UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Algorithm and System Co-Design for Trustworthy and Secure AI

Permalink

https://escholarship.org/uc/item/6pj4k2dp

ISBN

9798197898531

Author

Zhang, Ruisi

Publication Date

2026-07-13

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Algorithm and System Co-Design for Trustworthy and Secure AI

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Machine Learning and Data Science)

by

Ruisi Zhang

Committee in charge:

Professor Farinaz Koushanfar, Chair
Professor Tara Javidi
Professor Xiaolong Wang
Professor Jishen Zhao

2026

Copyright

Ruisi Zhang, 2026

All rights reserved.

The Dissertation of Ruisi Zhang is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

DEDICATION

To all conscious beings who share the same curiosity and passion to explore the unknown world.

EPIGRAPH

Unity of knowledge and action.

Wang Yangming (1472–1529)

TABLE OF CONTENTS

Dissertation Approval Page iii

Dedication iv

Epigraph v

Table of Contents vi

List of Figures x

List of Tables xiii

Acknowledgements xvi

Vita xix

Abstract of the Dissertation xxi

Chapter 1 Introduction 1
1.1 Provenance Tracing for AI-Generated Content 2
1.2 Authentication for Edge AI Deployment 4
1.3 Intellectual Property Protection for Human-AI Co-created Artifacts 5
1.4 Acknowledgements 6

Part I Provenance and Ownership Tracing for AI-Generated Content 7

Chapter 2 Efficient Natural Language Watermarking with Learned Neural Networks . 8
2.1 Background and Related Work on Natural Language Watermarking 11
2.1.1 LLM Watermarking 11
2.1.2 Comparison 12
2.2 REMARK-LLM Design 13
2.2.1 Problem Formulation 13
2.2.2 REMARK-LLM Training 16
2.2.3 Watermark Insertion 19
2.2.4 Watermark Extraction 20
2.3 REMARK-LLM Evaluations and Results 21
2.3.1 Experimental Setups 21
2.3.2 Watermarking Performance 25
2.3.3 Analysis and Visualizations 28
2.3.4 Ablation Studies 33
2.3.5 Attack Evaluations 36
2.3.6 Overall Evaluations 41
2.4 Conclusion 41
2.5 Acknowledgements 42

Chapter 3 Secure Code Watermarking with ML Crypto Co-design 43
3.1 Background on Code Watermarking and Zero Knowledge Proofs 46
3.1.1 Watermarking Large Language Model-generated Code 46
3.1.2 Zero-knowledge Proofs 47
3.2 RoSeMary Design 49
3.2.1 Problem Formulation 49
3.2.2 RoSeMary Architecture 51
3.2.3 RoSeMary End-to-End Training 53
3.2.4 Secure Watermark Verification 54
3.3 RoSeMary Evaluations and Results 56
3.3.1 Experiment Setup 56
3.3.2 RoSeMary Performance 58
3.3.3 Multi-lingual Code Watermarking 60
3.3.4 Zero Knowledge Watermark Verification Overhead 60
3.3.5 Robustness Evaluations 62
3.3.6 Ablation Study and Analysis 63
3.4 Conclusion 65
3.5 Acknowledgements 66

Part II Secure Authentication and Attestation for Edge AI Deployment 67

Chapter 4 Model Authentication with Edge LLMs Watermarking 68
4.1 Background and Related Work 70
4.1.1 Large Language Model Quantization 71
4.1.2 Machine Learning Model Watermarking 72
4.2 Threat Model 73
4.2.1 Motivation and Scenario 73
4.2.2 Watermarking Criteria 73
4.2.3 Watermarking Threats 74
4.3 EmMark Method 74
4.3.1 *Watermark Insertion* 74
4.3.2 *Watermark Extraction* 77
4.4 EmMark Evaluations and Results 77
4.4.1 Experiment Setup 78
4.4.2 EmMark's Fidelity 79
4.4.3 EmMark's Efficiency 80
4.4.4 Robustness 80
4.4.5 Ablation Study and Analysis 82
4.5 Conclusion 84
4.6 Acknowledgements 84

Chapter 5 Hardware Authentication with Edge LLMs Attestation 85
5.1 Background and Related Work 87
5.1.1 On-device LLM Watermarking 87

5.1.2 TEE for LLM IP Protection . . . 88
5.2 Threat Model . . . 89
5.2.1 Scenario . . . 89
5.2.2 Attackers' capability and potential threats . . . 90
5.3 AttestLLM Method . . . 90
5.3.1 Offline LLM Watermarking . . . 91
5.3.2 Online Attestation with TEE . . . 93
5.4 AttestLLM Evaluations and Results . . . 95
5.4.1 Experiment Setup . . . 95
5.4.2 Watermarking On-device LLMs . . . 97
5.4.3 Attestation Overhead . . . 98
5.4.4 Attestation Strength . . . 98
5.4.5 Robustness . . . 99
5.4.6 Ablation Studies and Analysis . . . 101
5.5 Conclusion . . . 102
5.6 Acknowledgements . . . 102

Part III Intellectual Property Protection for Human–AI Co-Creations . . . 104

Chapter 6 Scalable Combinatorial Optimization Artifact IP Protection . . . 105
6.1 Problem Formulation . . . 108
6.1.1 Problem Formulation . . . 108
6.1.2 Watermarking Principles . . . 109
6.2 Encode Robust Signatures with OptiMarks . . . 110
6.2.1 Structure-aware Node Label . . . 111
6.2.2 Robust Watermark Insertion . . . 111
6.2.3 Provenance Guarantee . . . 113
6.3 Verify Signatures from Suspect Solutions . . . 115
6.4 Distributed Watermark for Large-scale Solutions . . . 116
6.4.1 Distributed Watermark Insertion . . . 117
6.4.2 Distributed Watermark Verification . . . 117
6.5 OptiMarks Evaluations and Results . . . 118
6.5.1 Experimental Results . . . 119
6.5.2 Robustness Evaluation . . . 122
6.5.3 Analysis . . . 124
6.6 Conclusion . . . 125
6.7 Acknowledgements . . . 126

Chapter 7 Automated Integrated Circuit Artifact IP Protection . . . 127
7.1 Background and Related Work . . . 130
7.1.1 Graph Neural Networks for Physical Design . . . 131
7.1.2 VLSI Placement . . . 131
7.1.3 Watermarking in IC Design . . . 133
7.2 Motivation and Problem Formulation . . . 136

7.3 ICMARKS Design . . . 137
7.3.1 *Global Watermarking* (GW) . . . 138
7.3.2 *Detailed Watermarking* (DW) . . . 140
7.3.3 ICMARKS Watermarking . . . 142
7.4 AUTOMARKS Methodology . . . 143
7.4.1 *Wirelength-driven Watermark Search* . . . 143
7.4.2 *Timing-driven Watermark Search* . . . 147
7.4.3 *Watermark Insertion* . . . 149
7.4.4 *Watermark Extraction* . . . 149
7.5 ICMARKS Experiments . . . 149
7.5.1 Experimental Setup . . . 149
7.5.2 Experimental Results . . . 152
7.5.3 Attack Evaluation . . . 160
7.6 AUTOMARKS Experiments . . . 167
7.6.1 Experiment Setups . . . 167
7.6.2 AUTOMARKS Results . . . 168
7.6.3 AUTOMARKS's Robustness . . . 172
7.6.4 Ablation Study . . . 175
7.6.5 Visualizations . . . 177
7.7 Conclusion . . . 178
7.8 Acknowledgements . . . 179

