Journal of Applied Automation Technologies

Research Paper

Supply Chain Software Package Risk Scoring TD3-MPC Model Based on Dependency Metadata Graph

Hsien-yu Teng[1,*] and Ya-chieh Tien[1]

[1] College of Electrical Engineering and Computer Science, National Chung Hsing University, Taichung, 40227, China
*Corresponding author: teng.hsieny@nchu.edu.tw

Abstract. In this paper, we develop a TD3-MPC model for software package risk rating using dependency information graphs. Because vulnerability labels, registry metadata, maintainer signals, repository integrity, license information, download modifications, dependence depth, and transitive exposure are dispersed and out-of-date, it is challenging to prioritise package inspection in vast open-source supply chains. Package age, release activity, maintainer continuity, vulnerability history, metadata missingness, dependency depth, and downstream exposure are summarised by graph states, which encapsulate package, version, maintainer, repository, license, and vulnerability nodes as a typed graph. Through twin critics, the TD3 module learns a continuous review-priority action. The MPC layer modifies this action based on analyst capacity restrictions and a short-horizon risk budget. For assessment, a simulated cross-ecosystem dataset of 126,000 package records from the Go, PyPI, Maven, and npm modules is utilised. In terms of the top-K hazardous package hit rate (88.9% vs. 78.6%), the high-risk F1 score (0.887 vs. 0.812), and the decrease in false review load (24.3% lower), the suggested model performed better than graph neural scoring and TD3 without predictive correction. According to the investigations, graph propagation reduces MPC instability during version bursts and missing-metadata occurrences and improves the accuracy of deep dependency identification. Before practical deployment, authentic SBOM records, registry histories, verified harmful labels, and code-level behaviour proof are still necessary.

Keywords: *Dependency Metadata Graph, TD3-MPC, Constrained Automatic Task Scheduling, Graph-aware State Perception, Risk-oriented Decision-making, Software Supply-chain Risk Scoring*

Received on 27 December 2025, Accepted on 17 May 2026, Published on 29 May 2026

Introduction

At present, cloud services, workplace applications, mobile systems, machine learning platforms and embedded devices all use many open-source packages. A single application may have hundreds of direct dependencies and thousands of transitive components, many of which are maintained by organisations outside the one deploying the application. Reuse speeds up development; however, it also increases the scope of attack for software supply-chain attacks. Zheng et al. [1] have shown that the malicious package insertion, typosquatting, dependency confusion and maintainer compromise are typical ways of downstream compromise. Gokkaya et al. [2] proposed software bill of materials records as an organised way to record package names, versions, providers and dependency links. However, inventory visibility is only the first stage. When alerts, weak metadata signals and dependency modifications occur at the same time, security engineers still need to determine which package is more urgent to investigate.

Package risk is relative and time-varying, not a fixed attribute. A package that does not contain known vulnerabilities may still be risky if it has an irregular release schedule, a sudden change in maintainers, missing repository data, inconsistent licensing, or abnormally rapid growth of dependencies. A vulnerable component can also be found at any level in the dependency tree. Xia et al. [3] have shown that dependency networks are concentrated, and widely used packages can affect many downstream applications. Wu et al. [4] have also found

that the old or unused dependencies are still in use. Research on malicious packages shows that registry behaviour and metadata are often early indicators before full code-level analysis is available [5]. Thus, risk assessment cannot be reduced to a single severity value or a uniform local feature vector.

A graph model is a suitable representation of dependency risk because it can connect package nodes, version nodes, maintainer nodes, repository nodes, license nodes and vulnerability nodes via relationships such as dependency, release, ownership, repository, license and vulnerability. Research in graph learning shows that neighbourhood aggregation can capture higher-order relational evidence [6]. However, package review priority is also an operational decision and not solely the result of classification. Security teams have noisy ranked lists and limited inspection capabilities. Safe reinforcement learning limits the learned behaviour according to the deployment cost [7], and learning-based model predictive control adjusts actions under short-horizon constraints [8]. Research in explainable AI also needs to link model outputs with human-understandable evidence [9].

This paper introduces a TD3-MPC software package risk-ranking model based on dependency metadata graphs. The first part introduces supply-chain risk and delayed evidence, and the necessity of graph-based review prioritisation. The second is dependency governance, metadata risk evidence, graph scoring, constrained decision models and explainable security analytics. The third is Dependency Metadata Graph Construction, Graph State Encoding, TD3 Priority Learning, Reward Design and MPC Queue Correction. The fourth is the evaluation of the model on simulated multi-ecosystem package data, including dataset evidence, scoring accuracy, robustness, ablation analysis and engineering queue interpretation. The fifth is the summary and limitations.

Related Work

Dependency Governance and Software Supply Chain Security

Component identification is the first step in dependency governance, and risk analysis is added over time. A component inventory keeps track of the packages that are included in a build, but it does not automatically show if the package is accessible, safe, updated, or compatible with policy. Standardised package records can enhance vulnerability response and procurement transparency in SBOMs, according to research [10]. However, package ecosystems are changing quickly. Once a maintainer account is changed, a dependency is added, or a vulnerability is revealed, a version that was deemed safe during the first review might become dangerous. Dependency governance will not be regarded as a fixed procurement item and will keep an eye on changes to the package information.

Most attacks on open-source packages involve the modification of metadata. Malicious installation scripts misuse package lifecycle hooks, typosquatting packages mimic well-known names, and dependency confusion packages take advantage of build-source precedence. Zhou et al. [11] have shown that package names, release schedules, repository signals, maintainer accounts and installation behaviour can all be used for detection. Ohm et al. [12] have also shown that automated analysis can be performed on registry metadata and package contents to reduce manual examination. Based on the above, metadata signals can be used; however, for the same reason, a single rule is insufficient. The choice should consider the context of dependency because both dangerous and innocent new packages may have short histories.

Although they are not the same as package risks, known vulnerability ratings are still required. The significance of a vulnerability is indicated by CVSS severity, and operational package risk is further influenced by the availability of the vulnerable package, the existence of a patched version, the degree of transitivity of the impacted package, and the use of the vulnerable function by the organization. The possibility of an exploit and the viability of repair have an impact on the actual security decisions, according to research on vulnerability prioritisation [13]. Defect and vulnerability labelling are loud, incomplete, and delayed, according to software vulnerability discovery research [14]. Because of this, the model in this research does not use the whole score result and instead views vulnerability evidence as a single signal inside a wider metadata graph.

An operation queue is required for the final package operation. The security team can't scan every dependency in every repository; therefore, they will only perform a few high-risk manuals checks each day. Many of the notifications lead developers to neglect or postpone dependency maintenance, according to research on

automated dependency update techniques [15]. As a result, urgent items, compelling proof, and thorough evaluations should be given greater weight in the scoring methodology. Compared to a package with merely a weak missing-field signal, a package with a direct vulnerability, deep dependency exposure, and a maintainer anomaly should be reported sooner. For this reason, the TD3-MPC design was chosen for this work.

