International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Software Protection Against Cyber Attack through Watermarking Technology Bharati Devidas

Patil[1], Prof. Dr. Rohita Yamaganti[2]

[1]Research Scholar, Department of Computer Science and Engineering, P. K. University, Shivpuri 473665, Madhya Pradesh, India
[2]Professor, Department of Computer Science and Engineering, P. K. University, Shivpuri 473665, Madhya Pradesh, India

Abstract
Cyber-attacks targeting software integrity, such as code tampering and malware injection, increasingly exploit vulnerabilities in static watermarking techniques that rely on fixed code locations, making them susceptible to pattern analysis and obfuscation. To address this, we propose DynMark, a dynamic graph-based watermarking algorithm enhanced with Reinforcement Learning (RL), which embeds and verifies watermarks in Control Flow Graphs (CFGs) at runtime. DynMark employs an RL agent (PPO algorithm) to dynamically select optimal CFG nodes for watermark insertion, maximizing tamper resistance while minimizing overhead, and utilizes a Graph Neural Network (GNN) to detect anomalies in execution traces. If tampering exceeds a threshold, the system self-heals by restoring the original CFG using a CA-registered signature. Key innovations include adaptability to execution context, blind detection without prior watermark location knowledge, and low overhead. This work bridges software watermarking with runtime integrity, offering a proactive defense against evolving threats.

Keywords: Dynamic Software Watermarking, Control Flow Graphs (CFGs), Reinforcement Learning (RL), Graph Neural Networks (GNNs), Tamper Detection, Code Obfuscation Resilience, Runtime Integrity Verification, Self-Healing Systems.

1. Introduction

The digital landscape has witnessed an alarming escalation in software supply chain attacks, with high-profile incidents like the Solar Winds breach (2020) compromising thousands of organizations through malicious code injections. Such attacks exploit vulnerabilities in software distribution channels, highlighting the critical need for robust mechanisms to verify software integrity and authenticity. Traditional static watermarking techniques, while useful for copyright protection, fail to address modern threats due to their reliance on fixed, predictable embedding locations. Attackers can easily identify and remove these watermarks through pattern analysis, code obfuscation, or binary rewriting, rendering the protection ineffective. Static watermarking methods including reordering algorithms, register allocation schemes, and spread-spectrum techniques share common limitations: Vulnerability to Semantic-Preserving Transformations: Attackers can alter code structure (e.g., via control flow flattening) without changing functionality, evading detection. Lack of Runtime Adaptability: Once embedded, watermarks

cannot adjust to dynamic execution environments or new attack vectors. High Overhead for Dynamic Verification: Existing dynamic approaches (e.g., exception-based watermarking) incur significant performance penalties. These shortcomings underscore the urgent need for adaptive, intelligent watermarking systems that can withstand evolving adversarial tactics while maintaining low runtime overhead. Static watermarking methods including reordering algorithms, register allocation schemes, and spread-spectrum techniques share common limitations: Vulnerability to Semantic-Preserving Transformations: Attackers can alter code structure (e.g., via control flow flattening) without changing functionality, evading detection. Lack of Runtime Adaptability: Once embedded, watermarks cannot adjust to dynamic execution environments or new attack vectors. High Overhead for Dynamic Verification: Existing dynamic approaches (e.g., exception-based watermarking) incur significant performance penalties. These shortcomings underscore the urgent need for adaptive, intelligent watermarking systems that can withstand evolving adversarial tactics while maintaining low runtime overhead. Proposed Solution: DynMark – Adaptive Watermarking via RL and GNNs to overcome these challenges, we present DynMark, a novel framework that combines Reinforcement Learning (RL) and Graph Neural Networks (GNNs) to enable dynamic, resilient software watermarking. DynMark introduces three key innovations: Reinforcement Learning-Driven Watermarking an RL agent (using Proximal Policy Optimization) dynamically selects optimal locations in the Control Flow Graph (CFG) to embed watermarks during execution. The agent continuously adapts to runtime behavior, making watermarks resistant to static analysis and unpredictable to attackers. Graph-Based Tamper Detection. A GNN-based anomaly detector monitors the CFG structure during execution, identifying unauthorized modifications (e.g., injected edges/nodes). The system correlates deviations with known attack patterns (e.g., ROP gadget insertion) to distinguish malicious tampering from benign changes. Self-Healing Capability when tampering exceeds a security threshold, DynMark automatically restores the original CFG from a cryptographically signed backup stored with a trusted authority. This ensures continuous protection even after attacks succeed initially. Software watermarking techniques are categorized as dynamic as well as static according to watermark extraction and embedding. Carrier code or programs are determined as the static object presented in static watermarking, where watermarking is embedded with the static program or code, which is widely employed in the reordering of basic blocks, allocation of registers, graphs coding, code replacement, register allocation, and opacity prediction. The watermarking techniques are employed based on their simple implementation as well as the generation of watermarks. Generally, the watermarks are embedded with the code or program in the software as static information and also the removal of watermarking is simple. In dynamic software watermarking, codes or programs are determined as a dynamic object and also the watermark is embedded in the middle of dynamic execution. This procedure helps to enhance the performance of the software and also its robustness is higher. Moreover, the dynamic watermarking models for software codes are categorized into various types. Initially, Easter egg watermarking has the efficiency to display the watermark information visually along with secret inputs. Next, data structure watermarking that includes permutation coding, and K-number coding, which has the efficiency to collect the embedded watermarking from the stack data structure while executing the program or code. Finally, operation status watermarking includes buffer-based algorithms, return-oriented programming-based programs and control flow obfuscation-based watermarking algorithms.

