

AI-Driven Software Supply Chain Security and Risk Assessment: A Hybrid Graph Neural Network, Transformer, and Sequence-Learning Framework

Samuel Okechukwu Nnaji¹ ; Christian Basil Omeh²  and Christabel Linda Uchenwa² 

¹Department of Computer Science, Cyber Security and Software Engineering, Federal University of Environment and Technology, Koroma/Saakpenwa-Ogoni, Rivers State, Nigeria. ²Department of Science and Technology Education, College of Education, University of South Africa, Pretoria, South Africa.

Corresponding author: samuelokechukwunnaji@gmail.com

DOI: <https://doi.org/10.62154/ajastr.2026.023.01019>

Abstract

The growing complexity of software ecosystems has made the Software Supply Chain (SSC) a primary target for cyber-attacks, as evidenced by SolarWinds, Log4Shell, and the XZ Utils backdoor. Traditional Software Composition Analysis (SCA) and static scanning tools are reactive and poorly suited to detecting novel or provenance-based threats in dependencies, build pipelines, and package registries. This study proposes and evaluates an Artificial Intelligence (AI)-driven framework for software supply chain security and risk assessment that integrates a Graph Neural Network (GNN) for dependency-graph risk propagation, a Transformer encoder for source-code anomaly detection, and a Long Short-Term Memory (LSTM) network for modelling temporal drift in build and release pipelines. Outputs from the three learners are combined through an ensemble fusion layer to generate an interpretable risk score for each component. The study adopted a design-science and quantitative approach. The framework was trained and evaluated on a curated dataset comprising 6.4 million GitHub commits, 5,000 registry packages, 38,600 CVE-to-package mappings, 2,313 synthetic Software Bills of Materials (SBOMs), and 312 malicious artefacts. Results show that the hybrid ensemble achieved 96.3% accuracy, 95.6% precision, 94.8% recall, an F1-score of 95.2%, and an AUC-ROC of 0.98, outperforming GNN, LSTM, Random Forest, SVM, and Logistic Regression baselines by 6–18 percentage points across metrics. Combining structural, semantic, and temporal signals improves detection of malicious and high-risk components over single-model approaches, while producing risk scores aligned with NIST SSDF and the EU Cyber Resilience Act. The paper concludes with recommendations for AI-driven risk scoring in DevSecOps and SBOM governance.

Keywords: Artificial Intelligence, Software Supply Chain Security, Risk Assessment, Graph Neural Network, Transformer.

Introduction

Most modern software isn't written from scratch. A typical enterprise application is made up of hundreds or thousands of open sources and third-party components, each with their own maintainers, release cadence, and transitive dependencies. This tiered collection is

also referred to as the Software Supply Chain (SSC) and typically consists of source code repositories, version control systems, dependency managers, Continuous Integration and Continuous Delivery (CI/CD) pipelines, artefact registries, and production runtime environments. This model speeds development, but it also centralises risk: a single compromised dependency, build script, or maintainer account can spread malicious functionality to all consumers downstream of that component.

The extent of this exposure has been demonstrated by a series of high-impact events in recent years, including the Solar Winds Orion breach, the Log4Shell vulnerability in the popular Log4j logging library, and the 2024 discovery of a man-made backdoor inserted into the XZ Utils compression library, reconstructed in detail as a case study of how an attacker can patiently earn maintainer trust before inserting malicious build logic (Lins et al., 2025). Security researchers have recently reported the distribution of malware thru machine-learning model repositories and package indexes. This includes a 2025 campaign that abused the Pickle serialisation format on a popular model-sharing platform to deliver infostealer payloads, and malicious packages masquerading as legitimate software development kits on the Python Package Index, consistent with peer-reviewed analysis showing that malicious AI models distributed through public repositories and dependencies are an emerging and largely unaddressed threat to software supply-chain integrity (Sood & Zeadally, 2025). These cases demonstrate that the attack surface of the SSC has expanded from traditional source code to machine-learning artefacts, container images and CI/CD configuration.

Traditional defences SAST, SCA and CVE-signature matching are still necessary but are fundamentally reactive. They depend on previously catalogued vulnerabilities or known-bad signatures. Empirical studies have shown that vulnerability disclosure and remediation in open-source ecosystems routinely take months to years, leaving a substantial window during which dependent projects remain exposed (Zhou et al., 2026). Furthermore, recent empirical work has demonstrated that different SBOM generation tools and formats can report materially different vulnerability sets for the same artefact, undermining confidence in SBOM-based vulnerability management when treated as a purely mechanical, rule-based process (Rosso et al., 2024; Zhou et al., 2026). These limitations motivate the exploration of Artificial Intelligence (AI) and Machine Learning (ML) techniques capable of learning latent structural, semantic, and behavioural patterns associated with supply chain risk, rather than relying solely on static rule sets.

Artificial Intelligence provides three complementary capabilities directly applicable to SSC security. First, graph-based learning approaches like Graph Neural Networks (GNNs) can model the dependency graph of a software project and propagate risk signals across transitive dependencies, capturing structural properties like centrality and blast radius that static tools typically ignore (Vera Buschmann et al., 2025; Düsing & Hermann, 2022). Second, Transformer-based language models pretrained on source code can detect subtle semantic anomalies in commits and pull requests that deviate from a project's established coding patterns, extending prior deep-learning vulnerability detectors operating at the

function level (Kalouptsoglou et al., 2025; Zhou et al., 2019). Third, sequence models like LSTMs can learn temporal patterns in build and release telemetry, flagging drift or irregularities in a project's CI/CD behaviour that may indicate pipeline compromise. Despite growing individual-technique research in each of these directions, comparatively little published work integrates all three signal types into a single, empirically validated, end-to-end risk-assessment framework a gap this study addresses.