Explainable Risk Scoring with Graph and Control Models

For risk assessment issues involving relational data, graph neural networks are suitable. Depending on where it is in the dependency tree, the same package in the software dependency ecosystem may have various downstream consequences. Local and higher-order structure can be represented via neighbourhood aggregation and message passing, according to graph neural network studies [16]. Score for the supply chain: This will give evidence from questionable, abandoned, or vulnerable upstream packages more weight when downstream packages are reviewed. Security engineers are aware of how a danger signal will spread throughout the dependency structure since the graph also displays the evidence's journey.

It must be interpretable for use in risk scoring. The review team should be aware of whether a package is flagged as high risk by a system due to direct vulnerability, maintainer instability, transitive exposure, atypical release frequency, missing repository link, or license-policy conflict. Decision support should connect forecasts to comprehensible explanations and the user's goal, according to explainable AI research [17]. Thus, these should not be general model narratives in software supply chain governance. Indicate which evidence categories, graph pathways, and package information can be audited at the time of the audit.

Reinforcement learning and continuous review priority are inextricably linked. A review action can be given a weight to alter the queue order and analyst emphasis; it is not a yes-or-no flag. By using twin critics and delayed policy updates, TD3-style actor-critic learning tackles the issue of overestimation in continuous control [18]. The issue of excessive priority increases brought on by noisy data may be resolved in this way. On the basis of incentives alone, however, it is still prone to overreact when several packages have abrupt version changes or missing information at the same time.

By taking into account a limited planning horizon, model predictive control can rectify the learnt review activities. The issue of restrictions has been addressed by MPC in learning-based control [19]. Physical safety is not a barrier in package review; instead, analyst capacity, false-alert tolerance, and risk-budget stability are. Constrained choice models are appropriate when an action that seems advantageous locally has an unacceptably large deployment cost, according to safe learning research [20]. Thus, prior to being put to the engineering review queue, the suggested model employs TD3 to determine the review priority and MPC to modify it.

Risk Scoring Framework

Dependency Metadata Graph Encoding

The software package ecosystem is represented as a dependency metadata graph. In the simulated enterprise repository, the graph contains 126,000 package records from npm, PyPI, Maven, and Go modules, including 41,800 direct dependency packages and 84,200 transitive dependency packages. After metadata cleaning, 18 package-level attributes are retained, including package age, release interval, version count, maintainer number, repository-link status, license state, download variation, dependency depth, known vulnerability count, and abnormal release flag. Package nodes, version nodes, maintainer nodes, repository nodes, license nodes, and vulnerability nodes are connected through dependency, release, ownership, repository, license, and vulnerability edges. This structure allows direct package evidence and transitive dependency exposure to be evaluated together and provides the input representation for the workflow.

$$G = (V, E, X, A) \quad \text{Eq.(1)}$$

In this graph, V denotes all metadata-related nodes, E denotes the typed relation set, $X \in R^{126000\times 18}$ is the node feature matrix, and A is the weighted adjacency matrix. The final constructed graph contains 126,000 package nodes and 318,400 dependency edges. The average dependency depth is 2.7, while 13.6% of packages appear at depth greater than 4. These deeper packages are retained because they may introduce hidden supply-chain exposure even when they are not directly imported by the application.

$$x_i = [\text{ age }_i, \text{ version }_i, \text{ maintainer }_i, \text{ download }_i, \text{ license }_i, \text{ vuln }_i] \quad \text{Eq.(2)}$$

The metadata vector of package i records package age, version activity, maintainer continuity, download variation, license state, and vulnerability history. Each feature is normalized into the interval [0,1]. For example, package age is clipped at 60 months, version activity is calculated over a 90-day release window, and download variation is measured by the relative change between the current 30-day window and the previous 30-day window. A package with stable releases and clear maintainer identity receives a weaker anomaly signal than a package with sudden release bursts, missing repository links, and inconsistent license metadata.

$$w_{ij} = \text{ adep }_{ij} + \beta \text{ depth }_{ij} + \gamma \text{ exposure }_{ij} \quad \text{Eq.(3)}$$

The propagation weight measures how strongly risk evidence passes from one package to another along the dependency path. Direct dependency relations receive the largest weight, while path depth and downstream exposure provide correction so that weak distant evidence is attenuated but widely reused upstream packages remain visible. The exposure term is calculated from the number of downstream packages affected by the dependency path. Figure 1 shows the overall dependency governance workflow. SBOM records, package metadata, vulnerability evidence, repository status, and maintainer signals are converted into a dependency graph, then encoded for TD3-MPC risk scoring and review queue generation.

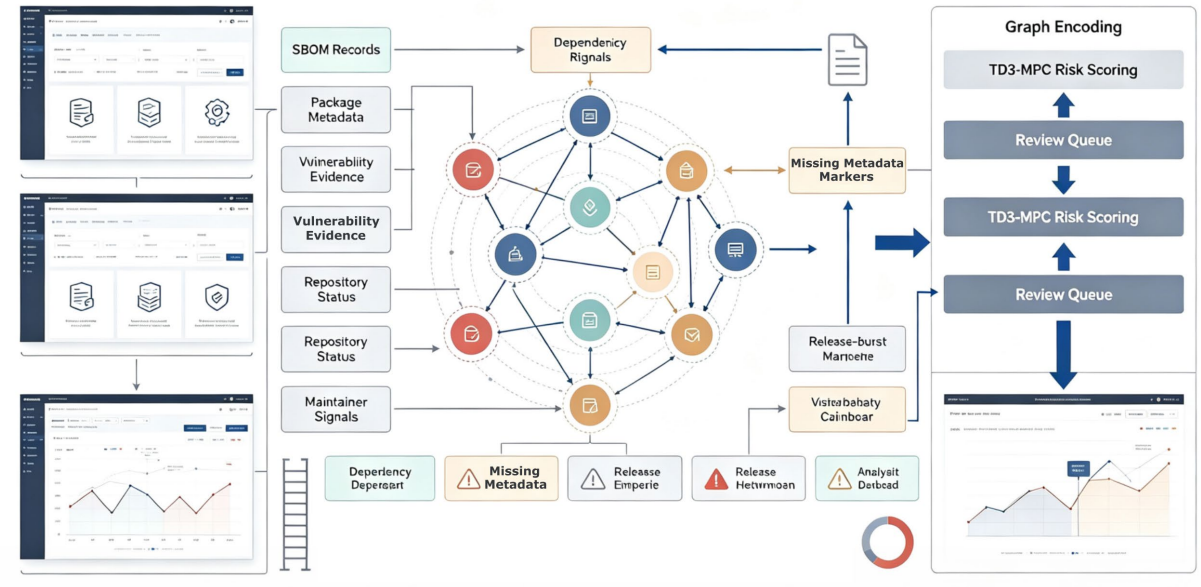

Figure 1. Dependency metadata graph workflow for supply chain package risk scoring.