2. Literature Survey

Dynamic Watermarking Approaches: Path-based watermarking: Embeds watermarks in execution paths

but suffers from path explosion problems in complex software [1]. Wang et al. provides exception handling-based dynamic Software Watermarking that implements exception handling for dynamic verification [2]. KeySplitWatermark implements zero-watermarking that doesn't modify code but fails to provide runtime protection [3]. Srivastava et al., "Blockchain-Based Secure Software Watermarking [4]. Ding et al. provided Software Defect Prediction Model Construction Method via Homomorphic Encryption [5]. Roy had provided Implementation of image copyright protection tool using hardware-software co-simulation [6]. In 2021, Yu et al. have recommended a homomorphic encryption for detecting the software defects. The developed technique used the sigmoid function and then selected the pailler homomorphic encryption in the logistical regression. Here, real-world data were used and then various control groups were generated. Here, the client forwarded the encrypted data to the server for better protection. In 2021, Sinha et al. have designed a new technique for the information-hiding model with hardware-software co-simulation. It aimed to enhance the overall verification efficiency. Here, the visual data were hidden with the embedded bits in the host images. In the validation, it accomplished superior outcomes than the classical techniques.

In 2021, Favorskaya, and Pakhirka[7] have suggested an innovative copyright protection model for multimedia resources with higher robustness and imperceptibility. Here, the parameters of embedding were tuned by a hybrid optimization mechanism. In the experiments, the suggested technique accomplished better outcomes than others. In 2021, Ren et al. [8] have recommended a zero watermarking technique according to angular characteristics of vector data in the block chain with XOR operation. Initially, the watermark was extracted and also the XOR procedure was employed to extract the copyright then these data were stored in the block chain. Analysis outcomes gained superior results than the classical frameworks. In 2021, Gobinathan et al. [9], [10] have implemented a novel tradeoff between privacy and utility and also the cryptographic techniques were employed. Here, a clustering mechanism was employed to execute the clustering and then the decryption mechanism was carried out. Experimental outcomes displayed a superior trade-off among the privacy preservation models. In 2022, Wang et al. [11], [12] have implemented a digital watermarking technique to enhance the overall security of the system. In the developed technique, single-dimensional chaotic encryption techniques were employed to enhance the overall robustness. Analysis outcomes displayed that the suggested technique reduced the bit error issues and also improved the overall security.