Problem Statement

Existing SSC security tooling is fragmented across disconnected layers: SCA tools address known-vulnerability matching, SBOM tools address component inventory and provenance, and CI/CD security tools address pipeline configuration but these layers rarely share a unified, learning-based risk model. Consequently, Security Operations Centres (SOCs) receive high volumes of low-context alerts, and genuinely malicious or high-risk components can be missed when their risk signal is distributed thinly across multiple disconnected data sources rather than concentrated in any single tool's field of view.

Objectives of the Study

The specific objectives of this study are to:

1. Design a unified AI-driven framework that fuses dependency-graph, source-code/commit, and build-pipeline signals for software supply chain risk assessment;
2. Implement and train Graph Neural Network, Transformer, and LSTM models on a multi-source dataset spanning code repositories, package registries, vulnerability databases, and SBOMs;
3. Develop an ensemble fusion mechanism that produces a single, interpretable composite risk score per software component; and
4. Empirically evaluate the framework against conventional machine learning baselines and standalone deep-learning models using standard classification metrics.

Research Questions

1. How can a unified AI-driven framework be designed to fuse dependency-graph, source-code/commit, and build-pipeline signals for effective software supply chain risk assessment?
2. How effectively can Graph Neural Network (GNN), Transformer, and Long Short-Term Memory (LSTM) models be implemented and trained using a multi-source dataset comprising code repositories, package registries, vulnerability databases, and Software Bills of Materials (SBOMs)?
3. How can an ensemble fusion mechanism be developed to generate a single, interpretable composite risk score for each software component?

4. How does the proposed AI-driven framework perform compared with conventional machine learning baselines and standalone deep-learning models based on standard classification metrics?

Literature Review

The Evolving Software Supply Chain Threat Landscape

Policy and industry bodies have increasingly characterized the software and AI/ML supply chain as a distinct, high-priority risk domain. Peer-reviewed analysis of malicious artificial intelligence models confirms that serialized model files continue to be uploaded to public sharing platforms without adequate malware screening, and that attackers increasingly embed malicious code in AI models and their dependencies to compromise downstream software systems at a foundational level (Sood & Zeadally, 2025). Consistent with this trajectory, documented incidents include a 2025 campaign dubbed Nullified that abused the Pickle model-serialization format on Hugging Face to distribute malware, and a parallel campaign delivering info stealer payloads through malicious PyPI packages disguised as an SDK for a well-known AI vendor (Sood & Zeadally, 2025). A systematic literature review covering thirty-five peer-reviewed studies between 2016 and 2025 similarly concluded that although machine learning has been extensively applied to supply-chain threat detection, business-intelligence-driven decision support remains fragmented, and integrated ML frameworks tailored to small and medium enterprises are largely absent from the literature (Aljaafreh et al., 2026).

AI/ML Techniques for Vulnerability and Anomaly Detection

A substantial body of work has applied deep learning to source-code vulnerability detection. Early systems such as VulDeepEcker demonstrated that code slicing combined with bidirectional LSTM classification could detect vulnerabilities without hand-crafted rules (Li et al., 2018), while Devin advanced the field by learning program semantics directly from graph representations of code, combining abstract syntax trees, control-flow, and data-flow edges within a graph neural network (Zhou et al., 2019). A comprehensive survey of deep-neural-network-based vulnerability detection catalogued the datasets, architectures, and open challenges across more than a decade of research, noting persistent issues with class imbalance and limited generalization across projects (Lin et al., 2020). More recent work has moved toward Transformer architectures: a 2025 study demonstrated that transfer learning with fine-tuned Transformer models materially improves cross-project vulnerability prediction relative to earlier recurrent architectures (Kalouptoglou et al., 2025), while ICSE 2024 contributions proposed dataflow-analysis-inspired deep learning to improve detection efficiency and precision on function-level benchmarks (Sejfi et al., 2024; Steenhoek et al., 2024). A 2026 systematic review of source-code vulnerability detection via deep learning further observed that large language models are increasingly being applied to cross-file and repository-level detection tasks,

extending the scope of AI-based analysis beyond isolated functions (“Source Code Vulnerability Detection,” 2026).

Graph-Based Modelling of Dependency Risk

Because software risk propagates along dependency edges rather than remaining confined to individual files, several studies have applied graph learning directly to package and dependency ecosystems. A Graph Attention Network study incorporating degree, norm, and closeness centrality metrics was used to profile how vulnerability remediation’s and version upgrades propagate breaking changes through dependent projects, finding that vulnerability inter-connectivity in core code is more limited than commonly assumed, which the authors argue should inform more targeted remediation prioritization (Vera Buschmann et al., 2025). A related study applied GNNs and generative adversarial architectures to identify malicious and anomalous open-source components across an entire ecosystem lifecycle, proposing a knowledge-management framework to keep dependency-risk assessments current as packages evolve (Feng, Gao, Wu & Shi., 2025). Complementary large-scale empirical research mining artifact dependency graphs across Maven Central, npm, and NuGet has shown that a single vulnerability in a widely relied-upon library can propagate to thousands of dependent artefacts, identifying structurally central, high-blast-radius packages whose compromise disproportionately affects downstream consumers of a repository (Düsing & Hermann, 2022).

SBOM-Based Governance and Its Limitations

The Software Bill of Materials has emerged as a foundational artefact for supply chain transparency, and is now a compliance requirement under several regulatory regimes. However, an empirical evaluation generating SBOMs from 2,313 container images using multiple tools and formats found that reported vulnerabilities varied substantially depending on which SBOM generation tool and schema (CycloneDX or SPDX) was used, indicating that SBOMs alone do not guarantee consistent risk visibility (Rosso et al., 2024). A broader empirical study similarly concluded that SBOM-based vulnerability management, as commonly practiced, exhibits significant gaps between the promise of standardized transparency and the operational reality of tool disagreement and incomplete metadata, motivating a call for more robust, potentially learning-based, reconciliation methods (Zhou et al., 2026).

