2022. VulRepair: a T5-based automated software vulnerability repair. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 935–947.

[59] Malcom Gethers, Bogdan Dit, Huzefa Kagdi, and Denys Poshyvanyk. 2012. Integrated impact analysis for managing software changes. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 430–440.

[60] GitHub. 2024. About autofix for CodeQL code scanning. https://docs.github.com/en/code-security/code-scanning/managing-code-scanning-alerts/about-autofix-for-codeql-code-scanning#supported-languages.

[61] GitHub. 2024. Dependabot. https://github.com/dependabot.

[62] Google. 2023. 2023 State of DevOps Report. https://cloud.google.com/devops/state-of-devops.

[63] Alicja Gosiewska and Przemyslaw Biecek. 2019. IBreakDown: Uncertainty of model explanations for non-additive predictive models. *arXiv preprint arXiv:1903.11420* (2019).

[64] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[65] GuardRails. 2023. From Security Last to DevSecOps: The Driving Forces Behind the Transformation. https://www.guardrails.io/blog/from-security-last-to-devsecops-the-driving-forces-behind-the-transformation/.

[66] Michele Guerriero, Martin Garriga, Damian A Tamburri, and Fabio Palomba. 2019. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 580–589.

[67] Huong Ha and Hongyu Zhang. 2019. DeepPerf: Performance prediction for configurable software with deep sparse neural network. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1095–1106.

[68] Tanja Hagemann and Katerina Katsarou. 2020. A systematic review on anomaly detection for cloud computing environments. In *Proceedings of the 2020 3rd Artificial Intelligence and Cloud Computing Conference*. 83–96.

[69] Susan Hall. 2024. AI-Assisted Dependency Updates without Breaking Things. https://thenewstack.io/ai-assisted-dependency-updates-without-breaking-things/.

[70] Dongqi Han, Zhiliang Wang, Wenqi Chen, Kai Wang, Rui Yu, Su Wang, Han Zhang, Zhihua Wang, Minghui Jin, Jiahai Yang, et al. 2023. Anomaly Detection in the Open World: Normality Shift Detection, Explanation, and Adaptation. In *30th Annual Network and Distributed System Security Symposium (NDSS)*.

[71] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. 2021. Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3197–3217.

[72] Shangbin Han, Qianhong Wu, Han Zhang, Bo Qin, Jiankun Hu, Xingang Shi, Linfeng Liu, and Xia Yin. 2021. Log-based anomaly detection with robust feature extraction and online learning. *IEEE Transactions on Information Forensics and Security* 16 (2021), 2300–2311.

[73] Sichong Hao, Xianjun Shi, Hongwei Liu, and Yanjun Shu. 2023. Enhancing Code Language Models for Program Repair by Curricular Fine-tuning Framework. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 136–146.

[74] Mubin Ul Haque, M Mehdi Kholoosi, and M Ali Babar. 2022. KGSecConfig: A Knowledge Graph Based Approach for Secured Container Orchestrator Configuration. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 420–431.

[75] A.W. Harzing. 2007. Publish or Perish. Available online at https://harzing.com/resources/publish-or-perish.

