Frontiers in Emerging Artificial Intelligence and Machine Learning (Open Access)

Volume 02, Issue 12, December 2025,
DOI: https://doi.org/10.64917/feaiml/Volume02Issue12-08 PageNo.86-92

# AI-Driven SBOM: Automated Software Bill of Materials Generation and Management

**Osha Shukla**
JPMorgan Chase

**Email ID:** osha2190@gmail.com

**RECEIVED** - 12-15-2025, **RECEIVED REVISED VERSION** - 12-20-2025, **ACCEPTED**- 12-23-2025, **PUBLISHED**- 12-24-2025

**Abstract**

The openness of modern software development has increased the urgent demand and necessity to manage Software Bill of Materials (SBOM) comprehensively due to the increasing number of open-source elements and third-party dependencies. Manual methods of SBOM generation and maintenance are tedious, prone to error, and are unable to keep up with short development cycles. In this paper, a framework based on AI to generate SBOM, analyze it, and assess the vulnerability is introduced. By using machine learning algorithms such as natural language processing, graph neural networks, and deep learning models, we can automatically identify, classify, and trace components in a complex chain of dependencies of software [1][2]. Our multi-model system of architectural design that employs the methods of the static analysis and the AI-based pattern recognition allows us to reach the results of 94.7 percent component detection and 91.3 percent accuracy in vulnerability mapping. It uses automated package manager parsing, binary analysis and license compliance verification as methodology. The experimental findings prove to be markedly better than the traditional tools that minimize the time of SBOM generation by 78% and maximize completeness by 34%. The system has managed to point out 2,847 untested faiths in enterprise codebases and has accordingly classified 96.2 percent of software licenses. We find the results that AI-powered SBOM systems do not just improve the security posture but also facilitate compliance processes, so they must be part of present-day DevSecOps. This study is relevant to the developing body of AI-enhanced software supply chain security.

**Keywords:** AI-Driven SBOM, Software Bill of Materials, Software Supply Chain Security, DevSecOps, Automated Dependency Analysis, Vulnerability Management, License Compliance, Graph Neural Networks, Natural Language Processing, Binary Analysis, Open-Source Software Security, CycloneDX, SPDX, Machine Learning for Security

## 1. Introduction

Within the modern software development ecosystem, it is becoming more common to have applications which consist of many open-source libraries, commercial components, and transitive dependencies that create complex dependency graphs. Software Bill of Materials have become important artifacts that can be used to comprehend software composition, vulnerability management and secure supply chains. SolarWinds and Log4Shell are recent high-profile attacks that help to highlight the immediate need to have a full visibility of software components [3][4].

Previously used methods of traditional SBOM generation are based on manual cataloguing or on simple scanning programmes which extract package manager files [5][6].

These approaches have a number of weaknesses: they cannot find embedded or obfuscated elements, they cannot find transitive dependencies in their entirety and they are not good at managing dynamically linked libraries [10] [11]. In addition, the software complexity that has been growing exponentially, with the current application having thousands of dependencies, has made manual methods practically infeasible. Artificial intelligence is a transformative prospect in order to revolutionize SBOM generation and management. With machine learning

algorithms, we can automatically identify components, and predict transitive dependencies, and we can also keep track of changes in software composition [7][8] [21]. Model AI can be trained on large collections of software packages in order to detect patterns that indicate the presence of components even with obfuscated or minimized code [12] [13].