Bibliography . . . 180

LIST OF FIGURES

Figure 1.1. Research overview. 2

Figure 2.1. LLM-generated text watermarking scenario. 14

Figure 2.2. REMARK-LLM's watermarking framework. 17

Figure 2.3. Watermarking strength and semantic preservation comparison of different watermarking frameworks. 30

Figure 2.4. Watermarking performance under attacks. 37

Figure 2.5. Word frequency distribution of original LLM-generated texts and corresponding watermarked texts. 40

Figure 2.6. Overall evaluations of different frameworks. 42

Figure 3.1. Overview of watermark insertion and extraction. 45

Figure 3.2. RoSeMary watermarking overview. 51

Figure 3.3. Robustness evaluation results under different attacks. 63

Figure 3.4. Watermarked examples randomly selected from HumanEval and MBPP dataset. 65

Figure 4.1. EmMark watermarking overview. 75

Figure 4.2. EmMark's performance under parameter overwriting and re-watermarking attacks. 81

Figure 4.3. EmMark's watermark performance with inserted signature lengths increased from 50-bit to 200-bit. 83

Figure 5.1. Attestation overview. 87

Figure 5.2. AttestLLM pipeline. 91

Figure 5.3. Attestation latency and energy overhead (%) of LLMs quantized into INT4 and INT8. Lower is better. 98

Figure 5.4. Attestation overhead with different (f, k) choices. 102

Figure 6.1. Overview of tracing constrained combinatorial solutions with OptiMarks. 106

Figure 6.2. Overview of the constraint combinatorial optimization watermarking process. 110

Figure 6.3. OptiMarks performance under random, optimization, and transformation attacks. 124

Figure 6.4. Watermarking capacity of OptiMarks. 125

Figure 7.1. The VLSI physical design process. It generates the IC layout from a design RTL through synthesis, placement, and routing stages. 132

Figure 7.2. IC layout watermark scenario in the supply chain. 137

Figure 7.3. Global watermarking pipeline with stride = sliding window size. 139

Figure 7.4. Detailed watermarking pipeline with candidate cells from which watermarked cells are selected. 140

Figure 7.5. ICMARKS framework. 142

Figure 7.6. AUTOMARKS flow. 144

Figure 7.7. Watermarking capacity for different frameworks on ISPD'2019 benchmarks. 157

Figure 7.8. Watermarking capacity for watermark regions on ISPD'2019 benchmarks. 157

Figure 7.9. Watermarked Design Examples. 158

Figure 7.10. Impact of watermarking on routability. 160

Figure 7.11. Watermarking performance under different attacks for wirelength-driven placement on the ISPD'2015 and ISPD'2019 benchmarks. 162

Figure 7.12. Watermarking performance under different attacks for timing-driven placement on the ICCAD'2015 benchmarks. 165

Figure 7.13. Rank distribution of the watermarked region appears in the re-scored watermarked region. 166

Figure 7.14. The wirelength-driven watermark search time for different designs in ISPD'2015 and ISPD'2019. 171

Figure 7.15. Watermarking performance under different attacks for wirelength-driven placement on the ISPD'2015 and ISPD'2019 benchmarks. 173

Figure 7.16. Watermarking performance under different attacks for timing-driven placement on the ICCAD'2015 benchmarks. 174

Figure 7.17. Histogram distribution of AUTOMARKS's labels. 177

Figure 7.18. Loss curve during the AUTOMARKS's GNN training. 177

Figure 7.19. Watermarked and Non-watermarked examples. 178

LIST OF TABLES

Table 2.1. Comparison of post-hoc LLM-generated text watermarking schemes. 13

Table 2.2. REMARK-LLM architecture details. 22

Table 2.3. REMARK-LLM hyperparameters during training/inference. 22

Table 2.4. Dataset to benchmark the watermarking performance. 23

Table 2.5. Segment-level watermarking comparison. 25

Table 2.6. Long text sequence watermarking comparison. 27

Table 2.7. Efficiency (time and memory) comparison among watermarking frameworks. 31

Table 2.8. Watermarking integrity on different datasets. 31

Table 2.9. REMARK-LLM performance in watermarking texts generated by different LLM architectures. 32

Table 2.10. Watermarked text examples. 33

Table 2.11. The effectiveness of different model backbones in REMARK-LLM performance. 34

Table 2.12. The effectiveness of different masking percentages and gumbel noises in REMARK-LLM performance. 35

Table 2.13. The effectiveness of different loss coefficients in REMARK-LLM performance. 35

Table 2.14. The effectiveness of different inference hyperparameters in REMARK-LLM performance. 36

Table 2.15. Classification performance on watermarked and non-watermarked texts. .. 41

Table 3.1. RoSeMary performance on watermarking HumanEval, MBPP, and EvalPlus datasets. 59

Table 3.2. Multi-lingual code watermarking performance. 61

Table 3.3. Zero-knowledge proof generation (Proof Gen.) and watermark verification (WM Verif.) overhead. 61

Table 3.4. Different frameworks' watermark insertion overhead. 64

Table 3.5. Impact of loss weights on RoSeMary's performance. 64
Table 3.6. Impact of decoders on RoSeMary's performance. 65
Table 4.1. Watermarked embedded large language models performance. 79
Table 4.2. EmMark's watermarking efficiency. 80
Table 4.3. Effectiveness of different insertion coefficients. 83
Table 4.4. EmMark's integrity evaluation on watermarked and non-watermarked models. 84
Table 5.1. AttestLLM's performance in watermarking on-device LLMs quantized to INT4 and INT8. 97
Table 5.2. Model replacement attack (MPA) performance. 100
Table 5.3. Watermark insertion before/after quantization. 101
Table 5.4. Attestation overhead with/without optimizations. 101
Table 6.1. OptiMarks watermarking performance on MaxCut problem. 120
Table 6.2. OptiMarks watermarking performance on MaxInd problem. 120
Table 6.3. OptiMarks watermarking performance on SAT problem. 120
Table 6.4. OptiMarks watermarking performance on protein prediction problem. 122
Table 6.5. OptiMarks watermarking performance on drug discovery problem. 122
Table 6.6. OptiMarks watermarking performance on social recommendation problem. 122
Table 6.7. OptiMarks watermarking performance in distributed training settings. 122
Table 6.8. Watermarking integrity of OptiMarks. 124
Table 7.1. Comparison of IC watermarking (WM) frameworks. 135
Table 7.2. Node feature construction for wirelength-driven watermark search. 145
Table 7.3. Node feature construction for timing-driven watermark search. 147
Table 7.4. Benchmark statistics and node technology. 150
Table 7.5. Performance on the ISPD'2015 benchmarks. 153