[76] David Hin, Andrey Kan, Huaming Chen, and M Ali Babar. 2022. LineVD: Statement-level vulnerability detection using graph neural networks. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 596–607.
[77] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems* 33 (2020), 6840–6851.
[78] White House. 2023. Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence. https://www.whitehouse.gov/briefing-room/presidential-actions/2023/10/30/executive-order-on-the-safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence/.
[79] Michael Howard and Steve Lipner. 2006. *The security development lifecycle*. Vol. 8. Microsoft Press Redmond.
[80] Michael A Howard. 2006. A process for performing security code reviews. *IEEE Security & privacy* 4, 4 (2006), 74–79.
[81] Jun Huang, Yang Yang, Hang Yu, Jianguo Li, and Xiao Zheng. 2023. Twin Graph-Based Anomaly Detection via Attentive Multi-Modal Learning for Microservice System. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 66–78.
[82] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. 2023. An Empirical Study on Fine-Tuning Large Language Models of Code for Automated Program Repair. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1162–1174.
[83] Peng Huang, William J Bolosky, Abhishek Singh, and Yuanyuan Zhou. 2015. Confvalley: A systematic configuration validation framework for cloud services. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–16.
[84] Jez Humble and David Farley. 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.
[85] Mariam Ibrahim and Ruba Elhafiz. 2023. Security analysis of cyber-physical systems using reinforcement learning. *Sensors* 23, 3 (2023), 1634.
[86] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. Cure: Code-aware neural machine translation for automatic program repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1161–1173.
[87] Jirayus Jiarpakdee, Chakkrit Kla Tantithamthavorn, Hoa Khanh Dam, and John Grundy. 2020. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering* 48, 1 (2020), 166–185.
[88] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. Inferfix: End-to-end program repair with llms. *arXiv preprint arXiv:2303.07263* (2023).
[89] Per Jönsson and Mikael Lindvall. 2005. *Impact Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 117–142. https://doi.org/10.1007/3-540-28244-0_6
[90] Staffs Keele et al. 2007. Guidelines for performing systematic literature reviews in software engineering.
[91] Chaiyakarn Khanan, Worawit Luewichana, Krissakorn Pruktharathikoon, Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, Morakot Choetkiertikul, Chaiyong Ragkhitwetsagul, and Thanwadee Sunetnanta. 2020. JITBot: an explainable just-in-time defect prediction bot. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. 1336–1339.
[92] Wael Khreich, Babak Khosravifar, Abdelwahab Hamou-Lhadj, and Chamseddine Talhi. 2017. An anomaly detection system based on variable N-gram features and one-class SVM. *Information and Software Technology* 91 (2017), 186–197.
[93] Anna Khrupa. 2022. What Is Software Impact Analysis. https://qarea.com/blog/what-is-software-impact-analysis.
[94] Barbara Kitchenham, Lech Madeyski, and David Budgen. 2022. SEGRESS: Software engineering guidelines for reporting secondary studies. *IEEE Transactions on Software Engineering* 49, 3 (2022), 1273–1298.
[95] Kun Kuang, Lian Li, Zhi Geng, Lei Xu, Kun Zhang, Beishui Liao, Huaxin Huang, Peng Ding, Wang Miao, and Zhichao Jiang. 2020. Causal inference. *Engineering* 6, 3 (2020), 253–263.
[96] Jinpeng Lan, Lina Gong, Jingxuan Zhang, and Haoxiang Zhang. 2023. BTLink: automatic link recovery between issues and commits based on pre-trained BERT model. *Empirical Software Engineering* 28, 4 (2023), 103.
[97] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 492–504.
[98] David LeBlanc and Michael Howard. 2002. *Writing secure code*. Pearson Education.
[99] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Maat: Performance Metric Anomaly Anticipation for Cloud Services with Conditional Diffusion. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 116–128.
[100] Tiina Leppänen, Anne Honkaranta, and Andrei Costin. 2022. Trends for the DevOps security. A systematic literature review. In *International Symposium on Business Modeling and Software Design*. Springer, 200–217.
[101] Jingyue Li, Reidar Conradi, Christian Bunse, Marco Torchiano, Odd Petter N Slyngstad, and Maurizio Morisio. 2009. Development with off-the-shelf components: 10 facts. *IEEE software* 26, 2 (2009), 80–87.