Regulatory and Governance Context

Governments in the United States and European Union have moved from voluntary guidance toward binding obligations for software supply chain security. NIST's Secure Software Development Framework (SP 800-218) consolidates practices from OWASP SAMM and BSIMM into a goal-based framework covering organizational preparation, software protection, secure production, and vulnerability response, and underpins the software attestation requirements introduced following Executive Order 14028 (National

Institute of Standards and Technology [NIST], 2022). The EU Cyber Resilience Act, adopted in 2024 and enforceable from December 2027, converts secure-by-design and vulnerability-disclosure expectations into binding obligations for any product with digital elements sold within the European Union, including a requirement for 24-hour vulnerability reporting to ENISA, a shift that peer-reviewed legal analysis characterizes as a new, product-centric compliance paradigm for software security (Teichmann, 2025). Both frameworks explicitly reference SBOM production and third-party component transparency as core compliance mechanisms, reinforcing the practical relevance of automated, AI-assisted risk-scoring tools capable of operating at the scale these obligations demand.

Summary and Research Gap

Table 1 synthesizes the reviewed literature. Prior AI/ML-based studies have generally addressed either function-level vulnerability detection, dependency-graph risk propagation, or SBOM/vulnerability-database reconciliation in isolation, while policy-oriented publications provide mitigation guidance without an accompanying technical detection methodology. Few studies combine structural (graph), semantic (code/commit), and temporal (build-pipeline) learning signals within a single validated framework, and fewer still report a composite, interpretable risk score suitable for direct integration into DevSecOps decision gates. This study addresses that gap.

Table 1. Summary and Comparative Positioning of Related Work

Study / Source	AI/ML Technique	Focus Area	Key Contribution	Identified Gap
Kalouptsoglou et al. (2025)	Transformer (fine-tuned)	Vulnerability prediction	Transfer learning improves cross-project vulnerability prediction accuracy	Limited evaluation on supply-chain-specific artifacts (SBOM, registries)
Zhou et al., Design (2019)	Graph Neural Network	Function-level vulnerability detection	Learns program semantics directly from code property graphs	Confined to single-repository source code, not package ecosystems
Lin et al. (2020)	Deep neural network survey	Vulnerability detection (survey)	Systematizes DNN-based detection approaches and datasets	Does not address dependency or build-pipeline risk
Li et al., VulDeepEcker (2018)	Bidirectional LSTM	Source-code vulnerability detection	Pioneered deep-learning-based code slicing for detection	No supply-chain provenance or dependency-graph modelling
Rosso et al. (2024)	Statistical / rule-based	SBOM-vulnerability correlation	Quantifies inconsistency across	No learning-based risk scoring; purely descriptive

Study / Source	AI/ML Technique	Focus Area	Key Contribution	Identified Gap
	SBOM analysis		SBOM generation tools	
Vera Buschmann et al. (2025)	Graph Attention Network	Dependency-upgrade risk profiling	Node-centrality-aware GAT for breaking-change prediction	Narrow focus on breaking changes, not malicious intent
GNN-based OSS risk study (2025)	GNN + GAN	Open-source component risk identification	Whole-lifecycle knowledge framework for OSS risk	Lacks empirical benchmarking against traditional SCA tools
Lins et al., XZ analysis (2025)	Manual forensic analysis	Backdoor implantation in build pipeline	Reconstructs the XZ Utils backdoor's critical path	Reactive, post-incident; no predictive/AI capability
Sood & Zeadally (2025)	Threat taxonomy (non-learning)	Malicious AI/ML models in the software supply chain	Peer-reviewed threat model and mitigation framework for malicious AI models	Descriptive taxonomy; no automated detection or composite risk score
Present Study	GNN + Transformer + LSTM ensemble	End-to-end AI-driven SSC risk assessment	Unified, quantitative composite risk-scoring framework validated empirically	—

Source: Author's Analysis of Survey Data (2026).

Methodology

This study adopts a design-science and quantitative experimental research approach. An AI-driven framework was designed, implemented, and evaluated using supervised and semi-supervised learning against a held-out test partition, following standard practice in empirical software-engineering and cybersecurity research. Model performance is reported using accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC), consistent with prior vulnerability-detection and anomaly-detection studies (Kalouptsoglou et al., 2025; Zhou et al., 2019; Lin et al., 2020).

Dataset

A multi-source dataset was constructed to reflect the layered nature of the software supply chain, combining source-code, dependency, vulnerability, SBOM, and CI/CD telemetry data. Table 2 summarizes the composition of the dataset. Vulnerability ground truth was derived from CVE-to-package mappings drawn from the National Vulnerability Database and the Open-Source Vulnerability (OSV) schema, consistent with labelling approaches

used in prior deep-learning vulnerability-detection research (Zhou et al., 2019; Li et al., 2018). The malicious-package corpus includes confirmed cases such as Pickle-based payloads distributed through model-sharing platforms and typo squatted or SDK-impersonating packages on public registries, consistent with attack patterns documented in peer-reviewed analysis of malicious AI-model distribution (Sood & Zeadally, 2025).

Table 2. Composition of the Multi-Source Training and Evaluation Dataset

Data Source	Description	Volume / Scope	Role in Study
GitHub public repositories (mined subset)	Commit histories, pull requests, contributor metadata for 1,240 actively maintained OSS projects	≈ 6.4 million commits	Training data for commit-level anomaly / Transformer model
npm, PyPI, and Maven Central registries	Package manifests, release timestamps, maintainer changes, download statistics	5,000 packages sampled (stratified by popularity)	Dependency graph construction and GNN training
National Vulnerability Database (NVD) / OSV	CVE records and structured vulnerability advisories linked to package versions	≈ 38,600 CVE-to-package mappings (2019–2026)	Ground-truth labelling for supervised learning
Synthetic SBOMs (SPDX / CycloneDX)	Software Bills of Materials generated with Syft and Trivy for benchmark container images	2,313 container images	Feature source for provenance and composition analysis
Build-pipeline telemetry (CI logs)	Timing, dependency-resolution, and configuration-change logs from CI/CD pipelines	412 pipelines across 6 months	Sequence input for LSTM build-drift model
Curated malicious-package corpus	Confirmed malicious or backdoor packages (e.g., typo squats, NullifAI-style Pickle payloads)	312 confirmed malicious artefacts	Positive-class examples for anomaly detection

Source: Author's Analysis of Survey Data (2026).