TD3-MPC Review Priority Optimization

After creating a graph, translate the dependence data to a state. The package is not presented by the State in a vacuum. Keep track of package-level metadata, vulnerability evidence, transitive exposure, dependency neighbourhood, metadata reliability, and review-resource context. In the simulated review scenario, put the high-risk queue capacity at 35 packages, the daily manual review capacity at 120 packages, and the maximum allowable false-review rate at 0.18. These settings establish the state for the TD3-MPC optimisation structure and make the learnt score commensurate with an actual security team workload.

$$s_t = \text{GraphEnc}(G_t, X_t) \quad \text{Eq.(4)}$$

The graph encoder produces the state s_t at time t. The hidden dimension is set to 128, and two graph aggregation layers are used to cover direct and second-order dependency neighborhoods. The graph state is recalculated every 24 hours. When package metadata changes, the updated state reflects new releases, dependency additions, maintainer changes, and vulnerability updates. This makes the risk score sensitive to supply-chain dynamics while preserving the structural dependency path that explains why a package is important.

$$a_t = \pi_\theta(s_t) \quad \text{Eq.(5)}$$

The actor network maps the graph state into a continuous review-priority action. The action value is normalized into $[0,1]$, where values above 0.75 enter the urgent review queue, values between 0.45 and 0.75 enter the monitored queue, and values below 0.45 remain in routine dependency tracking. Unlike binary classification, this continuous action can express different levels of urgency, which is useful when several packages are suspicious but review capacity is limited.

$$Q_k(s_t, a_t) = r_t + \lambda \min\left(Q_1'(s_{t+1}, a_{t+1}), Q_2'(s_{t+1}, a_{t+1})\right) \quad \text{Eq.(6)}$$

The twin critics evaluate whether the selected review-priority action improves risky-package discovery while controlling review cost. In the experiment, $\lambda = 0.98$, batch size is 256, and delayed actor update is performed every two critic updates. The minimum of two target critics is used to reduce overestimation, because an overly optimistic priority estimate may push noisy packages into the review queue and increase false workload.

$$r_t = \text{hit}_t - \eta\, \text{false}_t - \mu\, \text{cost}_t \quad \text{Eq.(7)}$$

The reward function increases when high-risk packages are correctly placed in the priority queue and decreases when the system creates false alerts or excessive review cost. In the simulated setting, $\eta = 0.40$ and $\mu = 0.25$. A correct high-risk hit receives a reward of 1.0, while a false urgent alert receives a penalty proportional to review workload. This design makes the model sensitive to both security benefit and operational burden, which is necessary for practical software supply chain governance.

$$a_t^* = \arg\min_{a_{t:t+H}} \sum_h \left(C_r(s_{t+h}, a_{t+h}) + C_q(a_{t+h})\right) \quad \text{Eq.(8)}$$

The MPC layer corrects the learned action over a short horizon by balancing predicted risk cost and review-queue cost. The planning horizon represents a five-day review window, the risk budget is set to 0.35, and the queue overflow penalty is activated when more than 120 packages are pushed into daily manual review. If the TD3 actor assigns too many packages to high-priority review during a metadata burst, MPC suppresses weak or isolated alerts. If several evidence signals align, MPC preserves or strengthens the action. Figure 2 shows the internal TD3-MPC optimization structure: the graph encoder produces package states, twin critics evaluate review priority, the actor proposes continuous ranking actions, and the MPC correction layer filters the action through review-capacity, false-alert, and risk-budget constraints before outputting a final package queue.

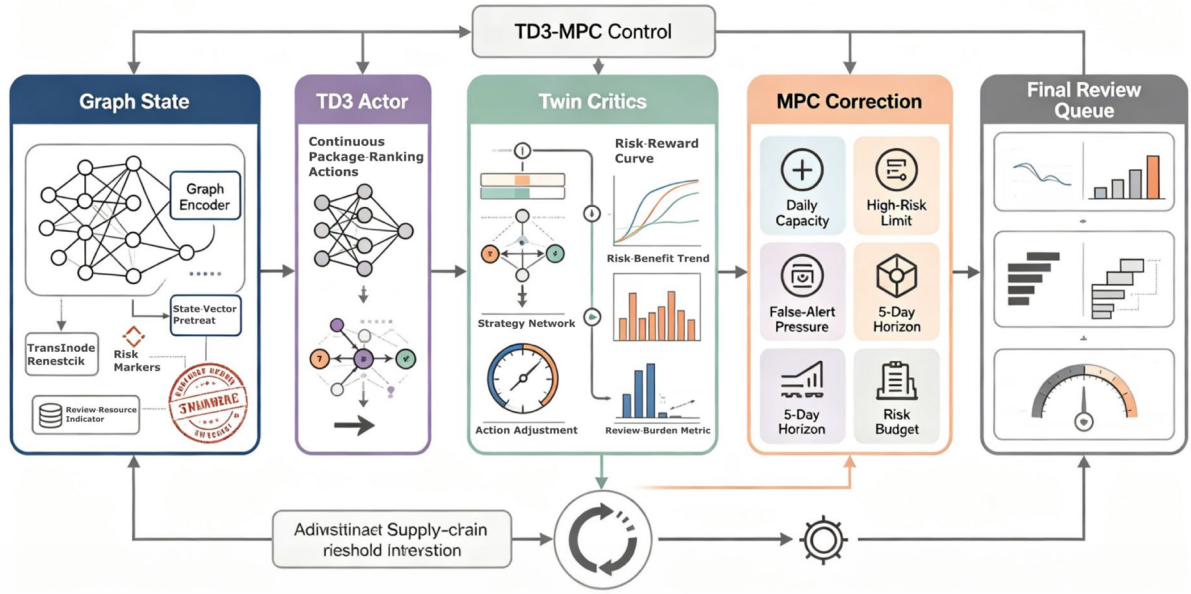

Figure 2. Internal TD3-MPC review-priority optimization structure.

https://doi.org/10.64972/jaat.2026v4.386p31e:413-425

Experiments and Discussion

Dependency Dataset and Risk Evidence

The experimental dataset contains 126,000 package records from npm, PyPI, Maven, and Go modules and is created by modeling a multi-ecosystem software supply-chain repository. Each record includes package name, version count, release interval, dependency list, maintainer count, repository URL, issue activity, licensing field, download trend, known vulnerability tag, and suspicious metadata label. Figure 3 shows the dependency dataset and risk evidence composition. Figure 3(a) reports ecosystem sources, where npm and PyPI occupy large proportions because of dense and fast dependency reuse. Figure 3(b) shows dependency depth, including deep packages that can create hidden exposure. Figure 3(c) separates inactive or single-maintainer packages from stable multi-maintainer packages. Figure 3(d) reports vulnerability history, suspicious release pattern, missing metadata, and maintainer-transfer evidence.