[102] Jiangming Li, Huasen He, Shuangwu Chen, and Dong Jin. 2023. LogGraph: Log Event Graph Learning Aided Robust Fine-Grained Anomaly Diagnosis. *IEEE Transactions on Dependable and Secure Computing* (2023).
[103] Weiwei Li, Wenzhou Zhang, Xiuyi Jia, and Zhiqiu Huang. 2020. Effort-aware semi-supervised just-in-time defect prediction. *Information and Software Technology* 126 (2020), 106364.
[104] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2022. SwissLog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing* (2022).
[105] Xiangwei Li, Xiaoning Ren, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. 2023. Prediction of Vulnerability Characteristics Based on Vulnerability Description and Prompt Learning. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 604–615.
[106] Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. Dlfix: Context-based code transformation learning for automated program repair. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 602–614.
[107] Yi Li, Shaohua Wang, and Tien N Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 292–303.
[108] Zhen Li, Deqing Zou, Shouhuai Xu, Zhaoxuan Chen, Yawei Zhu, and Hai Jin. 2021. Vuldeelocator: a deep learning-based fine-grained vulnerability detector. *IEEE Transactions on Dependable and Secure Computing* 19, 4 (2021), 2821–2837.
[109] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv preprint arXiv:1801.01681* (2018).
[110] Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. 2018. TABOR: A graphical model-based approach for anomaly detection in industrial control systems. In *Proceedings of the 2018 on asia conference on computer and communications security*. 525–536.
[111] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.
[112] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 1 (2012), 1–39.
[113] Shigang Liu, Guanjun Lin, Lizhen Qu, Jun Zhang, Olivier De Vel, Paul Montague, and Yang Xiang. 2020. CD-VulD: Cross-domain vulnerability discovery based on deep domain adaptation. *IEEE Transactions on Dependable and Secure Computing* 19, 1 (2020), 438–451.
[114] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*. 4768–4777.
[115] Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. Coconut: combining context-aware neural translation models using ensemble for program repair. In *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*. 101–114.
[116] Guolin Lyu and Robert W Brennan. 2023. Multi-agent modelling of cyber-physical systems for IEC 61499-based distributed intelligent automation. *International Journal of Computer Integrated Manufacturing* (2023), 1–27.
[117] Andi Mann, Michael Stahnke, Alanna Brown, and Nigel Kersten. 2019. State of DevOps Report, 2019.
[118] Runfeng Mao, He Zhang, Qiming Dai, Huang Huang, Guoping Rong, Haifeng Shen, Lianping Chen, and Kaixiang Lu. 2020. Preliminary findings about devsecops from grey literature. In *2020 IEEE 20th international conference on software quality, reliability and security (QRS)*. IEEE, 450–457.
[119] Diego Marcilio, Carlo A Furia, Rodrigo Bonifácio, and Gustavo Pinto. 2020. SpongeBugs: Automatically generating fix suggestions in response to static code analysis warnings. *Journal of Systems and Software* 168 (2020), 110671.
[120] Ehsan Mashhadi and Hadi Hemmati. 2021. Applying codebert for automated program repair of java simple bugs. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 505–509.
[121] Sonu Mehta, Ranjita Bhagwan, Rahul Kumar, Chetan Bansal, Chandra Maddila, Balasubramanyan Ashok, Sumit Asthana, Christian Bird, and Aditya Kumar. 2020. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 435–448.
[122] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.
[123] Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. 2020. Long-tail learning via logit adjustment. *arXiv preprint arXiv:2007.07314* (2020).
[124] Ibrahim Mesecan, Daniel Blackwell, David Clark, Myra B Cohen, and Justyna Petke. 2021. HyperGI: Automated detection and repair of information flow leakage. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1358–1362.