Table 7.6. Performance on ISPD'2019 benchmarks. 153

Table 7.7. Performance on ICCAD'2015 benchmarks. 155

Table 7.8. ICMARKS's efficiency on different benchmarks. 158

Table 7.9. Watermarking strength for different signature lengths by the proposed ICMARKS watermarking framework. 159

Table 7.10. GNN training settings. 168

Table 7.11. GNN architecture $\mathbf{M}_{wl}$ for WL-driven search. 168

Table 7.12. GNN architecture $\mathbf{M}_{timing}$ for timing-driven watermark search. 168

Table 7.13. Performance on the ISPD'2015 benchmarks. 169

Table 7.14. Performance on the ISPD'2019 benchmarks. 169

Table 7.15. Performance on ICCAD'2015 benchmarks. 170

Table 7.16. Watermark search runtime in ICCAD'2015. 171

Table 7.17. The effectiveness of different node feature constructions, learning rate choices, β and γ choices, and layer numbers on AUTOMARKS's performance. 176

ACKNOWLEDGEMENTS

I still believe the top search in my senior undergrad years is “Why doing a Ph.D.?” This journey, however, turns out to be truly fun and rewarding.

First and foremost, I’d like to thank my advisor, Prof. Farinaz Koushanfar, for her steadfast support, patient guidance, and inspiring discussions during my Ph.D. journey. With her mentorship, I learned how to identify important research questions, present my findings to the broader community, and make meaningful contributions to the field. I also deeply respect her curiosity and enthusiasm to push the boundaries of science. These skills and mindsets will continue to benefit me throughout my professional career.

I am sincerely grateful to Prof. Tara Javidi, Prof. Xiaolong Wang, and Prof. Jishen Zhao for serving on my Ph.D. committee and for their constructive feedback on my research. I’d also like to thank Prof. David Z. Pan at UT Austin for the wonderful collaboration through NSF TILOS Institute.

Throughout my research journey, I had the privilege to work with many brilliant mentors, whose insightful discussions on project directions, help in code debugging, and patience in revising my paper drafts have helped me to grow as a researcher. I’m particularly grateful to have the chance to work closely with Prof. Ye Pan during my undergrad years at SJTU and with Mojan Javaheripi, Shehzeen Hussain, Nasimeh Heydaribeni, Rachel Selina Rajarathnam, and Tianyu Liu during my Ph.D. years at UCSD.

I’d like to express my heartfelt gratitude to Tianyu Liu, Will Feng, Andrew Gu, Francisco Massa, Will Constable, and Animesh Jain for hosting my last two summers at PyTorch Meta. It was a dream come true to contribute to PyTorch, the framework I started to use since my undergrad, and to see my research grounded in real-world systems and making industrial impacts.

During the course of my Ph.D., I was fortunate to meet, discuss, and collaborate with Zahra Ghodsi, Huili Chen, Paarth Neekhara, Seira Hidano, Nojan Sheybani, Mingjia Huo, Mengxin Zheng, Yifei Zhao, Neusha Javidnia, and Yaman Jandali. I’d also love to thank Angelica Mcallister for her kind help during my graduate study. Although I didn’t get a chance to

work with Jung-Woo Chang, Xinqiao Zhang, Soheil Zibakhsh Shabgahi, Alexia Reyes, Pedram Aghazadeh, and Noah Marosok, I'm thankful for the fun and enjoyable times we had during lunch breaks, afternoon coffee, and official (or unofficial) ACES lab retreats.

My time in San Diego would not have been the same without friends who made life outside research colorful. I'm thankful to Mingjia Huo, Shiyang Jia, Hanwen Cao, Youwei Liang, Zhaoyi Liu, Yuexin Bian, and many others for the time we spent exploring new restaurants, watching sunsets at La Jolla beaches, going hiking, and traveling across the states.

Finally, and most importantly, I am profoundly grateful to my family, my parents and maternal grandparents, for their love, support, and care over the past 25 years. Raising a kid with strong opinions about nearly everything in her life cannot have been easy, yet they gave me abundant love, respect, patience, and the freedom to find my path. I am also thankful to Xiaochen Han for his encouragement and firm support through many difficult periods in my Ph.D.

The material in this dissertation is based on several published papers, as outlined below.

Chapter 1 in part contains in 1) Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar, "REMARK-LLM: A robust and efficient watermarking framework for generative large language models," in *USENIX Security Symposium*, 2024, 2) Ruisi Zhang, Yifei Zhao, Neusha Javidnia, Mengxin Zheng, and Farinaz Koushanfar, "AttestLLM: Efficient Attestation Framework for Billion-scale On-device LLMs," in *ACM/IEEE Design Automation Conference*, 2026, 3) Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar, "AutoMarks: A GNN-based Automated Physical Design Watermarking Framework," in *ACM Transactions on Design Automation of Electronic Systems*, 2025.

Chapter 2 is a reprint of the material as it appears in: Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar, "REMARK-LLM: A robust and efficient watermarking framework for generative large language models," in *USENIX Security Symposium*, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 3 contains material from work in submission, as it appears in: Ruisi Zhang, Neusha Javidnia, Nojan Sheybani, and Farinaz Koushanfar, "Robust and Secure Code Water-

marking for Large Language Models via ML/Crypto Codesign," *arXiv* 2025. The dissertation author was the primary investigator and author of this paper.

Chapter 4 is a reprint of the material as it appears in: Ruisi Zhang, and Farinaz Koushanfar, "EmMark: Robust Watermarks for IP Protection of Embedded Quantized Large Language Models," in *ACM/IEEE Design Automation Conference*, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 5 is a reprint of the material as it appears in: Ruisi Zhang, Yifei Zhao, Neusha Javidnia, Mengxin Zheng, and Farinaz Koushanfar, "AttestLLM: Efficient Attestation Framework for Billion-scale On-device LLMs," in *ACM/IEEE Design Automation Conference*, 2026. The dissertation author was the primary investigator and author of this paper.

Chapter 6 contains material from work in submission, as it appears in: Ruisi Zhang, Nasimeh Heydaribeni, and Farinaz Koushanfar, "OptiMarks: Tracing Constrained Combinatorial Solutions with Watermarking," in *arXiv*, 2026. The dissertation author was the primary investigator and author of this paper.

Chapter 7 is a reprint of the material as it appears in: 1) Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar, "ICMarks: A Robust Watermarking Framework for Integrated Circuit Physical Design IP Protection," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025; and 2) Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar, "AutoMarks: A GNN-based Automated Physical Design Watermarking Framework," in *ACM Transactions on Design Automation of Electronic Systems*, 2025. The dissertation author was the primary investigator and author of this paper.

VITA

2021	B.E. in Electronic Engineering, Shanghai Jiao Tong University
2023	M.S. in Electrical Engineering (Machine Learning and Data Science), University of California San Diego
2026	Ph.D. in Electrical Engineering (Machine Learning and Data Science), University of California San Diego

PUBLICATIONS

Nasimeh Heydaribeni, **Ruisi Zhang**, Tara Javidi, Cristina Nita-Rotaru, Farinaz Koushanfar, "SureFED: Robust Federated Learning via Uncertainty-Aware Inward and Outward Inspection," in *ICLR Workshop on Principled Design for Trustworthy AI*, 2026.