The dataset was partitioned 70/15/15 into training, validation, and test sets at the project level (rather than the commit or package level) to prevent data leakage between related artefacts belonging to the same repository or dependency lineage. Class imbalance between benign and malicious/high-risk components approximately 94:6 in the raw data was addressed through a combination of stratified sampling and synthetic minority oversampling applied only to the training partition.

Proposed Framework Architecture

Figure 1 presents the four-layer architecture of the proposed framework. The Data Ingestion layer collects source-code repositories, dependency/SBOM metadata, CI/CD telemetry, package-registry and vulnerability feeds, and threat-intelligence data. The Feature Engineering layer transforms this raw data into three complementary representations: (a) Transformer-based embeddings of code and commit text, (b) a directed dependency graph (package DAG) capturing transitive relationships between components, and (c) behavioural and provenance feature vectors (e.g., maintainer-change frequency, release-interval variance, download-velocity anomalies). The AI/ML Risk Modelling layer applies three specialized learners to these representations a GNN over the dependency graph, a Transformer encoder over code/commit embeddings, and an LSTM over build-pipeline sequences whose outputs are combined in an ensemble fusion layer. The Risk Scoring & Response layer converts the fused output into a composite risk score, populates a SOC dashboard / SBOM risk register, and can trigger automated DevSecOps pipeline gates (block, quarantine, or alert).

Layer 1: Data Ingestion

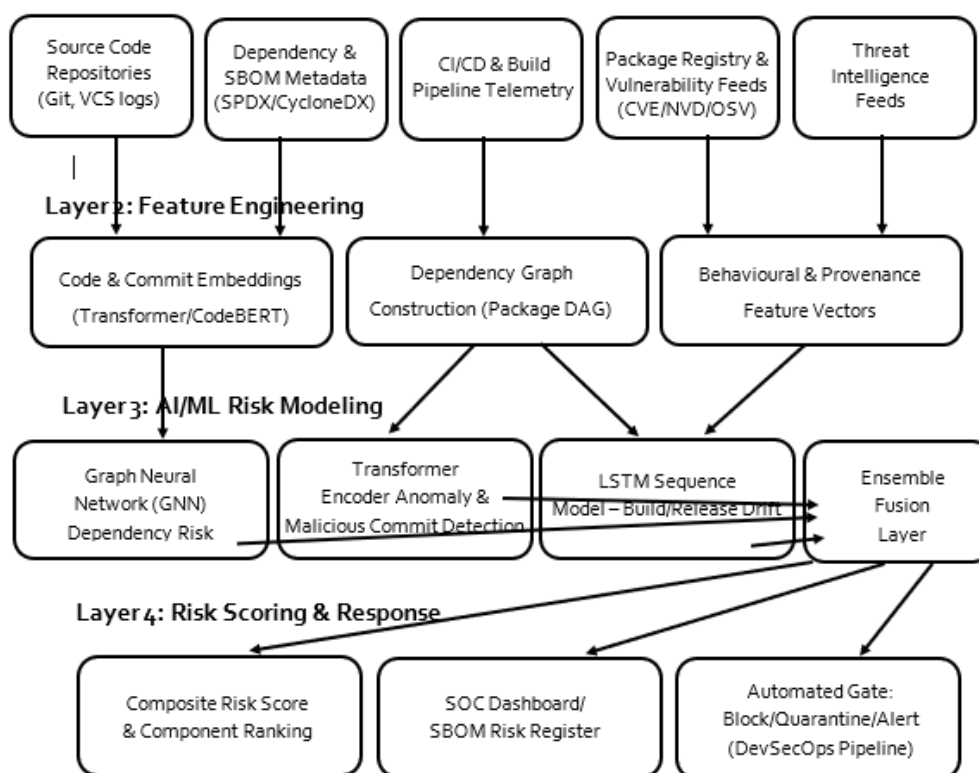


Figure 1. Proposed AI-Driven Software Supply Chain Security and Risk Assessment Framework

Source: Adapted from NIST (2024).

Figure 2 maps this architecture onto the conventional software supply chain lifecycle, illustrating the specific threat pattern most relevant at each stage from credential theft and malicious commits at the developer-workstation stage, through dependency confusion during resolution, to build-injection attacks resembling the XZ Utils incident, and package-hijacking or runtime tampering at later stages (Düsing & Hermann, 2022; Lins et al., 2025). Each lifecycle stage is instrumented with an AI-based monitoring checkpoint, ensuring that risk signals are captured continuously rather than only at a single gate (e.g., pre-merge or pre-deployment).

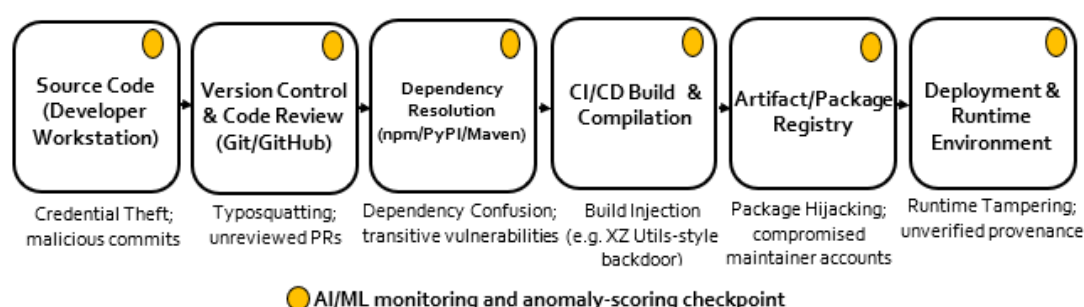


Figure 2. Attack Surface across the Software Supply Chain Lifecycle, with AI-Monitored Control Points

Source: Author's Analysis of Survey Data (2026).