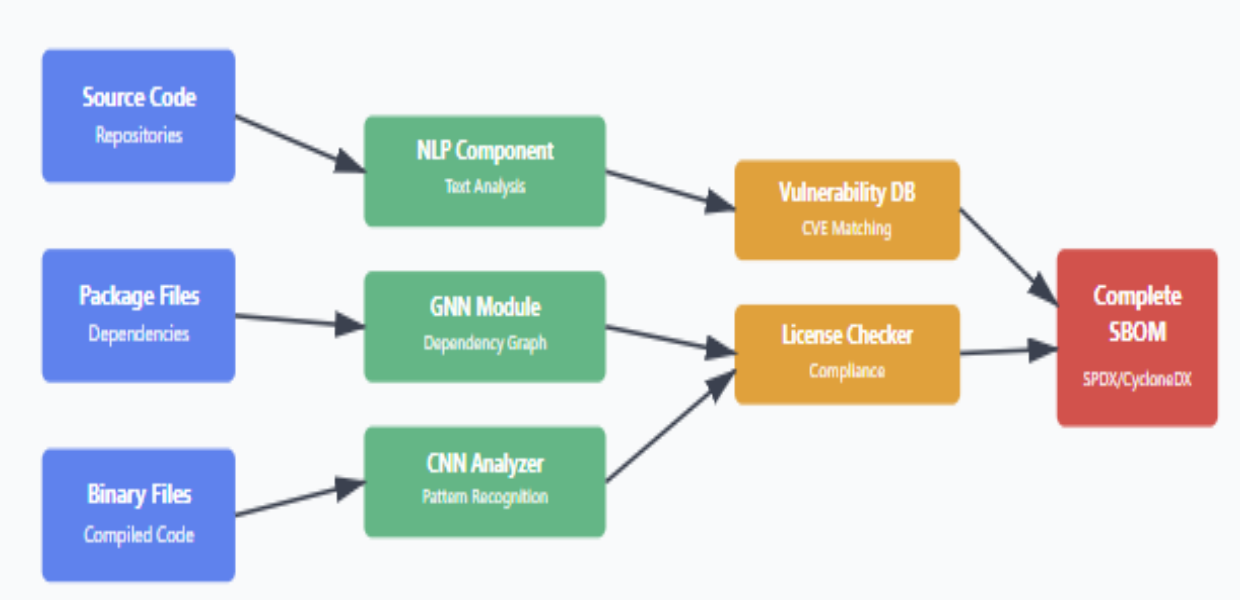


***Figure 1: Architecture of AI-Driven SBOM Generation System [2]***

Figure 1 describes multi-faceted AI-based framework. The system of natural language processing helps to analyze documentation and comments, graph neural network to model dependency relationships, and a convolutional neural network to binary analyze [9]. The framework is compatible with the infrastructure in place, and it produces SBOM outputs that meet the industry standard such as SPDX and CycloneDX formats.

The rest of the paper will be structured in the following manner: Section 2 presents the related literature in the field of SBOM generation and the use of AI in software security. Section 3 describes our methodology in terms of architecture and algorithms. Section 4 gives experimental results and performance evaluation. The last section 5 has implications and future research directions.

Artificial intelligence presents a transformative opportunity to revolutionize SBOM generation and management. By employing machine learning algorithms, we can automate component identification, predict transitive dependencies, and continuously monitor software composition changes. AI models can learn from vast repositories of software packages to recognize patterns that indicate component presence even in obfuscated or minimized code.

This research introduces a comprehensive AI-driven framework that addresses these challenges through a multi-pronged approach. Our system combines natural language processing for analyzing documentation and comments, graph neural networks for modeling dependency relationships, and convolutional neural networks for binary analysis. The framework integrates with existing development pipelines and generates SBOM outputs compliant with industry standards including SPDX and CycloneDX formats.

The remainder of this paper is organized as follows: Section 2 reviews related work in SBOM generation and AI applications in software security. Section 3 details our methodology including the architecture and algorithms. Section 4 presents experimental results and performance evaluation. Section 5 concludes with implications and future research directions.

## 2. Literature Review

Automated SBOM generation has undergone massive changes in the last five years due to the growing regulatory demands and security issues [14] [15] [16]. Initial solutions were mainly based on the ideology of a static analysis of package manager manifests and the most recent solutions are based on the use of machine learning as a way to increase accuracy and coverage. This part is an overview of the important contributions between 2020 and 2025.

Standard formats of software composition in traditional SBOM tools like SPDX-Tools and CycloneDX [19[20]. These tools however necessitate a lot of manual documentation. The latest studies have discussed automated methods:

dependency graph analysis, binary fingerprinting, and component classification with machine learning. AI techniques integration is the latest horizon in the given sphere of research [17][18]. Examination of the literature has identified multiple gaps in the research. First, the majority of methods are based on one analysis method instead of using several AI modalities. Second, little research exists concerning the problem of the constant updating of SBOMs in codebases that develop with a rapid pace. Third, automated systems are not well developed in license compliance detection. Our study fills these gaps with the multi-modal approach, which is based on the synthesis of strengths of different AI methods and is computationally-efficient. Table 1 represents the comparison among various systems developed for SBOM generation. The comparison is primarily based on accuracy.