Neusha Javidnia, **Ruisi Zhang**, Ashish Kundu, Farinaz Koushanfar, "SWaRL: Safeguard Code Watermarking via Reinforcement Learning," *arXiv preprint arXiv:2601.02602*, 2026.

Ruisi Zhang, Yifei Zhao, Neusha Javidnia, Mengxin Zheng, Farinaz Koushanfar, "AttestLLM: Efficient Attestation Framework for Billion-scale On-device LLMs," in *ACM/IEEE Design Automation Conference*, 2026.

Yaman Jandali, **Ruisi Zhang**, Nojan Sheybani, Farinaz Koushanfar, "Optimizing Privacy-Preserving Primitives to Support LLM-Scale Applications," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025.

Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, Farinaz Koushanfar, "ICMarks: A Robust Watermarking Framework for Integrated Circuit Physical Design IP Protection," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

Ruisi Zhang, Neusha Javidnia, Nojan Sheybani, Farinaz Koushanfar, "Robust and Secure Code Watermarking for Large Language Models via ML/Crypto Codesign," in *arXiv preprint arXiv:2502.02068*, 2025.

Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, Farinaz Koushanfar, "AutoMarks: A GNN-based Automated Physical Design Watermarking Framework," in *ACM Transactions on Design Automation of Electronic Systems*, 2025.

Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, Farinaz Koushanfar, "Automated Physical Design Watermarking leveraging Graph Neural Networks," in *ACM/IEEE International Symposium on Machine Learning for CAD*, 2024.

Ruisi Zhang, Tianyu Liu, Will Feng, Andrew Gu, Sanket Purandare, Wanchao Liang, Francisco

Massa, "SimpleFSDP: Simpler Fully Sharded Data Parallel with torch.compile," in *arXiv preprint arXiv:2411.00284*, 2024.

Ruisi Zhang, Farinaz Koushanfar, "Watermarking Large Language Models and the Generated Content: Opportunities and Challenges," in *58th Asilomar Conference on Signals, Systems, and Computers*, 2024.

Nasimeh Heydaribeni, Xinrui Zhan, **Ruisi Zhang**, Tina Eliassi-Rad, Farinaz Koushanfar, "Distributed constrained combinatorial optimization leveraging hypergraph neural networks," in *Nature Machine Intelligence*, 2024.

Mingjia Huo, Sai Ashish Somayajula, Youwei Liang, **Ruisi Zhang**, Farinaz Koushanfar, Pengtao Xie, "Token-Specific Watermarking with Enhanced Detectability and Semantic Coherence for Large Language Models," in *International Conference on Machine Learning*, 2024.

Ruisi Zhang, Farinaz Koushanfar, "EmMark: Robust Watermarks for IP Protection of Embedded Quantized Large Language Models," in *ACM/IEEE Design Automation Conference*, 2024.

Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, Farinaz Koushanfar, "REMARK-LLM: A Robust and Efficient Watermarking Framework for Generative Large Language Models," in *USENIX Security Symposium*, 2024.

Ruisi Zhang, Shehzeen Samarah Hussain, Huili Chen, Mojan Javaheripi, Farinaz Koushanfar, "Systemization of Knowledge: Robust Deep Learning using Hardware-software Co-design in Centralized and Federated Settings," in *ACM Transactions on Design Automation of Electronic Systems*, 2023.

Ruisi Zhang, Mojan Javaheripi, Zahra Ghodsi, Amit Bleiweiss, Farinaz Koushanfar, "AdaGL: Adaptive Learning for Agile Distributed Training of Gigantic GNNs," in *ACM/IEEE Design Automation Conference*, 2023.

Ruisi Zhang, Seira Hidano, Farinaz Koushanfar, "Text Revealer: Private Text Reconstruction via Model Inversion Attacks against Transformers," in *arXiv preprint arXiv:2209.10505*, 2022.

ABSTRACT OF THE DISSERTATION

Algorithm and System Co-Design for Trustworthy and Secure AI

by

Ruisi Zhang

Doctor of Philosophy in Electrical Engineering (Machine Learning and Data Science)

University of California San Diego, 2026

Professor Farinaz Koushanfar, Chair

In the era of artificial intelligence (AI), foundation models have become engines for producing text, code, images, and complex design artifacts through human–AI interaction. This paradigm shift transforms not only how content is created but also changes the security landscape. As AI systems are increasingly deployed in critical domains such as software engineering, hardware design, scientific discovery, and digital media, ensuring the trustworthiness of both models and their generated artifacts becomes a fundamental challenge. Key concerns include ownership attribution, provenance tracking, authenticity verification, misuse prevention, and regulatory compliance across the lifecycle.

However, achieving trustworthy and secure AI at scale is inherently difficult due to the

massive volume of generated content, the need for real-time interaction, and the risk that artifacts can be adversarially modified and redistributed. Effective protection mechanisms must, therefore, be scalable, efficient, fidelity-preserving, and robust, while remaining seamlessly integrated into high-throughput AI systems.

My research addresses this pressing issue through a holistic, multi-level co-design of machine learning algorithms, security protocols, and system infrastructure. For **AI-generated content**, I developed **REMARK-LLM** and **RoSeMary** to enable provenance tracing and misuse identification for both natural language and programming code generated by cloud-hosted foundation models. These frameworks use learning-based watermarking techniques to embed ownership signals while preserving the semantics and utility of generated outputs, and support efficient and secure ownership verification through zero-knowledge proofs. For **on-device LLMs**, which are prone to being reverse-engineered in the supply chain, I proposed **EmMark** and **AttestLLM**, which jointly protect the software IP of edge-deployed models and the hardware IP of execution platforms by ensuring that only legitimate large language models can run on vendors' devices. These systems leverage adaptive watermarking algorithms with trusted execution environments to authenticate the execution of legitimate LLMs with efficiency, robustness, and practical deployability. For **human–AI collaborative creations**, I introduced **OptiMarks** to protect the IP of AI-assisted scientific discovery artifacts, and **ICMarks** and **AutoMarks** to protect the IP of AI-assisted chip design layouts. The protection mechanisms not only preserve design quality but also ensure automation and scalability as complex artifacts are generated at high volume and scale.

Chapter 1

Introduction

Artificial intelligence (AI) is capable of generating high-quality natural language responses, software code, scientific hypotheses, and engineering designs by learning from large-scale datasets and interacting with human instructions. As a result, generative AI systems have become integral to critical domains such as software development, hardware design, and digital media production. In this emerging ecosystem, the outputs of these foundation models can be easily copied, modified, or redistributed without proper attribution. Besides, the models themselves may be reverse-engineered, unauthorizedly reused, or deployed on illegitimate platforms. As generative AI systems become increasingly integrated into the digital world, reliable identification of AI-generated artifacts, protection of proprietary models, and traceability of generated content become essential.