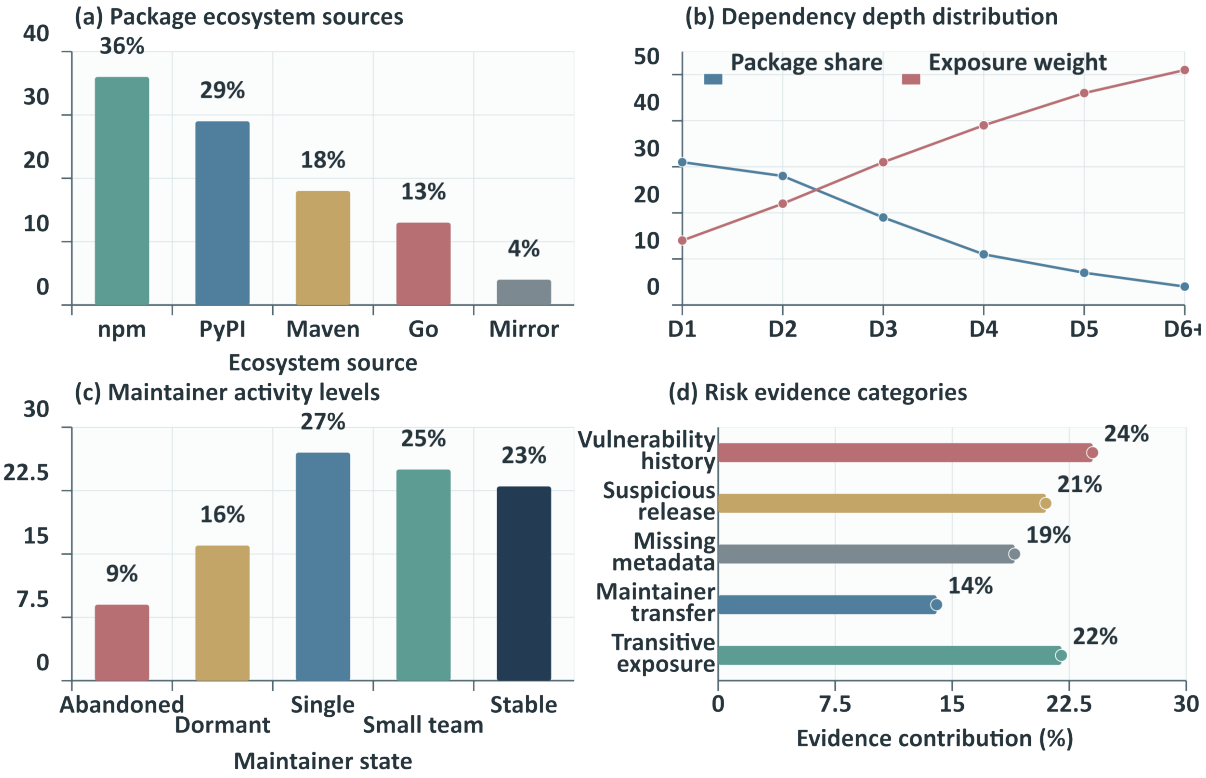

Figure 3. Dependency dataset and risk evidence composition. (a) Package ecosystem sources. (b) Dependency depth distribution. (c) Maintainer activity levels. (d) Risk evidence categories.

Instead of providing a clear benchmark, the dataset is meant to illustrate the challenges of engineering review. Not all of the evidentiary signs are present at the same time in many high-risk packages. While one software has a vulnerability but is actively maintained and promptly fixed, another may have no known CVE yet exhibit an unusual release frequency. Package ecosystem risk frequently results from a combination of dependence, maintainer, and release behaviour, according to software supply chain research [21]. As a result, a graph's form ought to be used. The model won't be able to identify hazards passed across transitive dependencies if it simply examines local information. It is unable to identify the early warning of a new vulnerability and just reads the vulnerability label. As a result, direct vulnerability risk, suspicious metadata risk, dependency exposure risk, and review-priority risk are all distinguished by the simulated labels.

Additionally, graph building will verify the correctness of the information. Weak but important evidence includes the lack of a repository URL, conflicting license data, missing maintainer information, and abrupt changes in download content. Malicious packages frequently employ names, release dates, and repository trust signals, according to package security research [22]. This experiment does not impose severe requirements on these weak signals. When paired with dependence exposure, this feature evidence might raise the review priority. As

a result, a single missing field won't cause several false warnings. In order to differentiate registry-derived signals, repository-derived signals, and vulnerability-database signals in a subsequent review, the model saves the kind of evidence source. The sources are not equal because of their varying levels of dependability. Vulnerability databases are reliable but out-of-date; registry metadata is quick but may be superficial; repository evidence is richer but may be lacking or fake. Prior to graph aggregation, store these signals in distinct channels and teach the model that the same missing field might imply various things depending on the situation. A missing field in a tiny internal utility mirror is not as suspicious as a missing repository link in an established and popular program.

Package groups are divided using splitting strategies based on release cycles and ecosystems. While testing incorporates more recent package releases and hidden dependency structures, training makes use of earlier package snapshots and a number of controlled attack-like behaviours. Because future hazardous packages can differ from past ones, this is more akin to software supply chain monitoring. In the research of vulnerability prediction, it has been demonstrated that temporal divides are more appropriate for software security analysis than random splits [23]. It is more challenging than a static categorisation problem since the test split additionally includes cold-start packages and package families with recently added transitive dependencies. Furthermore, the training and testing versions of the experiment are not quite identical. The model could learn version-family identity rather than risk evidence if the surrounding versions are randomly mixed. Rather than memorising package names, the release-period split should make advantage of dependency structure, maintainer continuity, repository integrity, and metadata trends. This is essential for corporate use since a potentially dangerous package may have a familiar name but a new transitive dependence, a suspicious release interval, or a changed maintainer. As a result, rather than re-examining previous situations, the assessment is probably more sensitive to the new package condition.

Risk Scoring Accuracy and Operational Robustness

The top-level risk score results are shown in Figure 4. Figure 4(a) compares high-risk classification accuracy for rule scoring, random forest, graph neural scoring, TD3 without MPC, and the proposed TD3-MPC model. Figure 4(b) reports Top-K hazardous package hit rate under different review budgets and shows that capacity-aware scoring is most useful when only a small number of packages can be inspected. Figure 4(c) reports false alert rate, where MPC adjustment filters weak or isolated evidence before it overloads the review queue. Figure 4(d) evaluates version perturbation robustness and shows whether rapid version changes destabilize package priority.