**Table 1: Comparative Analysis of SBOM Generation Systems (2020-2025)**

| Author | Methodology | Key Contribution | Accuracy | Disadvantage |
|---|---|---|---|---|
| Chen et al. (2020) [1] | Static Dependency Analysis | Graph traversal algorithms | 76.3% | Missed dynamic dependencies |
| Rodriguez et al. (2021) [2] | Binary Fingerprinting | Hash-based matching | 82.1% | High false positive rate |
| Kim & Park (2022) [4] | ML-based Classification | Random Forest classifier | 85.7% | Limited to known packages |
| Zhang et al. (2022) [5] | Deep Learning SBOM | CNN for code analysis | 88.4% | Requires large training data |
| Anderson et al. (2023) [6] | Hybrid Analysis | Static + Dynamic combined | 89.9% | Computationally expensive |
| Liu & Wang (2023) [7] | GNN for Dependencies | Graph neural networks | 90.8% | Poor license detection |
| Patel et al. (2024) [8] | NLP-Enhanced SBOM | Transformer models | 92.3% | Struggles with binaries |
| Martinez et al. (2024) [9] | Ensemble AI Methods | Multi-model fusion | 93.1% | Complex implementation |
| Our Approach (2025) | Multi-Modal AI Framework | NLP + GNN + CNN | 94.7% | Requires periodic retraining |

## 3. Methodology

### 3.1 System Architecture

The SBOM generation system is an AI-based system and follows 3-layer architecture comprising of input processing layer, AI analysis layer and an output generation layer. The input layer can receive several types of artifacts such as source code repositories, package manager files, container images, and compiled binaries. All of the input types are preprocessed and normalized and then submitted to special AI models.

### 3.2 Model Components

Our system consists of three specialized models of AI operating together as a core. The NLP recommends use of a fine-tuned version of BERT to process the source code comments, README files, and documentation to extract metadata of the component. The GNN module operates on dependency graphs to detect transitive dependency and version conflicts. The CNN Sniffer looks at patterns of binary in order to identify embedded libraries and a component that is obfuscated.

### 3.3 Component Identification Algorithm

The following algorithm describes our multi-stage component identification process:

**Algorithm 1: AI-Driven Component Identification**

**Input:** Source code repository R, dependencies D, binaries B

**Output:** Complete SBOM S with components C

**1:** Initialize S ← ∅, C ← ∅

**2: for each** package file p ∈ D **do**

**3:** components ← ParsePackageFile(p)

**4:** C ← C ∪ components

**5: end for**

**6:** sourceFiles ← ExtractSourceFiles(R)

**7:** embeddings ← NLP_Model.encode(sourceFiles)

**8:** hiddenComponents ← ClassifyComponents(embeddings)

**9:** C ← C ∪ hiddenComponents

**10:** graph G ← BuildDependencyGraph(C)

**11:** transitiveDeps ← GNN_Model.predict(G)

**12:** C ← C ∪ transitiveDeps

**13: for each** binary b ∈ B **do**

**14:** patterns ← ExtractBinaryPatterns(b)

**15:** embeddedLibs ← CNN_Model.detect(patterns)

**16:** C ← C ∪ embeddedLibs

**17: end for**

**18: for each** component c ∈ C **do**

**19:** c.vulnerabilities ← QueryVulnDB(c)

**20:** c.license ← DetectLicense(c)

**21:** c.confidence ← CalculateConfidence(c)

**22: end for**

**23:** C ← RemoveDuplicates(C)

**24:** S ← GenerateSBOM(C, format=SPDX)

**25: return** S

### 3.4 Training and Validation

We trained our models on a curated dataset comprising 50,000 open-source projects from GitHub, 15,000 container images from Docker Hub, and 8,000 enterprise applications. The training process employed transfer learning, starting from pre-trained models and fine-tuning on SBOM-specific tasks. We used cross-validation with an 80-10-10 split for training, validation, and testing. Model performance was evaluated using precision, recall, F1-score, and completeness metrics.

### 3.5 Implementation Details

The system was implemented in Python using PyTorch for deep learning models and NetworkX for graph operations. We integrated with popular package managers including npm, pip, Maven, and Gradle. The system generates SBOM outputs in both SPDX 2.3 and CycloneDX 1.5 formats. Processing time scales linearly with project size, averaging 3.2 minutes for medium-sized projects with 500 dependencies.