Achieving these goals, however, is difficult due to the scale, throughput, and fragility of modern generative AI systems. Foundation models contain billions of parameters and produce content at massive volume and high frequency, making manual oversight infeasible. Moreover, generated artifacts can be paraphrased, refactored, or transformed while preserving their semantic or functional properties, making traditional identification strategies ineffective. These challenges require regulatory mechanisms that can embed ownership and provenance signals directly into generated artifacts while enabling efficient and robust verification in real-world settings.

My research addresses these trustworthiness challenges through a holistic co-design

of machine learning algorithms, security protocols, and system infrastructure, as summarized in Figure 1.1. **At the algorithmic level**, I design learning methods that automatically embed ownership and provenance signals into generated artifacts while preserving fidelity and functional robustness. **At the security protocol level**, I develop secure authentication mechanisms with low overhead by co-designing crypto-friendly ML algorithms and hardware-aware systems. **At the system level**, I build distributed and parallelized infrastructures that enable these protections to be efficiently deployed in real-world generative AI pipelines, from cloud-scale services to resource-constrained edge platforms. Through this unified co-design, trustworthy regulatory mechanisms can be made scalable, robust, and practical without sacrificing performance or user experience.

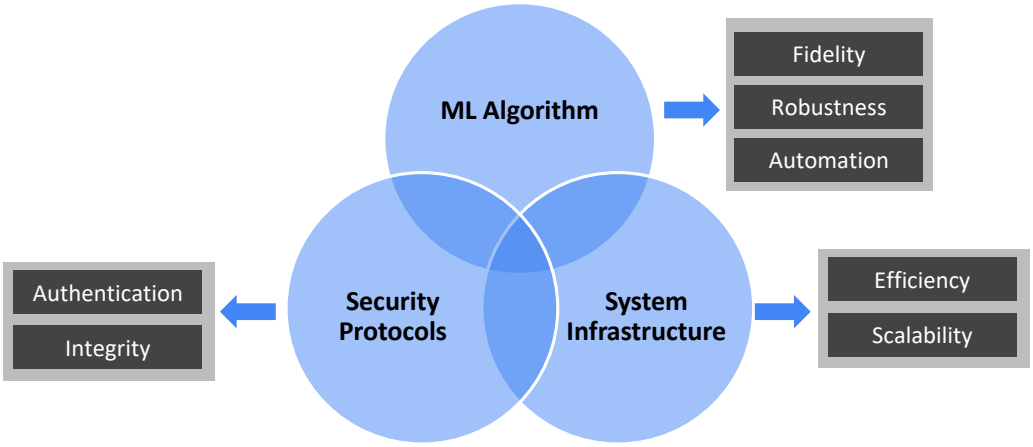


Figure 1.1. Research overview: My research leverages a holistic co-design of machine learning algorithms, security protocols, and system infrastructure to enable trustworthy and secure AI solutions.

1.1 Provenance Tracing for AI-Generated Content

Tracing the provenance and attributing the misuse of AI-generated outputs from cloud-hosted foundation models is generally done through watermarking, which involves embedding invisible identifiers into the output, and detecting them through decoding mechanisms. In large-scale GenAI services, content is produced at high volume and rapidly disseminated through

copy, repost, translation, or post-editing. These transformations can preserve semantics while destroying naive identifiers, making it difficult to enforce basic regulatory like tracing back to a specific model or provider scalably and robustly.

REMARK-LLM in Chapter 2 is designed to provide provenance for natural language generated from the cloud service. It embeds identifiable signals into generated text in a way that is difficult to remove without sacrificing utility and readability, while still enabling efficient watermark detection. REMARK-LLM adopts a learning-based approach that jointly trains the watermark encoder and decoder end-to-end. During training, it (i) preserves the semantics of watermarked text by minimizing a semantic cross-entropy loss objective, (ii) ensures reliable message recovery by minimizing the reconstruction loss between the embedded and decoded signatures, and (iii) improves robustness to adversarial editing by encouraging consistent decoding after adversarial transformations. REMARK-LLM demonstrates strong semantic preservation for text watermarking, high watermarking strength, and sufficient transferability across different LLM architectures and prompt sources.

RoSeMary in Chapter 3 extends provenance tracing to code generation, where regulatory requirements are stricter because watermarked programs must remain functional. To address this challenge, RoSeMary defines a set of semantics-preserving transformations, including syntactic transformations and variable renaming, that can be applied while keeping the code compilable. Similar to REMARK-LLM, RoSeMary jointly trains a watermark encoder and decoder end-to-end. The training objectives are designed to ensure: (i) functional correctness by minimizing the code feature mismatch between the original and watermarked code, (ii) accurate watermark detection by minimizing the message recovery loss, and (iii) robust signature recovery under malicious transformations. Besides, to prevent sensitive watermarks from being leaked to third parties during legal verification, RoSeMary incorporates zero-knowledge primitives to enable secure signature verification without revealing the signature itself and the decoding network parameters to third-party legal arbitrators. RoSeMary demonstrates strong preservation of watermarked code functionality and supports efficient non-interactive zero-knowledge proof

verification with minimal overhead.

1.2 Authentication for Edge AI Deployment

When foundation models are deployed on edge devices, the threat model changes substantially: end users and third parties along the supply chain may act as adversaries and gain direct access to model weights, execution traces, and the full inference stack. As such, I propose authentication mechanisms at two complementary model and hardware levels during edge deployment.

EmMark in Chapter 4 traces the origin and proves the ownership of edge-deployed models by embedding robust watermarks into quantized LLM weights while preserving model performance. During *watermark insertion*, EmMark encodes signatures into the quantized model using a novel parameter scoring function. This function ranks weight parameters according to two criteria: (i) fidelity preservation, measured by a parameter's sensitivity to watermark insertion, and (ii) robustness, achieved by embedding signatures in salient channels that are critical to model quality. Using these scores, EmMark randomly selects a subset of high-scoring candidate locations and embeds the watermark signature before deployment. When a suspect model is discovered, EmMark deterministically reconstructs the watermark locations using the random seed and watermark keys. The model owner then queries the suspect model, extracts the weights at the reconstructed locations, and decodes the embedded signature. Ownership is verified by comparing the decoded signature against the originally embedded one. Experimental results show that EmMark is resilient to various watermark removal and forgery attacks and preserves model quality across multiple quantization levels.

AttestLLM in Chapter 5 ensures that only authorized LLMs can execute on the target platform. AttestLLM adopts an algorithm/software/hardware co-design to embed robust watermark signatures into the activation distributions of LLM layers. AttestLLM embeds watermarks via gradient-based optimization before quantization and refines watermark embedding

post-quantization using zeroth-order optimization. To make verification efficient, AttestLLM further optimizes the attestation protocol inside a Trusted Execution Environment (TEE) by parallelizing watermark decoding and overlapping REE-TEE communication with computation, which achieves authentication without noticeable inference throughput reduction. Extensive evaluations on various on-device LLMs demonstrate that AttestLLM provides reliable attestation with strong fidelity and low overhead. Moreover, AttestLLM enforces model legitimacy and remains resilient to model replacement and forgery attacks.