[125] Microsoft. 2022. DevSecOps controls. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/secure/devsecops-controls.
[126] Microsoft. 2022. What is infrastructure as code (IaC)? https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code.
[127] Microsoft. 2023. What is DevOps? https://learn.microsoft.com/en-us/devops/what-is-devops.
[128] Microsoft. 2024. Microsoft Copilot for Security is generally available on April 1, 2024, with new capabilities. https://www.microsoft.com/en-us/security/blog/2024/03/13/microsoft-copilot-for-security-is-generally-available-on-april-1-2024-with-new-capabilities/.
[129] Microsoft. 2024. Threat Modeling. https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling.
[130] Yisroel Mirsky, George Macon, Michael Brown, Carter Yagemann, Matthew Pruett, Evan Downing, Sukarno Mertoguno, and Wenke Lee. 2023. VulChecker: Graph-based Vulnerability Localization in Source Code. In *31st USENIX Security Symposium, Security 2022*.
[131] Håvard Myrbakken and Ricardo Colomo-Palacios. 2017. DevSecOps: a multivocal literature review. In *Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings*. Springer, 17–29.
[132] Rennie Naidoo and Nicolaas Möller. 2022. Building Software Applications Securely With DevSecOps: A Socio-Technical Perspective. In *ECCWS 2022 21st European Conference on Cyber Warfare and Security*. Academic Conferences and publishing limited.
[133] Marjane Namavar, Noor Nashid, and Ali Mesbah. 2022. A controlled experiment of different code representations for learning-based program repair. *Empirical Software Engineering* 27, 7 (2022), 190.
[134] Chao Ni, Kaiwen Yang, Yan Zhu, Xiang Chen, and Xiaohu Yang. 2023. Unifying Defect Prediction, Categorization, and Repair by Multi-Task Deep Learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1980–1992.
[135] NSF. 2024. Cyber-physical systems. https://www.nsf.gov/news/special_reports/cyber-physical/.
[136] Yassine Ouali, Céline Hudelot, and Myriam Tami. 2020. An overview of deep semi-supervised learning. *arXiv preprint arXiv:2006.05278* (2020).
[137] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. 2023. Fine-grained commit-level vulnerability type prediction by CWE tree structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 957–969.
[138] Luca Pascarella, Fabio Palomba, and Alberto Bacchelli. 2019. Fine-grained just-in-time defect prediction. *Journal of Systems and Software* 150 (2019), 22–36.
[139] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. 2018. Vulnerable open source dependencies: Counting those that matter. In *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.
[140] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining zero-shot vulnerability repair with large language models. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2339–2356.
[141] Chanathip Pornprasit, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Michael Fu, and Patanamon Thongtanunam. 2021. Pyexplainer: Explaining the predictions of just-in-time defect models. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 407–418.
[142] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. 2021. JITLine: A simpler, better, faster, finer-grained just-in-time defect prediction. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 369–379.
[143] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. 2022. Deeplinedp: Towards a deep learning approach for line-level defect prediction. *IEEE Transactions on Software Engineering* 49, 1 (2022), 84–98.
[144] Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin Shar, Darius Foo, Andrew E Santosa, Asankhaya Sharma, and David Lo. 2021. Out of sight, out of mind? How vulnerable dependencies affect open-source projects. *Empirical Software Engineering* 26 (2021), 1–34.
[145] Luís Prates, João Faustino, Miguel Silva, and Rúben Pereira. 2019. Devsecops metrics. In *Information Systems: Research, Development, Applications, Education: 12th SIGSAND/PLAIS EuroSymposium 2019, Gdansk, Poland, September 19, 2019, Proceedings 12*. Springer, 77–90.
[146] Fangcheng Qiu, Zhipeng Gao, Xin Xia, David Lo, John Grundy, and Xinyu Wang. 2021. Deep just-in-time defect localization. *IEEE Transactions on Software Engineering* 48, 12 (2021), 5068–5086.
[147] Fangcheng Qiu, Meng Yan, Xin Xia, Xinyu Wang, Yuanrui Fan, Ahmed E Hassan, and David Lo. 2020. JITO: a tool for just-in-time defect identification and localization. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 1586–1590.

[148] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.
[149] Saima Rafi, Wu Yu, Muhammad Azeem Akbar, Ahmed Alsanad, and Abdu Gumaei. 2020. Prioritization based taxonomy of DevOps security challenges using PROMETHEE. *IEEE Access* 8 (2020), 105426–105446.
[150] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 164–175.
[151] Akond Rahman and Laurie Williams. 2018. Characterizing defective configuration scripts used for continuous deployment. In *2018 IEEE 11th International conference on software testing, verification and validation (ICST)*. IEEE, 34–45.
[152] Akond Rahman and Laurie Williams. 2019. Source code properties of defective infrastructure as code scripts. *Information and Software Technology* 112 (2019), 148–163.
[153] Roshan N Rajapakse, Mansooreh Zahedi, M Ali Babar, and Haifeng Shen. 2022. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and software technology* 141 (2022), 106700.
[154] RedHat. 2022. What is Infrastructure as Code (IaC)? https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac.
[155] RedHat. 2023. What is CI/CD security? https://www.redhat.com/en/topics/security/what-is-cicd-security.
[156] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.
[157] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).
[158] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *Journal of Systems and Software* 158 (2019), 110406.
[159] Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. 2018. Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 757–762.
[160] Mary Sánchez-Gordón and Ricardo Colomo-Palacios. 2020. Security as culture: a systematic literature review of DevSecOps. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 266–269.
[161] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. 2018. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software* 139 (2018), 84–106.
[162] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*. Springer, 146–157.
[163] Adriana Sejfia, Satyaki Das, Saad Shafiq, and Nenad Medvidović. 2023. Toward Improved Deep Learning-based Vulnerability Detection. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 730–741.
[164] Mojtaba Shahin, Mansooreh Zahedi, Muhammad Ali Babar, and Liming Zhu. 2019. An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering* 24 (2019), 1061–1108.
[165] Chenze Shao and Yang Feng. 2022. Overcoming catastrophic forgetting beyond continual learning: Balanced training for neural machine translation. *arXiv preprint arXiv:2203.03910* (2022).
[166] Adam Shostack. 2014. *Threat modeling: Designing for security*. John Wiley & Sons.
[167] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International Conference on Machine Learning*. PMLR, 3145–3153.
[168] Yangyang Shu, Yulei Sui, Hongyu Zhang, and Guandong Xu. 2020. Perf-AL: Performance prediction for configurable software through adversarial learning. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.
[169] Mohammed Latif Siddiq, Md Rezwanur Rahman Jahin, Mohammad Rafid Ul Islam, Rifat Shahriyar, and Anindya Iqbal. 2021. Sqlifix: Learning based approach to fix sql injection vulnerabilities in source code. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 354–364.
[170] Snyk. 2015. Snyk. https://snyk.io.
[171] Snyk. 2022. Software dependencies: How to manage dependencies at scale. https://snyk.io/series/open-source-security/software-dependencies/.
[172] Sonatype. 2024. Sonatype. https://www.sonatype.com/.