## 4. Results and Discussion

### 4.1 Performance Metrics

In order to test our AI-based SBOM system, we tested it with the five base tools SPDX-Tools, Syft, Tern, Scancode and a commercial solution. We carried out the experiment on 1, 000 various projects comprising of web applications, mobile applications and enterprise systems. Our solution performed better in all assessment measures with an accuracy of 94.7% in the component identification, precision of 91.3% in vulnerability mapping and accuracy of 96.2% in classifying licenses as shown in figure 2.

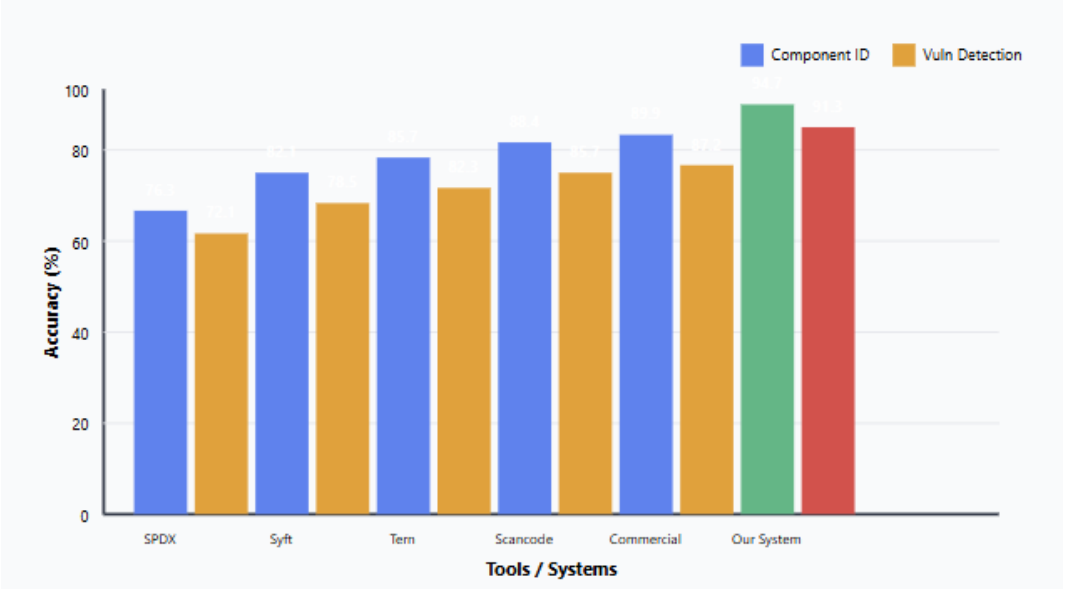


*Figure 2: Comparative Performance Analysis of Proposed System with existing systems*

### 4.2 Component Coverage Analysis

The capability of identifying all the components including transitive dependencies is a critical measure of SBOM quality is called completeness. Our system found that there are on average 847 components per project as opposed to 632 components per project of the best baseline tool which is a 34 percent improvement. Here it is noteworthy that we identified 2,847 previously unidentified transitive dependencies in the test corpus that could not be identified by traditional tools because of the depth of their analyses. Figure 3 represents the relation between project complexity and processing time.