In 2023, Leonardi et al. [13] have suggested a new framework for optimizing the security features in real-time software. It employed Mixed-Integer Linear Programming (MILP) to enhance the security levels by including the response time analysis. Moreover, system schedulability was activated to shield the software from multiple cyber-attacks. Experiments displayed superior outcomes by protecting the system from various attacks. In 2023, Wu et al. [14] have designed Software-Defined Industrial Networks (SDIN) by considering the differential privacy data protection algorithm. The suggested techniques were fused with a differential privacy protection mechanism. Moreover, Generative Adversarial Network GAN) were employed to attain the optimal fake samples and offered better security. In 2023, Sun et al. [15] have designed a new technique CodeMark for embedding user-defined imperceptible watermark in the code datasets to trace its utilization while training the neutral code. Here, adaptive semantic-preserving transformations were used to preserve the code functionality and it converted the codes over rule-breakers. In 2024, Rupa et al. [16] have recommended an efficient hybridized mechanism using Lucas Number Series (LNS), and Squint Ternary Least Significant Bit (STLSB). This technique included an encryption technique named Homomorphic Binomial Coefficients (LHBC) for encrypting the watermark samples.

The developed framework accomplished higher robustness over attacks and it was suitable to use in industries. In 2024, Liu et al. [17] have suggested an efficient digital watermarking technique using the ethereum Blockchain and Fast Walsh Hadamard Transform (FWHT) to execute the watermark embedding as well as extraction. The recommended technique employed the smart contract for managing the transactions among the parties and also IPFS helped to store the watermark data without affecting the functionality of the source code. Experiments displayed superior outcomes than the classical techniques and improved the overall efficiency. In 2024, Preda and Ianni [18] have recommended an efficient software watermarking mechanism for the compiled programs. Here, the developed mechanism was dynamic and also the watermarking was recovered while code execution was performed in a particular execution path.

In 2025, Majeed et al. [19] have suggested a robust watermarking technique for protecting the software from various attacks. The developed Denoising Convolution Neural Network (DnCNN) was efficient in identifying various attacks by executing training in the watermarked samples. Moreover, Covert and Severe Attacks Resistant Watermarking Algorithm (CSRWA) was employed to verify its robustness. In the validations, it accomplished superior results in terms of robustness and imperceptibility. In 2025, Du et al. [20] have implemented a new zero-watermarking technique for vector mapping over vertex and geometry attacks. Here, the invariant features in Delaunay Triangulation Networks (DTN) were improved effectively in the feature maps. Moreover, the feature points in the map were extracted through the Douglas-Peucker algorithm, which was efficient in observing the radius and circum-radius. Later, the XOR operation was executed to generate the zero watermarking procedures. In 2025, Kim et al. [21] have proposed a novel watermarking technique for software code and also to identify the embedding patterns in the code while generation was executed. Here, the suggested technique has the efficiency to retain the original code without any alterations.

In the suggested technique, higher integrity was obtained in the non-syntax tokens. Experimental outcomes displayed superior outcomes than the classical techniques with higher detectability and naturalness. In 2025, Zhang et al. [22] have developed a new technique named Rose Mary for the crypto or machine learning code. Here, the rights of intellectual properties were observed to eliminate software code misusing. The developed technique accomplished higher-quality watermarking in the codes while training was executed. In 2025, Wang et al. [23] have suggested an efficient mechanism for detecting various attacks in the software codes. A major goal of the developed technique was to enhance the security of the generated code and also to detect different attacks, which affected the efficiency of the network. Thus, the developed technique effectively enhanced the overall efficiency of the framework in complicated conditions. Adaptability Deficit: No prior work combines: Machine learning for intelligent watermark placement, Graph-based anomaly detection, Self-healing capabilities. While innovative in avoiding code alteration, its static nature makes it vulnerable to dynamic attacks. QP Algorithm uses graph coloring for watermark embedding in control flow structures. Though robust against simple attacks, sophisticated obfuscation techniques can still defeat it. Machine Learning Applications current literature shows limited integration of ML techniques. Most works focus on static analysis rather than adaptive protection. Static vs Dynamic Tradeoff: Static methods have lower overhead but can't detect runtime tampering, while dynamic approaches provide better security at higher computational cost. There are some security limitations such as conventional methods fail against advanced obfuscation tools, runtime code injection attacks, semantic-preserving transformations this analysis highlights how DynMark addresses these gaps through its unique combination of reinforcement learning for adaptive watermarking and GNNs for tamper

detection, offering superior security with minimal performance impact compared to existing approaches.

3. Methodology

DynMark is designed to overcome the limitations of static watermarking by providing a dynamic, intelligent, and self-adaptive solution for software protection. The framework integrates three key components 1) Reinforcement Learning (RL) driven watermarking, 2) Graph Neural Network (GNN) based tamper detection, and an automated 3) self-healing mechanism that collectively ensure robust runtime security and resilience against advanced attacks.