[173] Daniel Stahl, Torvald Martensson, and Jan Bosch. 2017. Continuous practices and devops: beyond the buzz, what does it all mean?. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 440–448.

[174] Benjamin Steenhoek, Hongyang Gao, and Wei Le. 2023. Dataflow Analysis-Inspired Deep Learning for Efficient Vulnerability Detection. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 166–178.

[175] Benjamin Steenhoek, Md Mahbubur Rahman, Richard Jiles, and Wei Le. 2023. An empirical study of deep learning models for vulnerability detection. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2237–2248.

[176] Hudan Studiawan, Ferdous Sohel, and Christian Payne. 2020. Anomaly detection in operating system logs with deep learning-based sentiment analysis. *IEEE Transactions on Dependable and Secure Computing* 18, 5 (2020), 2136–2148.

[177] Alexander Suh. 2020. Adapting bug prediction models to predict reverted commits at Wayfair. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1251–1262.

[178] Yan Sun, Celia Chen, Qing Wang, and Barry Boehm. 2017. Improving missing issue-commit link recovery using positive and unlabeled data. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 147–152.

[179] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive Network: A Successor to Transformer for Large Language Models. arXiv:2307.08621 [cs.CL]

[180] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*. PMLR, 3319–3328.

[181] Synopsys. 2024. Black Duck Software Composition Analysis. https://www.synopsys.com/.

[182] Pierre Tempel and Eric Tooley. 2024. Found means fixed: Introducing code scanning autofix, powered by GitHub Copilot and CodeQL. https://github.blog/2024-03-20-found-means-fixed-introducing-code-scanning-autofix-powered-by-github-copilot-and-codeql/.

[183] Hailemelekot Demtse Tessema and Surafel Lemma Abebe. 2021. Enhancing just-in-time defect prediction using change request-based metrics. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 511–515.

[184] Testsigma. 2023. Impact Analysis In Software Testing- A Complete Overview. https://testsigma.com/blog/impact-analysis-in-testing/.

[185] Richard J Turver and Malcolm Munro. 1994. An early impact analysis technique for software maintenance. *Journal of Software Maintenance: Research and Practice* 6, 1 (1994), 35–52.

[186] Akshay Utture and Jens Palsberg. 2023. From Leaks to Fixes: Automated Repairs for Resource Leak Warnings. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 159–171.

[187] Yolanda Valdés-Rodríguez, Jorge Hochstetter-Diez, Jaime Díaz-Arancibia, and Rodrigo Cadena-Martínez. 2023. Towards the Integration of Security Practices in Agile Software Development: A Systematic Mapping Review. *Applied Sciences* 13, 7 (2023), 4578.

[188] Validata. 2024. AI-powered Live Impact Analysis. https://www.validata-software.com/products/validata-sense-ai/ai-powered-live-impact-analysis.

[189] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[190] Vy Vo, Van Nguyen, Trung Le, Quan Hung Tran, Gholamreza Haffari, Seyit Camtepe, and Dinh Phung. 2022. An additive instance-wise approach to multi-class model interpretation. *arXiv preprint arXiv:2207.03113* (2022).