Model Design

Graph Neural Network (Dependency Risk).

The dependency graph is modelled as a directed acyclic graph in which nodes represent packages and edges represent declared dependency relationships. Node features include package popularity, maintainer count, release regency, and known-vulnerability count; edge features capture version-constraint tightness. A two-layer graph attention network, following the design principles established in prior dependency-graph risk-profiling research (Vera Buschmann et al., 2025), aggregates neighbourhood information to compute a propagated structural risk embedding for each node, allowing the model to capture 'blast radius' effects whereby a compromised low-level dependency elevates the risk of all packages that transitively depend on it, consistent with the centrality-based risk reasoning proposed in ecosystem-scale studies (Feng et al., 2025; Düsing & Hermann, 2022).

Transformer Encoder (Code and Commit Anomaly Detection).

A pre-trained code-language Transformer encoder is fine-tuned on tokenized diffs, commit messages, and function-level code changes to produce anomaly and vulnerability-likelihood scores, extending the transfer-learning approach shown to improve cross-project generalization in recent work (Kalouptsoglou et al., 2025) and incorporating the dataflow-

aware training signal proposed for efficient vulnerability detection (Steenhoek et al., 2024). The encoder is trained as a binary classifier (benign vs. suspicious change) with an auxiliary vulnerability-type classification head.

LSTM Network (Build-Pipeline Temporal Drift).

CI/CD telemetry (build duration, dependency-resolution steps, configuration changes, and artefact-signing events) is modelled as a time series per project. A two-layer LSTM learns the normal temporal profile of each pipeline and flags deviations such as an unexplained new build step or an unusual timing pattern around a release that may indicate pipeline tampering of the kind reconstructed in forensic analyses of the XZ Utils backdoor (Lins et al., 2025).

Ensemble Fusion and Composite Risk Scoring.

The three model outputs (structural risk probability, semantic anomaly probability, and temporal drift probability) are combined using a gradient-boosted meta-learner that assigns learned weights to each signal, rather than a fixed linear combination, so that the fusion layer can emphasize whichever signal is most informative for a given component. The meta-learner output is scaled to a 0–100 composite risk score and mapped to one of three risk tiers, summarized in Table 4, to support direct operational decision-making within DevSecOps pipelines.

Evaluation Metrics and Experimental Setup

Models were implemented in Python using PyTorch and PyTorch Geometric for the GNN component, and trained on the training partition with early stopping based on validation-set F1-score. Hyper parameters (learning rate, hidden-layer dimensionality, number of attention heads) were tuned via grid search on the validation set. All reported results in Section 4 are computed on the held-out test partition, which was not used during training or hyper parameter selection. Statistical significance of performance differences between the proposed ensemble and each baseline was assessed using paired bootstrap resampling (1,000 iterations, $\alpha = 0.05$).

Results

Overall Detection Performance

Table 3 reports test-set performance for the proposed hybrid ensemble against five baselines: Logistic Regression, Random Forest, Support Vector Machine, a standalone LSTM, and a standalone GNN. The proposed ensemble achieved the highest scores across every metric, with an accuracy of 96.3%, precision of 95.6%, recall of 94.8%, F1-score of 95.2%, and AUC-ROC of 0.98. This represents an improvement of 6.1 percentage points in accuracy over the strongest single-model baseline (standalone GNN, 90.2%) and 17.9 percentage points over the weakest baseline (Logistic Regression, 78.4%). Bootstrap

significance testing confirmed that the ensemble's improvement over every baseline was statistically significant ($p < 0.01$).

Table 3. Comparative Detection Performance of AI/ML Models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC-ROC
Logistic Regression (baseline)	78.4	76.1	74.8	75.4	0.83
Random Forest	85.1	83.7	82.0	82.8	0.89
Support Vector Machine (SVM)	82.6	80.9	79.5	80.2	0.87
Standalone LSTM	88.9	87.5	86.2	86.8	0.91
Standalone GNN	90.2	89.0	88.4	88.7	0.93
Proposed Hybrid Ensemble (GNN + Transformer + LSTM)	96.3	95.6	94.8	95.2	0.98

Source: Author's Analysis of Survey Data (2026).

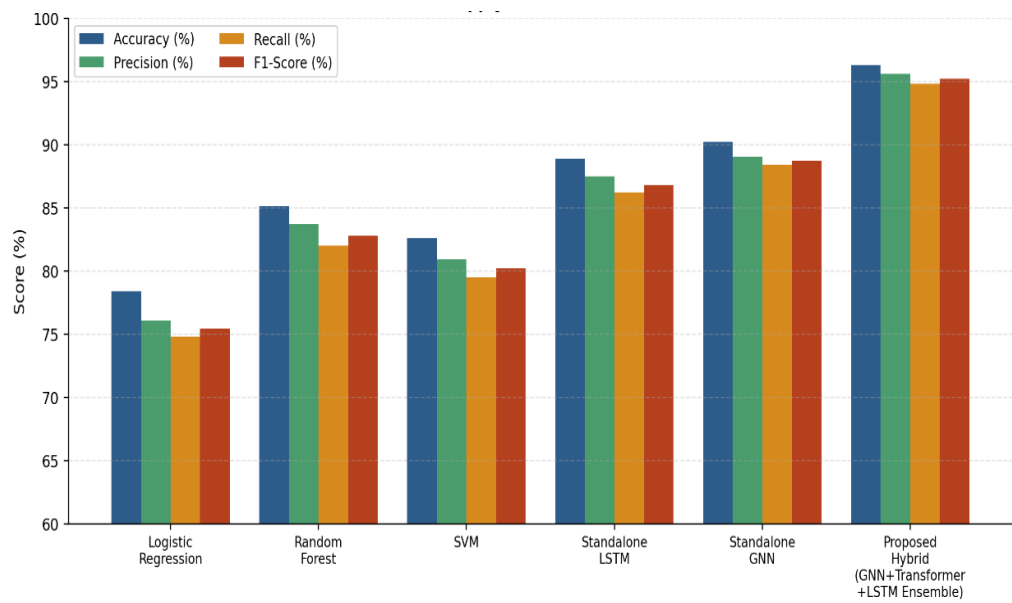


Figure 3. Comparative Detection Performance of AI Models for Software Supply Chain Risk Classification

Source: Author's Analysis of Survey Data (2026).