1.3 Intellectual Property Protection for Human-AI Co-created Artifacts

I further extend generative AI regulation to high-value artifacts co-created by humans and AI, such as scientific solutions, optimization outputs, and chip design layouts. In these settings, regulation must ensure not only attribution but also quality preservation, including solution correctness, optimality, and manufacturability. These artifacts are highly structured (e.g., graphs or sequences of design decisions) and are produced through distributed pipelines and tools, making regulation and IP protection highly challenging.

OptiMarks in Chapter 6 targets IP protection for scientific discovery and combinatorial optimization artifacts. It embeds ownership information into solutions by introducing watermark constraints as an additional objective that the solver satisfies during optimization, thereby preserving the functionality and quality of watermarked solutions. Ownership can later be verified by mapping a suspect solution into the watermarking space and checking whether it includes the proprietor's signature. To enable reliable extraction under post-processing and adversarial changes, OptiMarks incorporates adversarial training and error-correcting codes for robust message recovery. For large-scale instances, OptiMarks further supports distributed watermarking by jointly embedding signature fragments across multiple servers in parallel. Extensive experiments across diverse constrained optimization benchmarks and application

scenarios demonstrate OptiMarks's transferability, robustness, and efficiency.

ICMARKS and **AUTOMARKS** in Chapter 7 protect the intellectual property of chip layout designs throughout the supply chain. I first present ICMARKS, which introduces a novel search function to adaptively identify suitable regions for embedding signatures in modern IC layouts that include macros and fence regions. The selected region, together with the corresponding cells, is then incorporated into the placement algorithm as a co-optimization term, which enables signature embedding while preserving design quality. To further improve efficiency, I propose AUTOMARKS, which uses graph neural networks to predict and search for high-quality watermark regions in parallel and reduces region search time by up to 50%. Extensive experiments on both wirelength-driven and timing-driven benchmarks show that ICMARKS and AUTOMARKS preserve layout quality while remaining robust against signature removal and forgery attacks.

1.4 Acknowledgements

This chapter is, in part, a reprint of the published material in 1) Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar, "REMARK-LLM: A robust and efficient watermarking framework for generative large language models," in *USENIX Security Symposium*, 2024, 2) Ruisi Zhang, Yifei Zhao, Neusha Javidnia, Mengxin Zheng, and Farinaz Koushanfar, "AttestLLM: Efficient Attestation Framework for Billion-scale On-device LLMs," in *ACM/IEEE Design Automation Conference*, 2026, 3) Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar, "AutoMarks: A GNN-based Automated Physical Design Watermarking Framework," in *ACM Transactions on Design Automation of Electronic Systems*, 2025.

Part I

Provenance and Ownership Tracing for AI-Generated Content

Chapter 2

Efficient Natural Language Watermarking with Learned Neural Networks

Recent advancements in the development of large language models (LLMs) such as ChatGPT [210], LLaMA [237], and GPT-4 [174] indicate a paradigm shift in human-computer dialogue interactions. These AI-powered systems have the capacity to generate human-like text responses and are integrated into various aspects of our daily lives. Training LLMs [22, 165, 176] requires substantial computational resources and extensive datasets, both of which are critical components of valuable intellectual property (IP). Concurrently, the increasing capabilities of these foundational models present potential risks in the form of malicious uses, including spam and plagiarism. Thus, there is a need to devise mechanisms to claim ownership of LLM-generated text and trace the distribution of the generated content.

Watermarking offers a promising solution to tackle two persistent issues: asserting ownership of generated output and tracing the source of content. By embedding watermark signatures into the outputs of LLMs, model proprietors can effectively monitor their content utilizations and validate their ownership. As such, the system can be applied to detect plagiarism to maintain academic integrity [107], protect the copyrights of LLM owners [120], and track the distribution of the potential misinformation generated by LLMs [154]. Existing literature on text watermarking can be classified into three categories [229]: (1) rule-based watermarking [101,

164], (2) inference-time watermarking [103, 129], and (3) neural-based watermarking [1]. The rule-based watermarking replaces synonym [98] or transforms syntactic structures [28] in the paragraph to insert as watermarks. Such manually designed features make the inserted signatures statistically removable through word distribution or syntactical analysis. The inference-time watermarking [103] splits vocabulary into green/red lists and restricts the LLM decoding to predict the next tokens from the green list. While the inserted watermarks are robust against attacks, the decoding strategy drastically distorts the semantic similarity between the watermarked and original LLM outputs. The neural-based approach [1] leverages an end-to-end learning technique to integrate the binary watermarking signatures into the LLM-generated texts while maintaining semantic coherence. However, the maximum encodable signature length per token segment is limited compared with the rule-based and inference-time frameworks, thus hindering the practical usage of this approach.

Generally speaking, watermarking text data presents several challenges. First, text data exhibits a pronounced sparsity compared with other modalities, such as images and audio. For instance, a 256-pixel image offers approximately 65k feasible positions for watermark insertion [167], whereas the maximum token limit in GPT-4 [175] is 8.2k. Besides, text data is fragile in that subtle alterations may obfuscate or compromise the semiotic fidelity [274], whereas minor perturbations in images can remain imperceptible. Note that all inserted watermarks, including the ones applied to LLM-generated text, should be resilient to potential watermark removal and detection attacks from end users [102].

In this chapter, I propose REMARK-LLM, a new robust and efficient watermarking framework to insert watermarks into LLM-generated texts without compromising the semantic integrity. REMARK-LLM composed of three key modules, namely, message encoding, reparameterization, and message decoding. The message encoding module encodes the LLM-generated texts and their corresponding signatures into latent feature space. Their feature representations are added and yield the watermarked distribution over the vocabulary. Sequentially, the reparameterization component exploits Gumbel-Softmax methodology [88] to transform the

watermarked distribution to the sparse distribution of the watermarked textual tokens. Next, the message decoding module extracts watermarking signatures by leveraging a transformer to predict the inserted messages. REMARK-LLM enhances its robustness by incorporating malicious transformations during training, including text addition, deletion, and substitution over the transformed textual token distribution into the message decoding phase.

The three modules are trained end-to-end, targeting to (1) preserve the semantic fidelity by minimizing a semantic loss between the original LLM-generated and watermarked texts, (2) ensure watermark extraction by minimizing a message recovery loss between the inserted and extracted watermarking signatures from the watermarked texts, and (3) enhance robustness by extracting watermarking signatures from the malicious transformations.

With the trained REMARK-LLM, the LLM proprietor leverages the message encoding module to embed binary signatures into the LLM-generated texts and obtain a watermarked distribution. An optimized beam search algorithm subsequently translates the output of this module's distribution into watermarked texts, ensuring linguistic coherence, unwavering semantic fidelity, and the successful extraction of signatures. Next, the watermarked texts are disseminated to end-users as coherent responses. The watermark existence can be verified by extracting the inserted signatures using the message decoding module. It compares the extracted messages with the inserted signatures to determine if the LLM generates the texts. In summary, the contributions are as follows:

- We introduce REMARK-LLM, a robust and efficient watermarking framework tailored for LLMs that maintains the semantic integrity of watermarked text while exhibiting resilience against potential watermark detection and removal attacks.