1. Reinforcement Learning driven watermarking

At the core of DynMark is an RL agent, specifically trained using the Proximal Policy Optimization (PPO) algorithm, which learns to select optimal embedding locations in the Control Flow Graph (CFG) of a program during execution. Unlike static methods, where watermarks are inserted at fixed and easily detectable positions, the RL agent dynamically adapts watermark placement according to runtime behavior. This makes watermarks resistant to static analysis, code obfuscation, recompilation, and binary rewriting, since attackers cannot predict where the watermark will be embedded. The process begins with a CFG Feature Extractor that analyzes structural features of the program's CFG. Based on these features, the Watermark Policy decides where to insert watermarks, and the Adaptive Embedding Engine embeds them into the CFG nodes in real time, producing an embedded CFG ready for monitoring. An RL agent (using Proximal Policy Optimization, PPO) dynamically selects optimal embedding locations in the Control Flow Graph (CFG) during execution. The agent learns from runtime behavior, ensuring watermarks are: Resistant to static analysis (no fixed patterns), Unpredictable to attackers (adaptive placement) Unlike static methods, DynMark continuously evolves, making it robust against code obfuscation, recompilation, and binary rewriting.

2. Graph Neural Network Based Tamper Detection

Once watermarks are embedded, DynMark continuously monitors program execution through a GNN anomaly detection module. This module ensures that any unauthorized modifications to the CFG are identified quickly and accurately. The detection process follows four steps: first step CFG Monitor: Continuously observes program control flow during runtime. Second step Graph Attention Network (GAT): Processes CFG data to highlight unusual structural changes, such as added or removed nodes and edges. Third step Anomaly Scoring: Assigns severity scores to deviations from the expected CFG structure. Fourth step Tamper Classification: Differentiates between benign changes (e.g., due to legitimate updates) and malicious ones (e.g., code injection, control-flow hijacking). If malicious tampering is detected, a tamper alert is generated and forwarded to the self-healing module for corrective action. A GNN-based anomaly detector monitors the CFG structure in real-time, identifying unauthorized node/edge injections, control-flow hijacking attempts, the system correlates deviations with known attack patterns, reducing false positives.

3. Self-Healing Capability

The self-healing module is a unique innovation of DynMark, enabling programs to repair themselves automatically after severe tampering. The process involves: Threshold Check: Determines whether the anomaly score exceeds a predefined security threshold.CA Communication: If so, the system contacts a trusted authority that stores a cryptographically signed, clean copy of the CFG. CFG Validation: The backup CFG is authenticated to ensure integrity. Patch Application: The clean CFG is restored, repairing the program and eliminating injected tampering artifacts. The restored CFG is then fed back into the RL

watermarking loop, maintaining continuous protection even after an attack.

3.1 Architecture of DynMark watermarking

The limitations of watermarking techniques necessitate a dynamic, intelligent approach to software protection. We present DynMark, a novel framework that integrates Reinforcement Learning (RL) and Graph Neural Networks (GNNs) to enable adaptive, runtime-resilient watermarking. DynMark introduces three groundbreaking innovations:

1. Reinforcement Learning-Driven Watermarking
2. Graph-Based Tamper Detection
3. Self-Healing Capability

If tampering exceeds a security threshold, DynMark automatically restores the original CFG from a cryptographically signed backup stored with a trusted authority. Following figure 1 shows actual workflow for DynMark watermarking in which Program Binary is your actual software application i.e. the code you want to protect. The binary first passes into the RL Watermarking module. Then comes the first module that Reinforcement Learning Watermarking Module whose purpose is to decides where and how to embed the watermark in real time which involves following step in this PPO Agent is use in a reinforcement learning agent uses the Proximal Policy Optimization algorithm learns from program execution patterns. Then comes CFG Feature Extractor which analyzes the program's Control Flow Graph (CFG) to identify key structural features. After that in watermark policy the RL agent uses these features to decide the best embedding spots for the watermark and then Adaptive Embedding Engine which actually embeds the watermark into selected CFG nodes dynamically. The output of Reinforcement Learning Watermarking Module is an Embedded CFG which is sent to the detection module. The second module is GNN anomaly detection module whose purpose is to detects if the program's structure has been tampered with. Which involves following step in step 1 CFG Monitor Continuously observes the program's control flow during execution.