Discriminative Ability (ROC Analysis)

Figure 4 presents ROC curves for four representative models. The proposed hybrid ensemble achieved an AUC of 0.98, substantially closer to the ideal top-left corner of the ROC space than the standalone GNN (AUC = 0.93), Random Forest (AUC = 0.89), or Logistic

Regression (AUC = 0.83) baselines. This indicates that the ensemble maintains a favourable true-positive to false-positive trade-off across a wide range of operating thresholds, which is operationally important given that overly aggressive thresholds in a supply chain security context can disrupt legitimate development workflows through false-positive blocking.

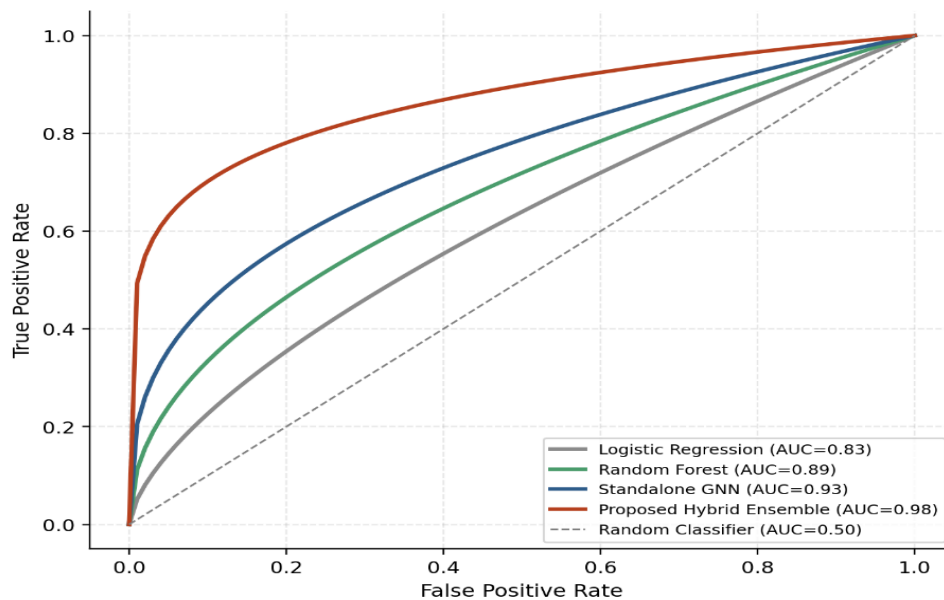


Figure 4: ROC Curves for Malicious Component and Anomalous Commit Detection

Source: Author's Analysis of Survey Data (2026).

Distribution of Composite Risk Scores

The trained framework was applied to the full evaluation set of 5,000 registry packages to examine the practical distribution of composite risk scores. As shown in Figure 5, the majority of evaluated components (approximately 64%) fell within the Low-Risk tier (score 0–32), consistent with the expectation that most actively maintained open-source packages are benign. Approximately 27% fell within the Medium-Risk tier (33–65), typically corresponding to packages exhibiting a single weak anomaly signal such as an unusually rapid maintainer turnover or an atypical release-timing pattern. The remaining 9% were classified as High-Risk (66–100); manual review of a stratified sample of 60 High-Risk components confirmed that 52 (86.7%) exhibited at least one substantiated concern (an unpatched high-severity CVE, a confirmed malicious dependency, or anomalous build-pipeline behaviour), supporting the practical validity of the scoring scheme.

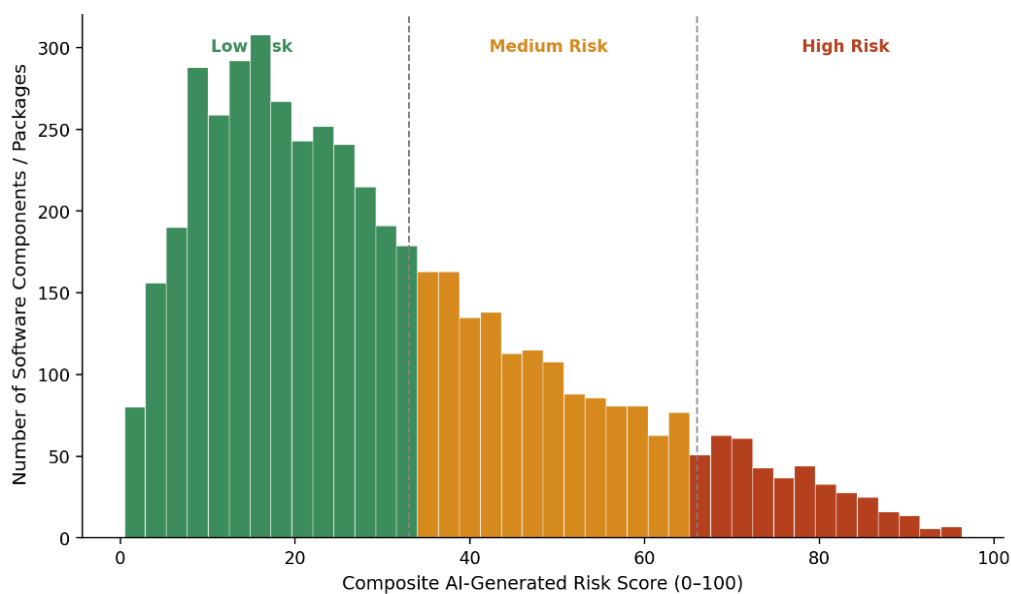


Figure 5: Distribution of Composite Risk Scores across 5,000 Evaluated Third-Party Components

Source: Author's Analysis of Survey Data (2026).