[191] Tianyi Wang, Shengzhi Qin, and Kam Pui Chow. 2021. Towards Vulnerability Types Classification Using Pure Self-Attention: A Common Weakness Enumeration Based Approach. In *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*. IEEE, 146–153.

[192] Wenbo Wang, Tien N Nguyen, Shaohua Wang, Yi Li, Jiyuan Zhang, and Aashish Yadavally. 2023. DeepVD: Toward Class-Separation Features for Neural Network Vulnerability Detection. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2249–2261.

[193] Xuheng Wang, Jiaxing Song, Xu Zhang, Junshu Tang, Weihe Gao, and Qingwei Lin. 2023. LogOnline: A Semi-Supervised Log-Based Anomaly Detector Aided with Online Learning Mechanism. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 141–152.

[194] Xinda Wang, Shu Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. 2020. A machine learning approach to classify security patches into vulnerability types. In *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–9.

[195] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922* (2023).

[196] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[197] Laura Wartschinski, Yannic Noller, Thomas Vogel, Timo Kehrer, and Lars Grunske. 2022. VUDENC: vulnerability detection with deep learning on a natural codebase for Python. *Information and Software Technology* 144 (2022), 106809.

[198] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A systematic literature review on the use of deep learning in software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–58.

[199] Jason Wei and Kai Zou. 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 6382–6388.

[200] David A. Wheeler. 2024. Flawfinder. https://dwheeler.com/flawfinder/.

[201] Emily Rowan Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, John Woodward, Serkan Kirbas, Etienne Windels, Olayori McBello, et al. 2022. Towards developer-centered automatic program repair: findings from Bloomberg. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1578–1588.

[202] Bozhi Wu, Shangqing Liu, Yang Xiao, Zhiming Li, Jun Sun, and Shang-Wei Lin. 2023. Learning Program Semantics for Vulnerability Detection via Vulnerability-Specific Inter-procedural Slicing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1371–1383.

[203] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 15–25.

[204] Xingfang Wu, Heng Li, and Foutse Khomh. 2023. On the effectiveness of log representation for log-based anomaly detection. *Empirical Software Engineering* 28, 6 (2023), 137.

[205] Liang Xi, Dehua Miao, Menghan Li, Ruidong Wang, Han Liu, and Xunhua Huang. 2023. Adaptive-Correlation-aware Unsupervised Deep Learning for Anomaly Detection in Cyber-physical Systems. *IEEE Transactions on Dependable and Secure Computing* (2023).

[206] Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 959–971.

[207] Yuanjie Xia, Zishuo Ding, and Weiyi Shang. 2023. CoMSA: A Modeling-Driven Sampling Approach for Configuration Performance Testing. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1352–1363.

[208] Zhe Xie, Changhua Pei, Wanxue Li, Huai Jiang, Liangfei Su, Jianhui Li, Gaogang Xie, and Dan Pei. 2023. From Point-wise to Group-wise: A Fast and Accurate Microservice Trace Anomaly Detection Approach. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1739–1749.

[209] Qinghua Xu, Shaukat Ali, and Tao Yue. 2021. Digital twin-based anomaly detection in cyber-physical systems. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 205–216.

[210] Qinghua Xu, Shaukat Ali, and Tao Yue. 2023. Digital Twin-based Anomaly Detection with Curriculum Learning in Cyber-physical Systems. *ACM Transactions on Software Engineering and Methodology* (2023).

[211] Qinghua Xu, Shaukat Ali, Tao Yue, Zaimovic Nedim, and Inderjeet Singh. 2023. KDDT: Knowledge Distillation-Empowered Digital Twin for Anomaly Detection. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1867–1878.

[212] Tianyin Xu and Yuanyuan Zhou. 2015. Systems approaches to tackling configuration errors: A survey. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–41.

[213] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. 2014. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 590–604.

[214] Meng Yan, Xin Xia, Yuanrui Fan, Ahmed E Hassan, David Lo, and Shanping Li. 2020. Just-in-time defect identification and localization: A two-phase framework. *IEEE Transactions on Software Engineering* 48, 1 (2020), 82–101.