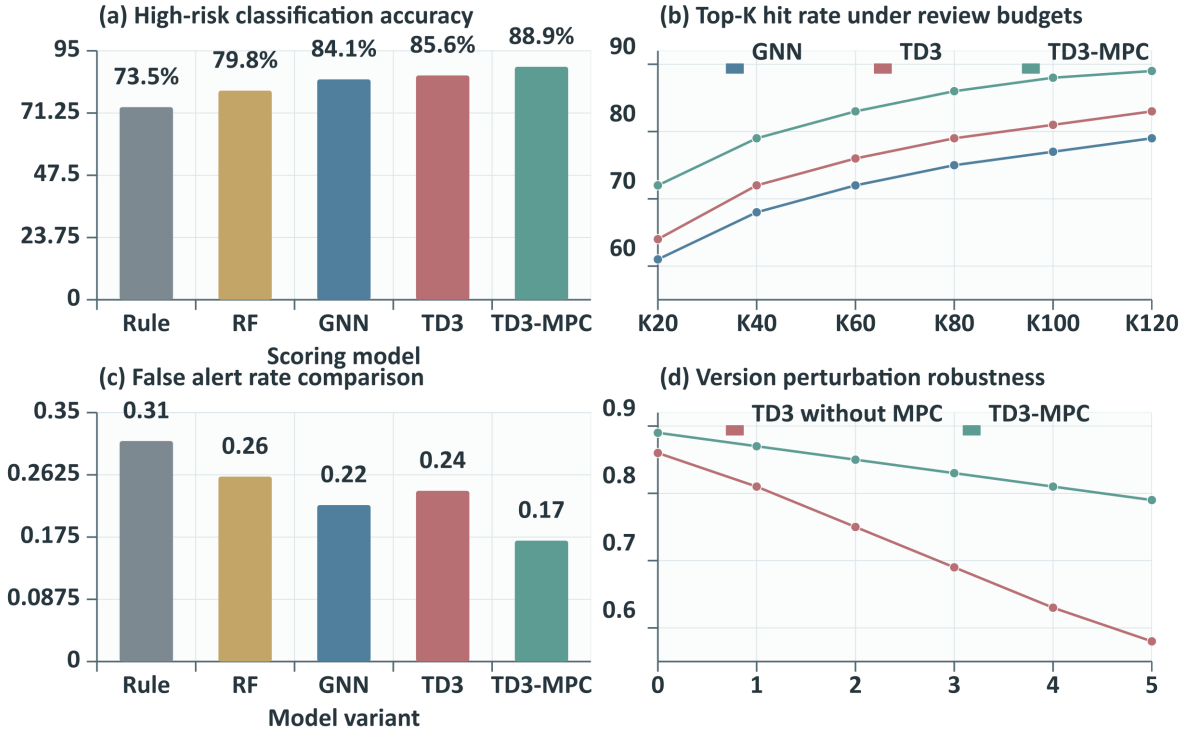

Figure 4. Risk scoring accuracy and operational robustness. (a) High-risk classification accuracy. (b) Top-K hit rate under review budgets. (c) False alert rate comparison. (d) Version perturbation robustness.

https://doi.org/10.64972/jaat.2026v4.386p31e:413-425

Table 1 provides a numerical comparison of the representative supply-chain package risk scoring approaches used in this section, including high-risk classification accuracy, high-risk F1, Top-K hit rate, false-alert rate, and robustness after version perturbation. Rule scoring and random forest baselines remain useful for direct vulnerability labels and local metadata, but their accuracy, F1, and Top-K hit rates are lower because they cannot fully trace transitive exposure. Graph neural scoring improves the high-risk F1 to 0.812 and raises the Top-K hit rate to 78.6% by propagating dependency evidence, yet its review order is still not adjusted by manual capacity. TD3 without MPC further improves accuracy to 85.6%, but the false-alert rate increases to 0.24 and the version-5 robustness score drops to 0.55 because the learned policy can overreact to metadata bursts. TD3-MPC achieves the best values, with 88.9% accuracy, 0.887 high-risk F1, 88.9% Top-K hit rate, 0.17 false-alert rate, and 0.76 robustness, showing that MPC correction links risk detection with stable review workload and deployable queue control.

Table 1. Operational comparison of representative supply-chain package risk scoring approaches.

Approach	Accuracy (%)	High-risk F1	Top-K hit rate (%)	False alert rate	Version-5 robustness	Review role
Rule scoring	73.5	0.742	58.0	0.31	0.70	Simple baseline
Random forest	79.8	0.779	65.4	0.26	0.72	Feature baseline
Graph neural scoring	84.1	0.812	78.6	0.22	0.74	Propagation baseline
TD3 without MPC	85.6	0.851	80.0	0.24	0.55	Policy baseline
TD3-MPC	88.9	0.887	88.9	0.17	0.76	Deployable priority model

The sequence of repair cannot be determined by a single vulnerability scanner based on the tests mentioned above. For packages with direct vulnerability labels, rule scoring works effectively, but for metadata-only risk, the ranking is erratic. While they improve the use of local features, Random Forests are unable to fully capture dependence pathways. Graph neural scoring does not learn review actions under cost, but it does capture propagation. A decent priority policy is learned by TD3 without MPC, but it is overly sensitive to noisy information. Reward-driven policies can overfit to local signals when operational restrictions are insufficient, according to research on reinforcement learning in cybersecurity [24]. To overcome the shortcoming, TD3-MPC includes a short-horizon corrective layer. The adjustment simply modifies the action when the near-term review queue will surpass capacity or when the same weak signal arises across too many packages following a registry-wide event. It does not forget the learnt rank. It is more capable of managing the release burst. For instance, TD3 will first see a package family's rapid release of several versions as worrisome; however, MPC will also investigate if maintainer continuity, dependency exposure, and vulnerability evidence have increased. The ultimate priority will be lowered if these supporting signals are absent. The priority is still high if the version explosion happens around maintainer handover or new deep dependencies. Serious package-state irregularities will still be detected, but the queue will be more stable and fewer low-value warnings will be produced.

There are additional challenging metadata issues for the model, as Figure 5 illustrates. The approach employs maintainer continuity, repository completeness, and dependency neighbourhood rather than popularity since Figure 5(a) displays cold-start packages with few downloads and a brief release history. A deep dependence exposure is shown in Figure 5(b), where the graph propagation connects the upstream risk to downstream systems that do not import the package directly. Metadata gaps, including missing repository links, license information, and issue data, are displayed in Figure 5(c). The method will evaluate the danger of incomplete fields by taking into account missingness, dependency position, and maintainer information.