- We devise the watermarking framework with novelties lie in (i) a pre-trained sequence-to-sequence backbone module to significantly improve the transferability and capacity of the watermarking framework; (ii) an optimizing beam search module for balanced readability and extraction accuracy; (iii) incorporate potential malicious transformation

into the training for improved robustness against watermark removal attacks.

- We validate the effectiveness and robustness of REMARK-LLM by conducting extensive evaluations on multiple datasets: (i) REMARK-LLM can successfully embed 2× more signatures into LLM-generated text compared to prior art AWT [1] within 1.5 seconds; (ii) REMARK-LLM maintains the LLM-generated texts' semantics with an average of 0.90 BERT score and exhibits transferability to watermark natural language from unseen sources without extra fine-tuning; (iii) REMARK-LLM is resilient under various watermark detection and removal attacks and maintains an average AUC of 0.85.

2.1 Background and Related Work on Natural Language Watermarking

In this section, we first introduce the background and related work for LLM watermarking. Then, we compare the capabilities of those watermarking techniques.

2.1.1 LLM Watermarking

Adding post-hoc watermarks in LLM-generated texts can be methodologically categorized into [229]: (1) Rule-based watermarking, (2) Inference-time Watermarking, and (3) Neural-based Watermarking.

Rule-based watermarking This approach integrates watermarks into LLM-generated texts by manipulating linguistic features [74, 260], altering lexical properties [75], and substituting synonyms [164, 256]. The rule-based watermarking approach aims to insert the synonym replacement or syntactic transformations as watermarks while ensuring the overall semantics are not distorted.

Inference-time Watermarking Inference-time watermarking [103, 102] approach inserts signatures at the LLM decoding stage. This approach divides the vocabulary into red/green lists and only allows LLM to decode tokens from the green list. Some follow-up works [38, 273] proposed different red/green list splitting algorithms or sampling algorithms from the green

list probabilistic distribution to enhance the explainability and robustness of the inference-time watermarking.

Neural-based Watermarking A neural based approach [1] encodes LLM-generated texts and associated message signatures through an end-to-end machine learning paradigm. It leverages a data-hiding network to infuse the watermark signatures into the LLM-generated texts and a data-revealing network to decode the signature from the watermark text. This facilitates the signatures to be integrated into the feature space of the watermarked text without compromising the semantic fidelity. However, a notable limitation of current state-of-the-art neural-based approach AWT [1] is the limited embeddable signature length capacity [229].

2.1.2 Comparison

An ideal text watermarking framework should adhere to the following criteria:

• **Criteria 1 Effectiveness**: The inserted watermark signatures can be seamlessly extracted.

• **Criteria 2 Fidelity**: The watermarked content quality shall not be compromised. This entails that signature insertion not only preserves the original semantics but also ensures that the text coherence and consistency remain undistorted.

• **Criteria 3 Efficiency**: The watermark insertion and extraction are efficient. This includes both minimal computation and time overheads to ensure rapid IP insertion/verification without excessive computational resources.

• **Criteria 4 Robustness**: Resilience against potential threats is crucial to help LLM proprietors verify IP and trace data sources. Therefore, the signatures shall remain extractable under watermark detection and removal attacks.

• **Criteria 5 Undetectability**: The watermarks are invisible upon inspection. As a result, the adversary cannot detect whether a given text is watermarked.

We systematically evaluate the capabilities of the watermarking frameworks previously against the proposed criteria in Table 2.1. The rule-based watermarking, like CATER [74], demonstrates effectiveness and efficiency by inserting watermarks in the linguistic attributes of

the texts. However, the adversary may exploit syntactic transformations or synonym replacements to remove manually designed watermarking signatures from CATER [74]. The inference-time watermarking achieves resilience by embedding watermarks during each token decoding. However, KGW [103] introduces semantic discrepancies between the watermarked and original texts, undermining the LLM's fidelity. While EXP [106] tries to improve the semantic preservation, the efficiency is compromised compared to KGW [103]. The neural-based approach like AWT [1] leverages machine learning algorithms to embed watermarks into the LLM-generated texts without tampering with textual semantics. It achieves efficiency and robustness by embedding the watermarks through text feature space via lightweight language models.

To take the best properties of the neural-based watermarking, we devise REMARK-LLM as a pioneering LLM-generated text watermarking methodology. It can embed up to $2\times$ longer signature sequences into the same contents compared with the best prior art [1] without compromising the textual semantics and coherence.

Table 2.1. Comparison of post-hoc LLM-generated text watermarking schemes.

Method	Effectiveness	Fidelity	Efficiency	Robustness	Undetectability
Rule-based [74, 260]	✓	✓	✓	✗	✓
Inference-time [103, 106]	✓	✗	✗	✓	✓
Neural-based [1]	✓	✓	✓	✓	✓

2.2 REMARK-LLM Design

2.2.1 Problem Formulation

Watermarking Objective

Given the increasing popularity of LLMs and the heightened prominence of machine-generated media, the dangers associated with the content they produce have become more significant. Content generated by LLMs may inadvertently be employed to create counterfeit

essays or inundate the internet with spam responses, thereby posing a threat to the credibility of online content. Recognizing this challenge, REMARK-LLM equips LLM proprietors with robust IP tracing toolsets as shown in Figure 2.1. The local users submit prompt requests to a remote cloud-hosted LLM API to obtain responses. The LLM inserts watermarks into their generated text response before sending it to local users. An LLM proprietor can trace malicious usages online and claim ownership by applying message decoding modules to extract the signatures. LLM proprietors can claim ownership and prove whether texts are machine-generated using REMARK-LLM by comparing the inserted and extracted signatures.

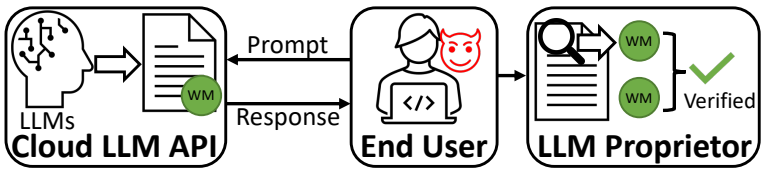


Figure 2.1. LLM-generated text watermarking scenario. The local user sends prompts to the remote LLM cloud API, and the API watermarks (WM) the responded texts before sending them back to users. LLM proprietor claims ownership by using the message decoding module to decode the signatures and compare them with inserted watermarks.

Applications: The watermarking system has wide applications with the emerging popularity of large language models: (i) Education [107]: Watermarking can help teachers and professors identify whether student homework submissions, like essays or research papers, are AI-generated. This maintains academic integrity and ensures students engage in original thinking and writing. (ii) Copyright Protection [120]: Watermarking can help detect if humans or AI write the given article. It protects the LLM owners' copyrights because the article publishers could make profits from the AI-generated content without proper acknowledgment. (iii) Misinformation Monitor [154]: Watermarking can be used by social media platforms to detect and label watermarked AI-generated content automatically. It helps to combat the potential spread of misinformation or inauthentic content.