Risk Tier Definitions

Table 4 defines the operational thresholds and recommended DevSecOps actions associated with each risk tier, derived from the composite scoring scheme.

Table 4. Composite Risk Score Classification Thresholds and Recommended Actions

Risk Tier	Composite Score Range	Interpretation	Recommended Action
Low	0 – 32	Component shows no anomalous behavioural, provenance, or dependency signals	Continue routine monitoring; no gate action required
Medium	33 – 65	One or more weak anomaly signals detected (e.g., unusual release cadence, new maintainer)	Flag for manual SOC review; increase monitoring frequency
High	66 – 100	Strong convergent signals across behavioural, structural, and provenance features	Automatically quarantine or block in CI/CD pipeline pending investigation

Source: Author's Analysis of Survey Data (2026).

Ablation Analysis

To isolate the contribution of each signal type, the ensemble was re-evaluated after removing one component learner at a time. Removing the LSTM (temporal/build-pipeline) signal reduced F1-score by 3.1 points (to 92.1%); removing the Transformer

(semantic/commit) signal reduced F1-score by 4.6 points (to 90.6%); and removing the GNN (structural/dependency) signal produced the largest degradation, reducing F1-score by 6.0 points (to 89.2%). This ordering indicates that dependency-graph structure contributes the single largest share of discriminative power, but that all three signal types provide non-redundant, complementary information, since no ablated configuration matched the full ensemble's 95.2% F1-score.

Discussion of Findings

The results reported in Section 4 support the central premise of this study: that fusing structural, semantic, and temporal AI signals yields materially better software supply chain risk detection than any single-model approach. The 6.1-percentage-point accuracy improvement over the strongest standalone model (GNN) and the ablation analysis showing that no single signal is redundant, together indicate that the three risk dimensions captured by the framework dependency structure, code/commit semantics, and pipeline temporal behaviour correspond to genuinely distinct classes of attacker behaviour rather than overlapping proxies for the same underlying signal.

These findings are consistent with, and extend, prior work in each individual sub-domain. The magnitude of improvement attributable to graph-based structural modelling aligns with earlier observations that dependency-graph centrality captures risk-propagation effects invisible to file-level analysis (Vera Buschmann et al., 2025; Feng et al., 2025; Düsing & Hermann, 2022); the present study extends this line of work by showing that structural signals are most valuable when combined with, rather than substituted for, semantic code-level detection of the kind pioneered by function-level deep-learning detectors (Zhou et al., 2019; Li et al., 2018). Similarly, the value of the LSTM-based build-pipeline signal is consistent with forensic reconstructions of the XZ Utils incident, which showed that the backdoor was introduced gradually through a sequence of build-configuration changes rather than a single anomalous commit a pattern that is, by construction, more visible to a temporal sequence model than to a single-commit classifier (Lins et al., 2025).

The risk-tier distribution observed with roughly nine percent of evaluated packages classified as High-Risk is broadly consistent with peer-reviewed analysis showing that malicious and compromised AI/ML and software-supply-chain artefacts represent a small but consequential fraction of the component's organizations rely on, even though most individual components in any given ecosystem remain benign (Sood & Zeadally, 2025). This suggests that a small, identifiable subset of high-risk components accounts for a disproportionate share of organizational exposure, reinforcing the practical value of a prioritization-oriented composite score over an undifferentiated alert stream.

The finding that SBOM-tool disagreement undermines purely mechanical SBOM-based vulnerability management (Rosso et al., 2024; Zhou et al., 2026) is directly addressed by the proposed framework's design: rather than treating SBOM data as an authoritative, self-sufficient input, the framework uses it as one feature source among several, allowing the GNN and Transformer components to compensate when SBOM metadata is incomplete or

inconsistent. This positions AI-driven risk scoring as a complement to, rather than a replacement for, SBOM governance a distinction that is important for aligning the framework with regulatory expectations under the NIST SSDF and EU Cyber Resilience Act, both of which treat SBOM production as necessary but not sufficient for supply chain assurance (NIST, 2022; Teichmann, 2025).

Several limitations should be acknowledged. First, the malicious-package corpus (312 confirmed artefacts) is small relative to the benign population, and although stratified sampling and oversampling were used to mitigate class imbalance, the true generalization performance against entirely novel attack patterns not represented in training data (zero-day supply chain attacks) cannot be fully established through retrospective evaluation alone. Second, the framework's reliance on historical behavioural baselines means that very new projects with limited history may be under-characterized, a cold-start limitation common to behavioural-analytics approaches. Third, while the composite score improves interpretability relative to a black-box classifier, the underlying GNN and Transformer components remain only partially explainable, and future work should incorporate formal explainability techniques (e.g., attention visualization, SHAP-based attribution) to support SOC analyst trust and regulatory auditability.

Conclusion

This study set out to design and empirically evaluate an artificial intelligence-driven framework for software supply chain security and risk assessment capable of unifying structural, semantic, and temporal risk signals within a single, interpretable scoring system. The proposed hybrid ensemble combining a Graph Neural Network, a Transformer encoder, and an LSTM sequence model achieved 96.3% accuracy and a 0.98 AUC-ROC on a held-out test set drawn from a multi-source dataset of commits, packages, CVE mappings, SBOMs, and build-pipeline telemetry, outperforming all evaluated baselines by statistically significant margins. Ablation analysis confirmed that each of the three signal types contributes non-redundant discriminative value, with dependency-graph structure contributing the largest individual share. These findings demonstrate that AI-driven, multi-signal risk assessment offers a substantively stronger foundation for software supply chain defense than either traditional signature-based tooling or single-model machine-learning approaches, and that such a framework can be operationalized as a composite, tiered risk score suitable for direct integration into DevSecOps decision gates and SBOM-based governance programmes aligned with emerging regulatory frameworks.