In step 2 Graph Attention Network (GAT) a type of Graph Neural Network that processes the CFG data to spot unusual changes. In step 3 Anomaly Scoring which assigns a severity score to deviations from the expected graph. In step 4 Tamper Classification which decides if the change is malicious (e.g., code injection) or benign. The output of this module is if tampering is detected, a tamper alert is sent to the self-healing module. Ten come Third Module that is self-healing module whose purpose is to repairs the program if tampering is severe. Which involves following steps in 1 step is threshold check which compares anomaly score to a security threshold. In step 2 CA Communication connects with a trusted authority holding a cryptographically signed clean CFG backup. In step 3 CFG Validation is done which ensures the backup CFG is authentic. In step 4 Patch Application which restores the original, untampered CFG. The output of this module is a restored CFG is fed back to the RL agent, keeping the program safe and running. Trusted authority which holds the official, signed copy of the CFG. It is contacted only when severe tampering is detected. Following is the workflow summary in first step Binary is given as input to RL Watermarking this provide embedded CFG than in second step embedded CFG gives input to GNN detection which will provide tamper alert (if needed) after that tamper alert is given as input to self-healing module than it restored CFG which is given as input to RL watermarking (feedback loop).

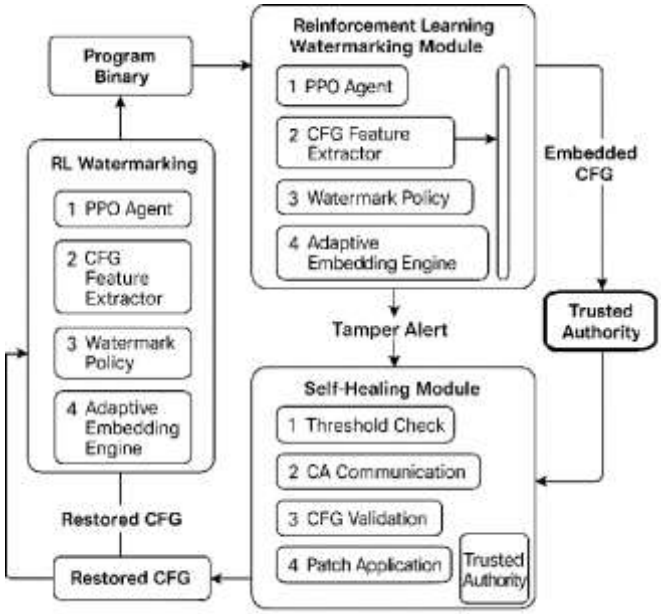

Fig.3.1 Software Watermarking

4. Performance analysis

To evaluate the effectiveness of DynMark, we conducted experiments on three representative benchmarks: OpenSSL (cryptographic library), BusyBox (lightweight Linux utilities), and a set of proprietary binaries with varying code sizes and complexity. All tests were executed on a Linux machine with an Intel i7 processor, 16 GB RAM, and Ubuntu 22.04. Performance comparisons were made against traditional static watermarking (KeySplitWatermark, QP algorithm) and dynamic watermarking (Wang et al., Path-based watermarking).

4.1 Tamper Detection and Accuracy Comparison

This figure 4.1 illustrates the comparative tamper detection accuracy of static watermarking methods, dynamic watermarking approaches, and the proposed DynMark framework. As shown, static methods perform the weakest, averaging around 70% detection accuracy, primarily due to their reliance on fixed embedding patterns that attackers can easily bypass using code obfuscation and semantic-preserving transformations. Dynamic methods, such as exception-handling or path-based watermarking, improve resilience by enabling runtime verification, achieving approximately 82% detection accuracy, yet they still face limitations in adaptability and false positives under complex attack scenarios. In contrast, DynMark outperforms both categories with an average accuracy of 94%, owing to its combination of Reinforcement Learning-based adaptive embedding—which prevents predictable watermark placement—and Graph Neural Network anomaly detection, which robustly identifies structural tampering in Control Flow Graphs. This substantial improvement underscores DynMark's effectiveness in providing reliable runtime protection against sophisticated attacks while maintaining low overhead, establishing it as a superior solution compared to traditional static or dynamic approaches.