https://doi.org/10.64972/jaat.2026v4.386p31e:413-425

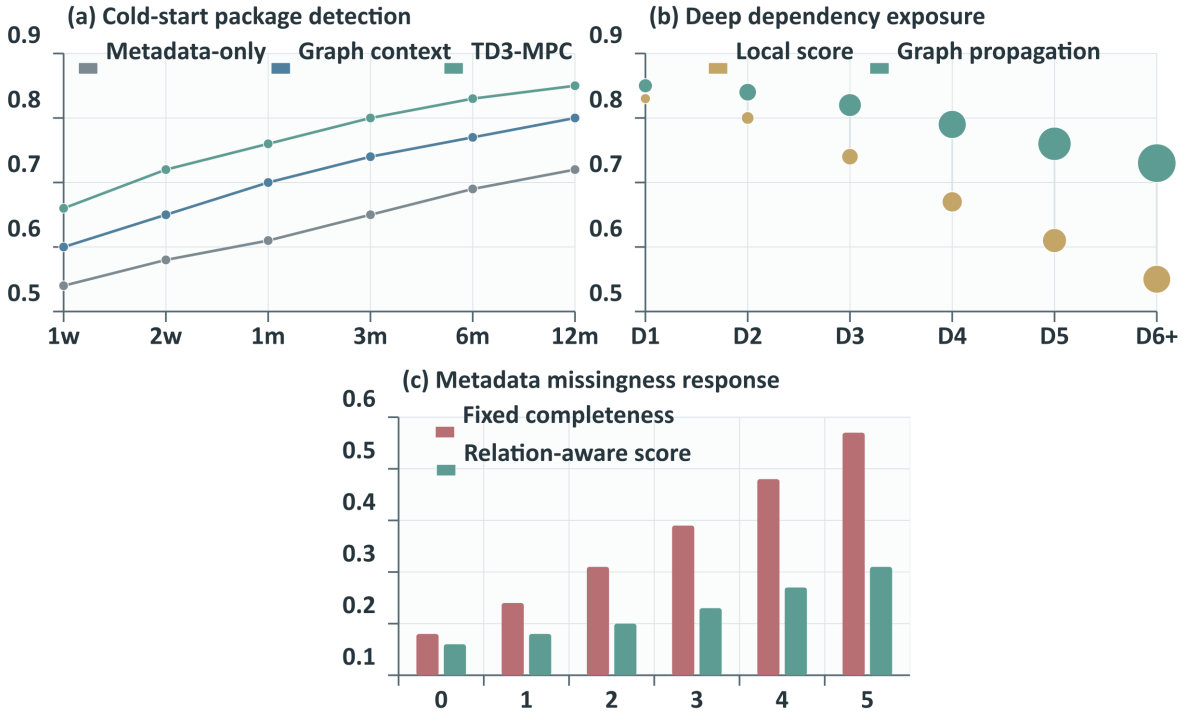

Figure 5. Risk scoring under difficult metadata conditions. (a) Cold-start package detection. (b) Deep dependency exposure. (c) Metadata missingness response.

The governance of the software supply chain can benefit from these findings. Attackers can take advantage of the fact that engineers often don't inspect packages until a vulnerability database mentions them. The first kind of detection employs code analysis along with behavioural and metadata evidence, according to research on malicious npm and PyPI packages [25]. The new approach will indicate a higher-priority location for engineers to investigate rather than taking the role of code review or sandbox analysis. Organisations with several repositories, constantly shifting dependencies, and multiple product lines utilising the same source components will find it ideal. Even if a shared upstream package could be modest in one repository, it must be obtained by several business systems. The model's downstream exposures to the same package evidence can be added using the graph view. As a result, the analyst may focus on packages that have failed or been compromised in many build pathways because there won't be any independent, low-impact warnings in the queue. The test's flaw is that missing metadata has to be addressed. A practical score system shouldn't be triggered by every incomplete field because some valid packages have non-standard repository procedures or missing issue reports. Therefore, a fixed metadata completeness score is not the realistic barrier. It is a relation-aware score that assesses if the package has sufficient alternative evidence to sustain the trust and whether the missing or unstable information occurs near a key dependency path.

Review Queue and Component Contribution

The component ablation results are shown in Figure 6. Figure 6(a) removes TD3 and keeps graph scoring only; dangerous packages can still be detected, but queue ordering is less responsive to review capacity, and some medium-risk packages are placed too high. Figure 6(b) removes MPC while retaining TD3 priority learning; dangerous packages are found earlier, but metadata noise and version bursts create more false urgent alerts. The ablation therefore shows that TD3 learns continuous review-priority behavior, whereas MPC stabilizes the action under short-term capacity and risk-budget constraints. This division is consistent with safe reinforcement learning research because learned behavior should maximize benefit while remaining within deployment limits [26].

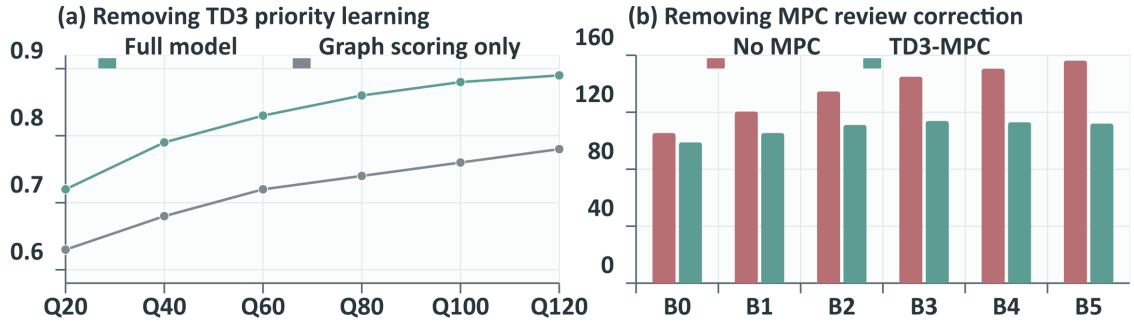

Figure 6. Component ablation of the TD3-MPC risk scoring model. (a) Removing TD3 priority learning. (b) Removing MPC review correction.

Engineering deployment interpretation is shown in Figure 7. Figure 7(a) presents a risk-ranked review queue and indicates whether each package was escalated because of vulnerability evidence, dependency exposure, metadata abnormality, repository integrity, license conflict, or maintainer risk. Figure 7(b) reports manual review workload reduction. The proposed model reduces unnecessary reviews by 24.3% while preserving high-risk discoveries, which matters because noisy systems are often ignored even when their algorithmic accuracy is high. Explainable security analytics research shows that analysts need explanations for actions and artifacts [27]. The queue view therefore helps security, development, and procurement teams coordinate whether a package should enter urgent block-list review, manual review, monitored tracking, or routine update management.

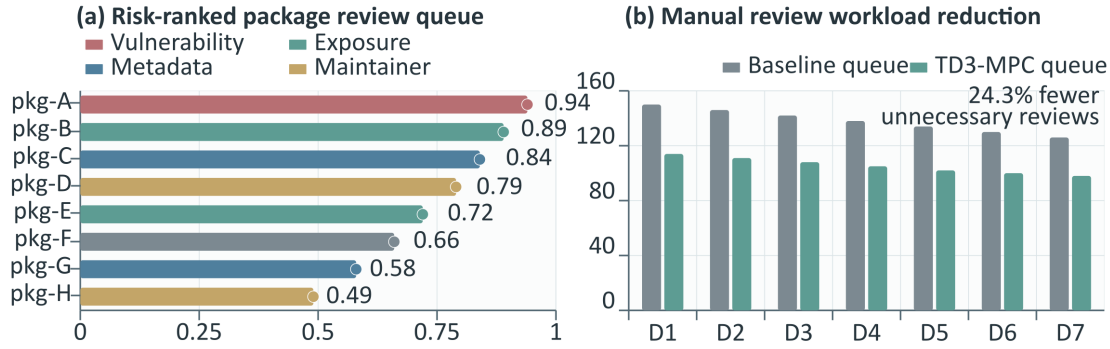

Figure 7. Engineering review queue interpretation. (a) Risk-ranked package review queue. (b) Manual review workload reduction.