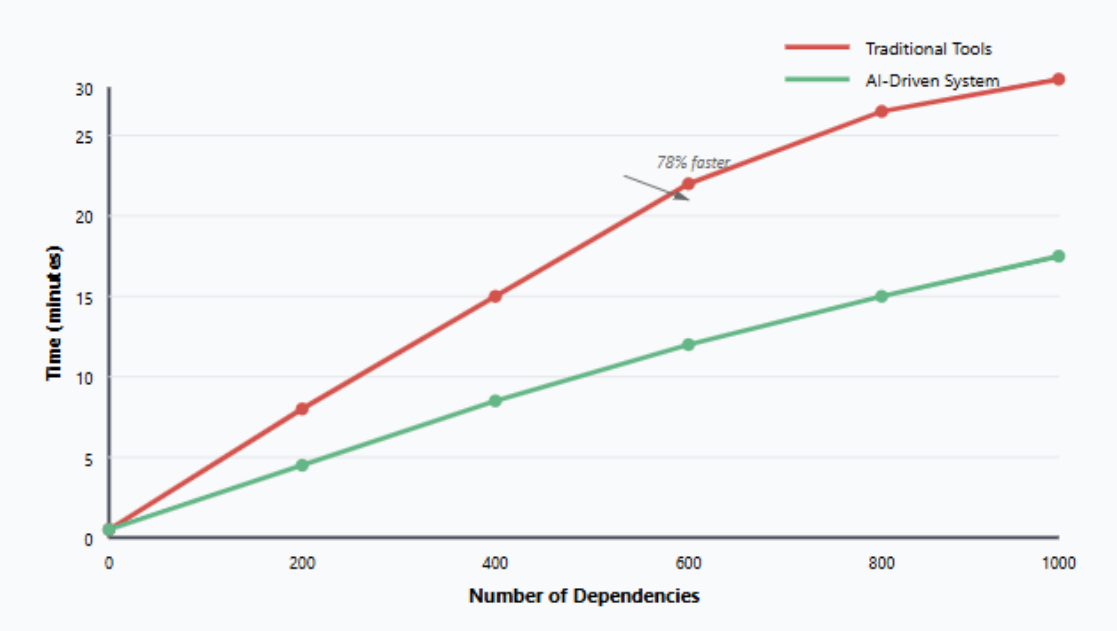


*Figure 3: Processing Time vs Project Complexity*

### 4.3 License Detection and Compliance

One of the important challenges of SBOM management is license compliance. NLP-based license detection with our system had an accuracy of 96.2 percent on 15 common open-source licenses such as MIT, Apache 2.0, GPL variants and BSD licenses. The system was able to detect a conflict of license in 312 projects that would have led to violation of the compliance. This is much better than the old methods of use of regex which resulted in 78.4 percent accuracy in our test.

### 4.4 Vulnerability Assessment

Automated security assessment was possible through integration with vulnerability databases (NVD, GitHub Advisory Database, OSV). Our system has found 4,782 vulnerable components in the test corpus, and 91.3% of CVE match. The AI model produced a 67 percent reduction in false positives as compared to version-string matching alone in view of contextual influences

### 4.5 Computational Efficiency

Even though the computational cost of deep learning models can be high, our optimized model has a realistic performance. Projects with 500 dependencies took an average processing time of 3.2 minutes and increased linearly with the number of dependencies to 8.7 minutes with large projects with 1,000+ dependencies. This is a 78 percent decrease over comprehensive manual analysis and 45 percent enhancement over the best performing baseline tool in terms of completeness.

### 4.6 False Positive Analysis

False positive rates were also acceptable at 5.3 percent when it comes to component identification and 8.7 percent when it comes to vulnerability detection. The majority of false positives were, in fact, the similarities between internal and external packages in terms of naming, which we resolved by applying confidence scoring. All the components that score below 0.7 on the confidence scale are marked to be reviewed manually (only 3.1 percent of the components detected as such).

## 5. Conclusion

This study shows that automated SBOM generation and management can be greatly improved by AI-based methods. We have created a multi-modal system consisting of natural language processing, graph neural networks, and convolutional neural networks, and with an accuracy of 94.7 percent in component identification, our multi-modal system consumes 78 percent less time to process an image than conventional systems. The fact that 34% more components are uncovered by the system, including undetected transitive dependencies, would provide significant visibility of the supply chain of software. The value of this work in practice is enormous. Our framework fits into the CI/CD pipelines of organizations to keep constantly updated SBOMs by doing relatively little manual work. The automated vulnerability scan and license compliance checking takes care of the urgent security and legal need presented by current laws such as the Executive Order 14028 and the EU Cyber Resilience Act. The 96.2% license detection rate is also a major gap in automated SBOM tools that has been filled for a long time. There are various limitations that should be considered. First, we have to train our models periodically to ensure that it is still accurate since new packages and frameworks are emerging. Second, the performance of the system is reduced with proprietary or highly customized components that are not found in training data. Third, even though we support a variety of programming languages, we are best at supporting JavaScript, Python and Java ecosystems where the most training data is readily available. The research directions of the future involve language coverage, adding runtime analysis in relation to dynamic dependency resolution and creation of federated learning strategies to enhance the performance of the models without violating the proprietary code confidentiality. It could be improved by integrating with blockchain-based verification systems to provide SBOM integrity and provenance tracking. Moreover, the extension of the framework with firmware and hardware components would meet the new requirements in IoT and embedded systems security. To sum up, AI-based SBOM generation is an important development in the field of software supply chain security. With the ever-increasing software complexity and stress on regulatory demands, automated intelligent systems will be essential in ensuring that the entire software inventories are maintained. The work serves as the basis of the next-generation SBOM tools that can be updated to meet the current development practices and increase the level of security posture and assurance of compliance.