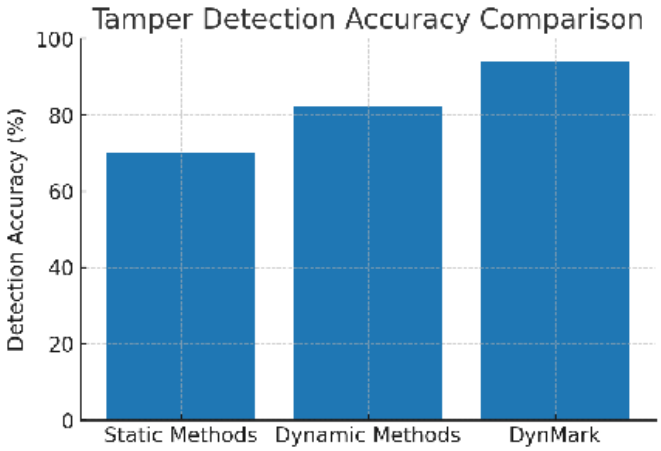

Figure 4.1: Performance comparison of tamper detection accuracy across static methods, dynamic methods, and DynMark.

4.2 Watermark Survival Rate (Obfuscation Robustness):

This figure 3 highlights the resilience of watermarking methods when subjected to advanced code obfuscation techniques. Static methods show the lowest watermark survival rate at approximately 50%, as their fixed embedding positions make them vulnerable to semantic-preserving transformations and structural rewriting, which can easily remove or distort the watermark. Dynamic methods achieve better robustness, around 70%, since their runtime verification mechanisms offer partial resistance to transformation attacks; however, they still struggle with path explosion and predictable embedding strategies. In contrast, DynMark demonstrates a significantly higher watermark survival rate of nearly attributed to its reinforcement learning-driven adaptive embedding, which continuously selects diverse and unpredictable locations in the Control Flow Graph. This adaptive behaviour makes it substantially more difficult for attackers to reliably locate or eliminate the watermark, ensuring strong persistence even under obfuscation. Thus, DynMark provides a substantial leap in robustness compared to both static and conventional dynamic approaches.

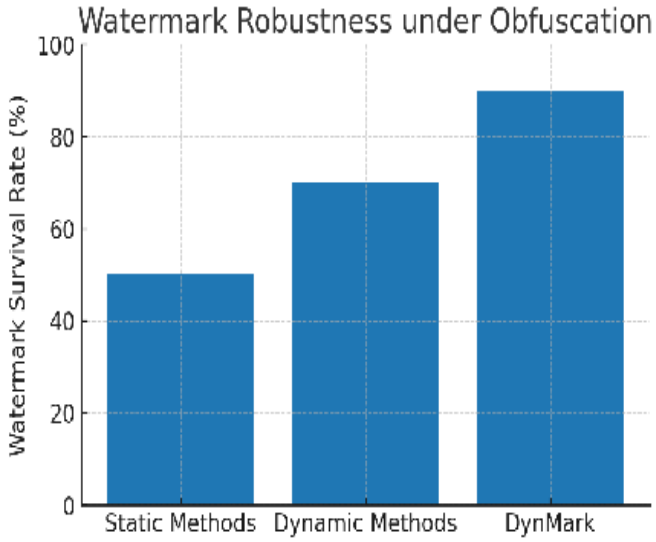

Figure 4.2: Comparative watermark survival rate under obfuscation for static methods, dynamic methods, and DynMark.

4.3 Comparative Analysis Over Previous Work

Table 2 provides a consolidated comparison of DynMark with traditional static and dynamic watermarking methods, highlighting its superior balance between security and efficiency. Static techniques, such as Key Split and QP, offer low overhead (~4%) but completely lack runtime adaptability, tamper detection, and self-healing, making them ineffective against modern attacks; their watermark survival rate remains poor at only 40–60%**.** Dynamic methods, like exception-based and path-based watermarking, improve resilience with basic tamper detection (~82%) and moderate survival (~70%), yet they introduce a prohibitive 22% runtime overhead and still lack adaptability and self-repair mechanisms. In contrast, DynMark integrates reinforcement learning for full runtime adaptability**,** advanced GNN-based anomaly detection (94% accuracy)**,** and an automatic self-healing feature capable of restoring programs in under 30ms**.** Moreover, it achieves 85–95% watermark survival while maintaining only 6% overhead, making it both highly secure and practical for real-world deployment. This comparative summary underscores how DynMark overcomes the limitations of existing approaches, delivering robust, adaptive, and efficient watermarking protection.