[215] Lin Yang, Junjie Chen, Shutao Gao, Zhihao Gong, Hongyu Zhang, Yue Kang, and Huaan Li. 2023. Try with Simpler–An Evaluation of Improved Principal Component Analysis in Log-based Anomaly Detection. *ACM Transactions on Software Engineering and Methodology* (2023).

[216] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium*

(USENIX Security 21). 2327–2344.

[217] Lanxin Yang, He Zhang, Haifeng Shen, Xin Huang, Xin Zhou, Guoping Rong, and Dong Shao. 2021. Quality assessment in systematic literature reviews: A software engineering perspective. *Information and Software Technology* 130 (2021), 106397.

[218] Xu Yang, Shaowei Wang, Yi Li, and Shaohua Wang. 2023. Does data sampling improve deep learning-based vulnerability detection? Yeas! and Nays!. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2287–2298.

[219] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32 (2019).

[220] Bin Yuan, Yifan Lu, Yilin Fang, Yueming Wu, Deqing Zou, Zhen Li, Zhi Li, and Hai Jin. 2023. Enhancing Deep Learning-based Vulnerability Detection by Building Behavior Graph Model. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2262–2274.

[221] Lun-Pin Yuan, Peng Liu, and Sencun Zhu. 2021. Recompose event sequences vs. predict next events: A novel anomaly detection approach for discrete event logs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 336–348.

[222] Kev Zettler. 2022. The DevSecOp tools that secure DevOps workflows. https://www.redhat.com/en/topics/devops/what-is-devsecops.

[223] Chunyong Zhang, Bin Liu, Yang Xin, and Liangwei Yao. 2023. CPVD: Cross Project Vulnerability Detection Based On Graph Attention Network And Domain Adaptation. *IEEE Transactions on Software Engineering* (2023).

[224] Chenyuan Zhang, Yanlin Wang, Zhao Wei, Yong Xu, Juhong Wang, Hui Li, and Rongrong Ji. 2023. EALink: An Efficient and Accurate Pre-trained Framework for Issue-Commit Link Recovery. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 217–229.

[225] Junwei Zhang, Zhongxin Liu, Xing Hu, Xin Xia, and Shanping Li. 2023. Vulnerability Detection by Learning from Syntax-Based Execution Paths of Code. *IEEE Transactions on Software Engineering* (2023).

[226] Quanjun Zhang, Chunrong Fang, Bowen Yu, Weisong Sun, Tongke Zhang, and Zhenyu Chen. 2023. Pre-trained model-based automated software vulnerability repair: How far are we? *IEEE Transactions on Dependable and Secure Computing* (2023).

[227] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[228] Yunhua Zhao, Kostadin Damevski, and Hui Chen. 2023. A systematic survey of just-in-time software defect prediction. *Comput. Surveys* 55, 10 (2023), 1–35.

[229] Yunhui Zheng, Saurabh Pujar, Burn Lewis, Luca Buratti, Edward Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. 2021. D2a: A dataset built for ai-based vulnerability detection methods using differential analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.

[230] Junwei Zhou, Yijia Qian, Qingtian Zou, Peng Liu, and Jianwen Xiang. 2022. DeepSyslog: Deep Anomaly Detection on Syslog Using Sentence Embedding and Metadata. *IEEE Transactions on Information Forensics and Security* 17 (2022), 3051–3061.

[231] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems* 32 (2019).

[232] Qihao Zhu, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. 2021. A syntax-guided edit decoder for neural program repair. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 341–353.

[233] Qihao Zhu, Zeyu Sun, Wenjie Zhang, Yingfei Xiong, and Lu Zhang. 2023. Tare: Type-aware neural program repair. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1443–1455.

[234] Armin Zirak and Hadi Hemmati. 2022. Improving automated program repair with domain adaptation. *ACM Transactions on Software Engineering and Methodology* (2022).

[235] Deqing Zou, Yutao Hu, Wenke Li, Yueming Wu, Haojun Zhao, and Hai Jin. 2022. mVulPreter: A Multi-Granularity Vulnerability Detection System With Interpretations. *IEEE Transactions on Dependable and Secure Computing* (2022).

[236] Deqing Zou, Sujuan Wang, Shouhuai Xu, Zhen Li, and Hai Jin. 2019. μVulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection. *IEEE Transactions on Dependable and Secure Computing* 18, 5 (2019), 2224–2236.

Received 8 April 2024