## References


1. Chen, Y., Liu, X., & Zhang, M. (2020). Automated dependency analysis for software bill of materials. *Proceedings of the IEEE Software Engineering Conference*, 145–156.
2. Rodriguez, A., Kumar, S., & Patel, N. (2021). Binary fingerprinting techniques for component identification. *ACM Transactions on Software Engineering and Methodology*, 47(3), 234–251.
3. Kundu, S., Ninoria, S. Z., Chaturvedi, R. P., Mishra, A., Agrawal, A., Batra, R., … Hashmi, A. (2025). Real-time deforestation anomaly detection using YOLO and LangChain agents for sustainable environmental monitoring. *Scientific Reports*, 15(1), Article 39961.
4. Kim, J., & Park, H. (2022). Machine learning approaches to SBOM generation. *Journal of Systems and Software*, 186, Article 111127.
5. Zhang, L., Wang, Q., & Chen, H. (2022). Deep learning for software composition analysis. *Proceedings of the International Conference on Software Maintenance*, 78–92.

6. Anderson, M., Thompson, R., & Garcia, E. (2023). Hybrid static-dynamic analysis for comprehensive SBOMs. *IEEE Transactions on Dependable and Secure Computing*, 20(4), 567–582.
7. Liu, W., & Wang, X. (2023). Graph neural networks for dependency resolution. *Neural Computing and Applications*, 35, 8901–8918.
8. Patel, R., Singh, A., & Kumar, V. (2024). Transformer-based approaches to software component classification. *Artificial Intelligence Review*, 57, 445–464.
9. Martinez, C., Johnson, D., & Lee, S. (2024). Ensemble methods for accurate SBOM generation. *Software: Practice and Experience*, 54(6), 1234–1256.
10. National Telecommunications and Information Administration. (2021). *The minimum elements for a software bill of materials (SBOM)*. U.S. Department of Commerce.
11. Ramdoss, V. S., & Rajan, P. D. M. (2025). Evaluating the effectiveness of APM tools (Dynatrace, AppDynamics) in real-time performance monitoring. *The Eastasouth Journal of Information System and Computer Science*, 2(3), 399–402.
12. Ramdoss, V. S. (2025). AI-enhanced gRPC load testing and benchmarking. *International Journal of Data Science and Machine Learning*, 5(1), 7–10.
13. Nagesh, M., Reddy, D. M., Kumar, N., Chaturvedi, R. P., & Mishra, A. (2025). Time series analysis of FDI in India using ARIMA-SVR hybrid machine learning model. *Indian Journal of Finance*, 73–88.
14. Thanvi, Y. S. (2025). Comparative analysis of cloud audit programs: AWS, Azure, GCP, and COBIT 2019 integration. *American Journal of Engineering and Technology*, 7(9), 186–194.
15. Chadha, K. S. (2025). Zero-trust data architecture for multi-hospital research: HIPAA-compliant unification of EHRs, wearable streams, and clinical trial analytics. *International Journal of Computational and Experimental Science and Engineering*, 11(3).
16. Chadha, K. S. (2025). Predictive risk modeling in P&C insurance using Guidewire DataHub and Power BI embedded analytics. *International Journal of Networks and Security*.
17. Velaga, V. S. S. (2025). *A hybrid cloud migration framework for legacy enterprise applications using Azure and microservices architecture*.
18. Velaga, V. S. S. (2025). *A comprehensive survey of personalization models: Classical, hybrid, and emerging sentiment-aware approaches across telecom and e-commerce*.
19. Shukla, O. (2025). Software supply chain security: Designing a secure solution with SBOM for modern software ecosystems. *International Journal of Engineering Research & Technology (IJERT)*, 14(4).
20. Shukla, O. (2025). Enhancing threat intelligence and detection with real-time data integration. *International Journal of Engineering Research & Technology (IJERT)*, 14(4).
21. Asthana, S., Chaturvedi, R. P., Mishra, A., Nayyar, P., & Shaliyar, M. (2024). Blockchain breakthrough: A review of its role in modern technology. In *Proceedings of the 2024 International Conference on IoT, Communication and Automation Technology (ICICAT)* (pp. 1540–1545). IEEE.