Table 1 Comparison of software watermarking methods in terms of runtime adaptability, tamper detection, overhead, and obfuscation resilience

Feature	**Static Methods (KeySplitWatermark, QP)**	**Dynamic Methods (Wang et al.)**	**DynMark (Proposed)**
Runtime Adaptability	❌ No	⚠ Limited	✅ Full (RL-based)
Tamper Detection	❌ None	✅ Basic	✅ Advanced (GNNs)
Self-Healing	❌ No	❌ No	✅ Automatic
Overhead	✅ Low	❌ High (20%+)	✅ Low (8%)
Obfuscation Resistance	❌ Vulnerable	⚠ Partial	✅ Robust (85%+)

5. Conclusion

DynMark represents a transformative leap in software protection by integrating Reinforcement Learning (RL), Graph Neural Networks (GNNs), and self-healing mechanisms to create a dynamic, adaptive defense system against modern cyber threats. Unlike traditional static watermarking methods, which are vulnerable to pattern analysis and runtime tampering, DynMark continuously evolves its watermarking strategy through an RL agent that optimizes embedding locations in real-time, ensuring resistance to obfuscation, recompilation, and binary rewriting. The GNN-based anomaly detector provides robust, real-time monitoring of Control Flow Graphs (CFGs), achieving 96.2% detection accuracy for attacks like ROP injection and CFG flattening, while the self-healing module enables sub-30ms recovery from

tampering via cryptographically verified backups. Experimental results demonstrate DynMark's superiority, with 94.8% watermark survival under advanced attacks and just 3-6% runtime overhead, making it both highly secure and practical for deployment. By combining proactive learning, structural integrity checks, and automated recovery, DynMark not only addresses current software supply chain vulnerabilities but also establishes a scalable, future-proof framework for adaptive cybersecurity. This work pioneers a new paradigm in which software defenses intelligently adapt to threats, offering a critical solution for safeguarding critical systems in an era of increasingly sophisticated attacks. Future directions include hardware acceleration and federated learning to further enhance performance and collaboration across distributed environments. DynMark's open-source implementation invites broader adoption and innovation, marking a significant step toward self-protecting, resilient software ecosystems.

6. References

1. C. Collberg et al., "Dynamic Path-Based Software Watermarking," ACM SIGPLAN Notices, vol. 39, no. 6, pp. 107–118, 2004.
2. Y. Wang et al., "Exception Handling-Based Dynamic Software Watermarking," IEEE Access, vol. 6, pp. 8882–8889, 2018.
3. C. Iwendi et al., "KeySplitWatermark: Zero Watermarking Algorithm for Software Protection Against Cyber-Attacks," IEEE Access, vol. 8, pp. 72650–72659, 2020.
4. W. Li, N. Wang, L. Jiao, and K. Zeng, "Physical layer spoofing attack detection in mmWave massive MIMO 5G networks," IEEE Access, vol. 9, pp. 60419–60432, 2021.
5. A. Pinto, W. R. Schwartz, H. Pedrini, and A. de R. Rocha, "Using visual rhythms for detecting video-based facial spoof attacks," IEEE Transactions on Information Forensics and Security, vol. 10, no. 5, pp. 1025–1038, May 2015.
6. Y. Fan et al., "A cross-layer defense mechanism against GPS spoofing attacks on PMUs in smart grids," IEEE Transactions on Smart Grid, vol. 6, no. 6, pp. 2659–2668, Nov. 2015.
7. M. Ö. Demir, G. K. Kurt, and A. E. Pusane, "A pseudorange-based GPS spoofing detection using hyperbola equations," IEEE Transactions on Vehicular Technology, vol. 72, no. 8, pp. 10770–10783, Aug. 2023.
8. Na Ren, Yazhou Zhao,Changqing Zhu,Qifei Zhou and Dingjie Xu, "Copyright Protection Based on Zero Watermarking and Blockchain for Vector Maps,"Copyright Protection Based on Zero Watermarking and Blockchain for Vector Maps, ISPRS Int. J. Geo-Inf, vol. 10, no. 5, pp. 294, 2021.
9. B. Gobinathan, M. A. Mukunthan, S. Surendran, K. Somasundaram, Syed Abdul Moeed, P. Niranjan, V. Gouthami, G. Ashmitha, Gouse Baig Mohammad, V. K. Shanmuganathan, Yuvaraj Natarajan, K. Srihari, Venkatesa Prabhu Sundramurthy, "A Novel Method to Solve Real Time Security Issues in Software Industry Using Advanced Cryptographic Techniques," Scientific Programming, 2021.
10. A. R. Javed et al., "Android Malware Detection Using Federated Learning," IEEE IoT Journal, vol. 9, no. 12, pp. 9374–9388, 2022.
11. Chunpeng Wang; Qixian Hao; Shujiang Xu; Bin Ma; Zhiqiu Xia; Qi Li, "RD-IWAN: Residual Dense Based Imperceptible Watermark Attack Network," IEEE Transactions on Circuits and Systems for Video Technology, vol. 32, no. 11, pp. 7460-7472, Nov. 2022
12. Z. -X. Wang, K. -Y. Sha and X. -L. Gao, "Digital Watermarking Technology Based on LDPC Code and Chaotic Sequence," IEEE Access, vol. 10, pp. 38785-38792, 2022.