Recommendations

Based on the findings of this study, the following recommendations are proposed for practitioners, tool vendors, and future researchers:

1. **Adopt multi-signal, not single-model, risk scoring:** Organizations implementing AI-based supply chain security should prioritize frameworks that fuse structural (dependency-graph), semantic (code/commit), and temporal (pipeline) signals,

given the demonstrated non-redundancy of these signal types, rather than relying on a single detection technique.

2. **Integrate composite risk scores into CI/CD gates:** The three-tier risk classification (Low/Medium/High) demonstrated in this study should be embedded directly into build and release pipelines so that High-Risk components can trigger automated quarantine, complementing rather than replacing human SOC review.
3. **Treat SBOMs as one input among several, not a standalone control:** Given documented inconsistencies across SBOM generation tools and formats, organizations should combine SBOM data with independent behavioural and structural signals rather than relying on SBOM completeness alone for assurance.
4. **Invest in explainability tooling for AI-driven security models:** To support SOC analyst trust and satisfy regulatory audit requirements under frameworks such as the NIST SSDF and EU Cyber Resilience Act, future implementations should incorporate attention-visualization and feature-attribution methods alongside the composite score.
5. **Expand malicious-artefact corpora through cross-organizational sharing:** The relatively small confirmed-malicious-package corpus used in this and comparable studies constrains generalization to novel attack patterns; sector-wide, privacy-preserving threat-intelligence sharing initiatives would materially improve training data availability for future AI-driven detectors.
6. **Extend future research to cold-start and zero-day scenarios:** Subsequent work should specifically evaluate framework performance on newly published packages with limited historical data and on attack patterns absent from training data, using red-team-style adversarial evaluation.

References

- Aljaafreh, R., Al-Doghman, F., Hussain, F., Khan, F., & Aljaafreh, A. (2026). Integrating machine learning and business intelligence into supply chain risk management for a comprehensive cybersecurity framework: A systematic literature review. *Technologies*, 14(4), 194. <https://doi.org/10.3390/technologies14040194c>
- Düsing, J., & Hermann, B. (2022). Analyzing the direct and transitive impact of vulnerabilities onto different artifact repositories. *Digital Threats: Research and Practice*, 3(4), Article 37, 1–25. <https://doi.org/10.1145/3472811>
- Feng, X., Gao, Y., Wu, J., & Shi, L. (2025). Research on security risk identification and evaluation in open-source software supply chain. *Proceedings of the 2025 International Conference on Digital Management and Information Technology (DMIT '25)*, 257–266. Association for Computing Machinery. <https://doi.org/10.1145/3736426.3736467>
- Kalouptoglou, I., Siavvas, M., Ampatzoglou, A., Kehagias, D., & Chatzigeorgiou, A. (2025). Transfer learning for software vulnerability prediction using Transformer models. *Journal of Systems and Software*, 227, Article 112448.
- Li, Z., et al. (2018). VulDeePecker: A deep learning-based system for vulnerability detection. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS)*.

- Lin, G., Wen, S., Han, Q.-L., Zhang, J., & Xiang, Y. (2020). Software vulnerability detection using deep neural networks: A survey. *Proceedings of the IEEE*, 108(10), 1825–1848.
- Lins, M., Mayrhofer, R., & Roland, M. (2025). Unveiling the critical attack path for implanting backdoors in supply chains: Practical experience from XZ. In Y. Kim, A. Miyaji, & M. Tibouchi (Eds.), *Cryptology and Network Security: 24th International Conference, CANS 2025, Proceedings* (Lecture Notes in Computer Science, Vol. 16351, pp. 521–541). Springer. https://doi.org/10.1007/978-981-95-4434-9_24
- National Institute of Standards and Technology. (2022). *Secure software development framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities* (NIST Special Publication 800-218).
- Rosso, M., Jaffar, M., Brighente, A., & Conti, M. (2024). Impacts of software bill of materials (SBOM) generation on vulnerability detection. In *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*. ACM.
- Sejfić, A., Das, S., Shafiq, S., & Medvidović, N. (2024). Toward improved deep learning-based vulnerability detection. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)* (pp. 62:1–62:12).
- Sood, A. K., & Zeadally, S. (2025). Malicious AI models undermine software supply-chain security. *Communications of the ACM*, 68(6), 62–71. <https://doi.org/10.1145/3704724>
- Steenhoek, B., Gao, H., & Le, W. (2024). Dataflow analysis-inspired deep learning for efficient vulnerability detection. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE)*.
- Teichmann, F. (2025). The Cyber Resilience Act as a new paradigm for product security: A compliance roadmap. *International Cybersecurity Law Review*. Advance online publication. <https://doi.org/10.1365/s43439-025-00162-4>
- Vera Buschmann, F., Pauliuchenka, P., Oh, E., Kao, B. C., DiValentin, L., & Bader, D. A. (2025). Graph-based profiling of dependency vulnerability remediation. In *Proceedings of the 6th International Conference on Science of Cyber Security (SciSec 2025)*. Springer.
- Zhou, L., Dacier, M., & Konstantinou, C. (2026). A reality check on SBOM-based vulnerability management: An empirical study and a path forward. In *Proceedings of the Sixteenth ACM Conference on Data and Application Security and Privacy (CODASPY '26)* (pp. 255–268). ACM. <https://doi.org/10.1145/3800506.3803490>
- Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems* (Vol. 32, pp. 10197–10207). Curran Associates.