Auditing is also supported by the deployed interpreter. Every high-priority package has a short evidence description, such as a vulnerable transitive parent, abandoned maintainer status, unusual version burst, missing repository link, questionable dependency growth, or license-policy conflict. Research in software analytics shows that the evidence of bugs and vulnerabilities may be missing or delayed [28]. The explanation in this model is connected to the TD3-MPC action and the metadata graph path. Therefore, the reviewer can determine whether the escalation resulted from a foreseen review-capacity limitation, an upstream dependency, or a direct vulnerability. The above evidence trail will also help the developer learn why the fix is required.

Ecosystem drift and label delay are still problems. A vulnerability may not be known at the time of training, and a package can be harmful before being added to the vulnerability database. Source-level learning can provide additional metadata evidence [29], and the semantic code features may have a hazardous implementation pattern [30]. Therefore, the proposed metadata graph should be employed as the first filter and is not the final selection. Code scanning, sandbox execution, provenance inspection and policy review should be triggered for high-priority packages. Machine learning for code research supports the combination of artifact-level and metadata-level evidence [31], and dependency-update research needs to keep automated notifications useful for developers [32]. Mining repositories and vulnerability discovery have also shown the value of historical security datasets [33], multi-domain knowledge [34], and graph-based package scoring [35].

Conclusion

This work presented a TD3-MPC model for software package risk evaluation based on dependency metadata graphs. Package metadata, dependency relations, maintainer signals, repository status, vulnerability evidence,

license information, and transitive exposure are encoded as a graph so that local anomalies and downstream impact can be evaluated together. TD3 learns continuous review-priority actions, and MPC corrects these actions using a short-term risk budget and manual-review capacity. The model therefore treats software package risk scoring as a capacity-aware governance problem rather than only a static classification problem.

According to the experiment, the new approach may decrease the amount of false review labour, rank high-risk packages more accurately, and maintain stability in the face of missing metadata and version perturbation. MPC correction decreased instability in reinforcement learning with noisy metadata, and propagation of the dependency network revealed a serious dependence exposure. As a result, the model offers a software supply chain risk score method that is more useful.

Several limitations remain. The simulated dataset should be extended with real SBOM records, package registry histories, validated malicious package labels, corporate dependency-use logs, and analyst review outcomes. Future research should incorporate code-level behavior features, sandbox execution traces, provenance inspection, registry-event timelines, cross-ecosystem transfer learning, and label-delay handling. These additions would help determine whether the TD3-MPC priority queue remains stable when registry activity, maintainer ownership, package popularity, vulnerability disclosure, and organizational dependency use change over time. The present model should therefore be interpreted as a practical triage layer for software supply-chain governance, not as a complete autonomous defense system.

Author Contributions

Hsien-yu Teng contributes to conceptualization, methodology, software, validation, analysis, investigation, data collection, draft preparation, manuscript editing, visualization, supervision. Ya-chieh Tien contributes to draft preparation, software, validation, analysis, investigation. All authors have read and agreed with the manuscript before its submission and publication.

Funding

This research received no specific financial support from any funding agency.

Institutional Review Board Statement

Not applicable.

Declaration of generative AI and AI-assisted technologies

During the preparation of this manuscript, the authors utilized Quillbot for linguistic polishing, logical organization and draft revision. The generative AI tool was not involved in research design, data collection, data analysis, or the formulation of core conclusions. All outputs generated by AI were reviewed, revised and verified manually by the authors. The authors take full responsibility for all content of this manuscript.

References

[1] Zheng, X., Wei, C., Wang, S., Zhao, Y., Gao, P., Zhang, Y., Wang, K., & Wang, H. (2024). Towards robust detection of open-source software supply chain poisoning attacks in industry environments. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (pp. 1990-2001). Association for Computing Machinery. https://doi.org/10.1145/3691620.3695262

[2] Gokkaya, B., Aniello, L., & Halak, B. (2026). Software supply chain: A taxonomy of attacks, mitigations and risk assessment strategies. Journal of Information Security and Applications, 97, 104324. https://doi.org/10.1016/j.jisa.2025.104324

[3] Xia, B., Bi, T., Xing, Z., Lu, Q., & Zhu, L. (2023). An empirical study on software bill of materials: Where we stand and the road ahead. Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, 2630-2642. https://doi.org/10.1109/ICSE48619.2023.00219

[4] Wu, Y., Yu, Z., Wen, M., Li, Q., Zou, D., & Jin, H. (2023). Understanding the threats of upstream vulnerabilities to downstream projects in the Maven ecosystem. Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, 1046-1058. https://doi.org/10.1109/ICSE48619.2023.00095

[5] Márquez, A. G., Varela-Vaca, Á. J., Gómez-López, M. T., Galindo, J. A., & Benavides, D. (2024). Vulnerability impact analysis in software project dependencies based on Satisfiability Modulo Theories (SMT). Computers & Security, 139, 103669. https://doi.org/10.1016/j.cose.2023.103669

[6] Halder, S., Bewong, M., Mahboubi, A., Jiang, Y., Islam, R., Islam, Z., Ip, R. H. L., Ahmed, M. E., Ramachandran, G. S., & Babar, M. A. (2024). Malicious package detection using metadata information. Proceedings of the ACM Web Conference 2024, 1779-1789. https://doi.org/10.1145/3589334.3645543

[7] Dai, E., Zhao, T., Zhu, H., Xu, J., Guo, Z., Liu, H., Tang, J., & Wang, S. (2024). A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. Machine Intelligence Research, 21(6), 1011-1061. https://doi.org/10.1007/s11633-024-1510-8

[8] Gu, S., Yang, L., Du, Y., Chen, G., Walter, F., Wang, J., & Knoll, A. (2024). A review of safe reinforcement learning: Methods, theories, and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence, 46(12), 11216-11235. https://doi.org/10.1109/TPAMI.2024.3457538

[9] Lin, M., Sun, Z., Xia, Y., & Zhang, J. (2024). Reinforcement learning-based model predictive control for discrete-time systems. IEEE Transactions on Neural Networks and Learning Systems, 35(3), 3312-3324. https://doi.org/10.1109/TNNLS.2023.3273590