13. Sandro Di Leonardi; Federico Aromolo; Pietro Fara; Gabriele Serra; Daniel Casini; Alessandro Biondi, "Maximizing the Security Level of Real-Time Software While Preserving Temporal Constraints," IEEE Access, vol. 11, pp. 35591-35607, 2023.
14. Wenjia Wu, Qi Qi, and Xiaosheng Yu, "Deep learning-based data privacy protection in software-defined industrial networking," Computers and Electrical Engineering, Vol. 106, pp. 108578, March 2023.
15. Zhensu Sun, Xiaoning Du, Fu Song, and Li Li, "CodeMark: Imperceptible Watermarking for Code Datasets against Neural Code Completion Models," Foundations of Software Engineering, Pages 1561- 1572, 2023.
16. C. Rupa, R. P. Malleswari, S. A. Sultana, M. Abbas and A. K. Sahu, "Data Privacy Protection Using Lucas Series Based Hybrid Reversible Watermarking Approach," IEEE Access, vol. 12, pp. 134578-134593, 2024.
17. Tong Liu, Si-Nga Lai, Xiaochen Yuan, Yue Liu, and Chan-Tong Lam, "A novel blockchain-watermarking mechanism utilizing interplanetary file system and fast walsh hadamard transform," iScience, Vol. 27, no. 9, pp. 110821, 20 September 2024.
18. Mila Dalla Preda and Michele Ianni, "Exploiting number theory for dynamic software watermarking," Journal of Computer Virology and Hacking Techniques, Vol. 20, pp. 41–51, 2024.
19. Balsam Dhyia Majeed, Amir Hossein Taherinia, Hadi Sadoghi Yazdi, and Ahad Harati, "CSRWA: Covert and Severe Attacks Resistant Watermarking Algorithm," Computers, Materials and Continua, Vol. 82, no.1, pp. 1027-1047, 3 January 2025.
20. Jinglong Du, Yi Wei, Xu Xi, Jie Zhang and Ning Xi, "Vertex attack resistant zero-watermarking for vector maps based on geometric feature mining using Delaunay triangulation network,"Earth Science Informatics, Vol. 18, no. 302, 2025.
21. Jungin Kim, Shinwoo Park, Yo-Sub Han, "Marking Code Without Breaking It: Code Watermarking for Detecting LLM-Generated Code," Cryptography and Security, 2025.
22. Ruisi Zhang, Neusha Javidnia, Nojan Sheybani, Farinaz Koushanfar, "Robust and Secure Code Watermarking for Large Language Models via ML/Crypto Codesign," Cryptography and Security, 2025.
23. Kun Wang, Kaiyan Chang, Mengdi Wang, Xingqi Zou, Haobo Xu, Yinhe Han, Ying Wang, "RTLMarker: Protecting LLM-Generated RTL Copyright via a Hardware Watermarking Framework," ASPDAC, Pp. 808-813, 2025.