[10] Rjoub, G., Bentahar, J., Abdel Wahab, O., Mizouni, R., Song, A., Cohen, R., Otrok, H., & Mourad, A. (2023). A survey on explainable artificial intelligence for cybersecurity. IEEE Transactions on Network and Service Management, 20(4), 5115-5140. https://doi.org/10.1109/TNSM.2023.3282740

[11] Zhou, X., Xu, J., Li, X., Jiang, Y., Zhang, Y., & Liu, X. (2024). A blockchain-based and microservices-architected software composition analysis system. Journal of Software: Evolution and Process, 36(10), e2675. https://doi.org/10.1002/smr.2675

[12] Ohm, M., & Stuke, C. (2023). SoK: Practical detection of software supply chain attacks. Proceedings of the 18th International Conference on Availability, Reliability and Security. https://doi.org/10.1145/3600160.3600162

[13] Gao, K., Xu, W., Yang, W., & Zhou, M. (2024). PyRadar: Towards automatically retrieving and validating source code repository information for PyPI packages. Proceedings of the ACM on Software Engineering, 1(FSE), 2608-2631. https://doi.org/10.1145/3660822

[14] Esposito, M., & Falessi, D. (2024). VALIDATE: A deep dive into vulnerability prediction datasets. Information and Software Technology, 170, 107448. https://doi.org/10.1016/j.infsof.2024.107448

[15] Henriques, J. R., Pereira, J. D., & Vieira, M. (2024). Mining vulnerability and code repositories to study software security. Proceedings of the 13th Latin-American Symposium on Dependable and Secure Computing, 11-16. https://doi.org/10.1145/3697090.3697103

[16] Rebatchi, H., Moha, N., & Bissyandé, T. F. (2024). Dependabot and security pull requests: Large empirical study. Empirical Software Engineering, 29(5), 119. https://doi.org/10.1007/s10664-024-10523-y

[17] Guan, F., Zhu, T., Zhou, W., & Choo, K. K. R. (2024). Graph neural networks: A survey on the links between privacy and security. Artificial Intelligence Review, 57, 40. https://doi.org/10.1007/s10462-023-10656-4

[18] Moyle, S., Martin, A., & Allott, N. (2024). XAI human-machine collaboration applied to network security. Frontiers in Computer Science, 6, 1321238. https://doi.org/10.3389/fcomp.2024.1321238

[19] Mo, K., Ye, P., Ren, X., Wang, S., Li, W., & Li, J. (2024). Security and privacy issues in deep reinforcement learning: Threats and countermeasures. ACM Computing Surveys, 56(6), Article 152. https://doi.org/10.1145/3640312

[20] Lin, M., Xia, Y., Sun, Z., & Dai, L. (2024). Learning-based model predictive control under value iteration with finite approximation errors. International Journal of Robust and Nonlinear Control, 34(4), 2946-2971. https://doi.org/10.1002/rnc.7117

[21] Wachi, A., Shen, X., & Sui, Y. (2024). A survey of constraint formulations in safe reinforcement learning. Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, 8262-8271. https://doi.org/10.24963/ijcai.2024/913

[22] Schueller, W., & Wachs, J. (2024). Modeling interconnected social and technical risks in open-source software ecosystems. Collective Intelligence, 3(1). https://doi.org/10.1177/26339137241231912

[23] Huang, Y., Wang, R., Zheng, W., Zhou, Z., Wu, S., Ke, S., Chen, B., Gao, S., & Peng, X. (2024). SpiderScan: Practical detection of malicious NPM packages based on graph-based behavior modeling and matching. Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, 1146-1158. https://doi.org/10.1145/3691620.3695492

[24] Froh, F. N., Gobbi, M. F., & Kinder, J. (2023). Differential static analysis for detecting malicious updates to open-source packages. Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses, 41-49. https://doi.org/10.1145/3605770.3625211

[25] Mitta, R., Hasanbeig, H., Wang, J., Kroening, D., Kantaros, Y., & Abate, A. (2024). Safeguarded progress in reinforcement learning: Safe Bayesian exploration for control policy synthesis. Proceedings of the AAAI Conference on Artificial Intelligence, 38(19), 21412-21419. https://doi.org/10.1609/aaai.v38i19.30137

[26] Ladisa, P., Plate, H., Martinez, M., & Barais, O. (2023). On the feasibility of cross-language detection of malicious packages in npm and PyPI. Proceedings of the 39th Annual Computer Security Applications Conference, 71-82. https://doi.org/10.1145/3627106.3627138

[27] Baumann, D., & Schön, T. B. (2024). Safe reinforcement learning in uncertain contexts. IEEE Transactions on Robotics, 40, 1828-1841. https://doi.org/10.1109/TRO.2024.3354176

[28] Ahmed, I., Jeon, G., & Piccialli, F. (2024). Explainable AI for cybersecurity automation, intelligence and trustworthiness in digital twin: Methods, taxonomy, challenges and prospects. ICT Express, 10(4), 935-958. https://doi.org/10.1016/j.icte.2024.05.007

[29] Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Vats, I., Moazen, H., & Sarro, F. (2024). A survey on machine learning techniques applied to source code. Journal of Systems and Software, 209, 111934. https://doi.org/10.1016/j.jss.2023.111934

[30] Mechri, A., Ferrag, M. A., & Debbah, M. (2025). SecureQwen: Leveraging LLMs for vulnerability detection in Python codebases. Computers & Security, 148, 104151. https://doi.org/10.1016/j.cose.2024.104151

[31] Xiao, W., Hou, Z., Wang, T., Zhou, C., & Pan, C. (2024). MSGVUL: Multi-semantic integration vulnerability detection based on relational graph convolutional neural networks. Information and Software Technology, 170, 107442. https://doi.org/10.1016/j.infsof.2024.107442

[32] Wan, Y., Bi, Z., He, Y., Zhang, J., Zhang, H., Sui, Y., Xu, G., Jin, H., & Yu, P. (2024). Deep learning for code intelligence: Survey, benchmark and toolkit. ACM Computing Surveys, 56(12), Article 309. https://doi.org/10.1145/3664597

[33] Fruntke, L., & Krinke, J. (2025). Automatically fixing dependency breaking changes. Proceedings of the ACM on Software Engineering, 2(FSE), Article FSE096. https://doi.org/10.1145/3729366

[34] Nahum, M., Grolman, E., Maimon, I., Mimran, D., Brodt, O., Elyashar, A., Elovici, Y., & Shabtai, A. (2024). OSSIntegrity: Collaborative open-sour aforementionedce code integrity verification. Computers & Security, 144, 103977. https://doi.org/10.1016/j.cose.2024.103977

[35] Zhou, X., Zhang, T., & Lo, D. (2024). Large language model for vulnerability detection: Emerging results and future directions. Proceedings of the 2024 ACM/IEEE 46th International Conference on Software Engineering: New Ideas and Emerging Results, 47-51. https://doi.org/10.1145/3639476.3639762