Challenges

When compared to other methods of watermarking various media [186, 264, 33, 45], the process of embedding watermarks into text data presents distinct challenges, as outlined below.

• **Challenges 1 Sparsity**: A given image with size 256×256 provides over 65,000 potential pixel positions for embedding watermarks [167]. This granularity ensures notable flexibility in watermark accommodation. However, the maximum token length LLMs like GPT-4 can generate for text data is 8.2k [175]. This significantly constrains the potential embedding locations and demands a more sophisticated-designed watermarking approach.

• **Challenges 2 Sensitivity**: Text data exhibits a heightened sensitivity to alterations [274]. Minor image pixel adjustments often remain imperceptible, ensuring the image's aesthetic after watermark embedding. Text data, on the other hand, minor changes can distort the intended meanings and make the text incoherent or misleading.

• **Challenges 3 Vulnerability**: If an adversary suspects or detects watermarking, they might attempt to remove or alter the inserted signatures by rephrasing or editing attacks [102].

Threat Model

Adversary's Capacity: We assume the adversary is an end-user of the LLM cloud service, where he has black access to the API. However, he does not have access to the trained watermarking models and the original LLM-generated outputs. The adversary attempts to exploit the LLM-generated content for malicious usage without being traced. Therefore, he performs attacks to detect and remove the signatures within the watermarked contents without distorting semantics.

This threat model setting is consistent with prior work AWT [1] and EXP [106] that assume: (i) the watermarking framework is kept private, where adversaries as end-users do not have control over REMARK-LLM's weights and/or hyperparameters; (2) the adversarial attacks do not greatly compromise the generated output quality, accuracy, and readability.

Potential Attacks: The adversary performs detection attacks to inspect if the LLM-generated contents have watermark insertion or not. If the adversary suspects or detects watermarks, he performs attacks to remove inserted signatures by directly manipulating the textual content or leveraging sophisticated NLP models for rephrasing.

• **Attack 1 Watermark Detection Attack**: The adversary uses statistical analysis or machine learning models to detect whether the texts are watermarked.

• **Attack 2 Text Edit Attack**: The adversary doesn't have prior linguistic knowledge. By randomly deleting, adding, or substituting words within the content, he attempts to destroy the watermark while preserving the overall meanings.

• **Attack 3 Text Rephrase Attack**: The adversary can exploit open-source NLP models, such as T5, to remove watermarks. By feeding the content into these models, the adversary aims to generate a rephrased version of the original texts to remove the watermark.

• **Attack 4 Re-watermarking Attack**: The adversary dispatches the watermarked texts into another LLM watermarking framework like REMARK-LLM and re-watermark it to remove the inserted signatures.

2.2.2 REMARK-LLM Training

The watermarking framework is depicted in Figure 2.2, which is a confluence of three major components: **Message Encoding**, **Reparameterization**, and **Message Decoding**. The message encoding inserts invisible watermarks into the LLM-generated texts via a Sequence-to-sequence (Seq2Seq) model. Reparameterization converts the watermarked distribution from the Seq2Seq model towards a more sparse distribution of the watermarked textual tokens using Gumbel-Softmax. Message decoding first maps the reparameterized distribution into their respective embedding representation using a mapper network. It then employs a transformer-based decoder to extract the secret messages from the embedding.

Message Encoding: The message encoding module takes an LLM-generated token sequence, denoted as $T = \{T_1, T_2, ...T_t\}$, alongside a binary signature sequence M as input. The texts

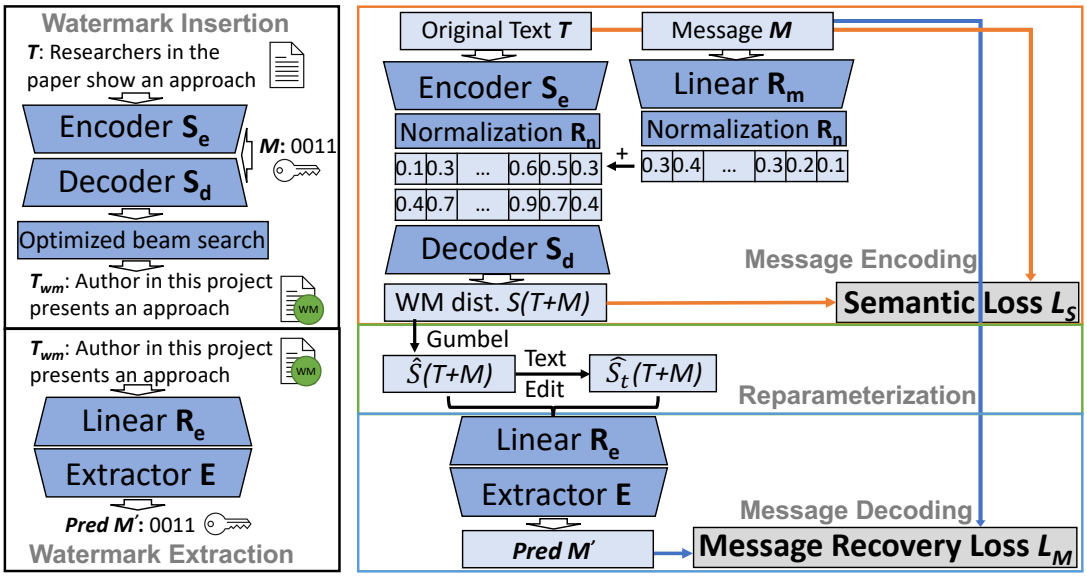


Figure 2.2. REMARK-LLM's watermarking framework. The left is an overview of REMARK-LLM: The message encoding module leverages an optimized beam search algorithm to produce coherent watermarked contents. The message decoding module is designed for efficient watermark extraction. The right is REMARK-LLM's training pipeline: The message encoding, reparametrization, and message decoding modules are trained jointly in an end-to-end fashion, aiming to minimize the semantic loss between original text T and watermarked distribution $S(T+M)$, as well as minimize the message recovery loss between the inserted message M and predicted message M'.

T goes through the encoder $\mathbf{S}_e$ of the Seq2Seq model and acquires its corresponding latent space representation $\mathscr{S}_e(T)$ at the final normalization layer $\mathbf{R}_n$. Concurrently, the message M is encoded by a linear layer $\mathbf{R}_m$ followed by the shared normalization layer $\mathbf{R}_n$ into the same latent space representation as $R_n(M)$. At the latent space, the messages are embedded into every token in T as $\mathscr{S}_e(T+M)$. The embedded latent feature $\mathscr{S}_e(T+M)$ is directed to the Seq2Seq's decoder $\mathbf{S}_d$ to obtain the watermarked distribution over the vocabulary as $\mathscr{S}(T+M)$.

Reparameterization: The message encoding module generates a dense token distribution, whereas the message decoding module extracts messages from the watermarked textual tokens' one-hot encoding. To bridge this gap, the reparameterization module connects them and transforms the dense token distribution into a sparser form while ensuring differentiability. To achieve this transformation, Gumbel-Softmax is applied in Equation 2.1 to approximate the wa-

termarked distribution $\mathscr{S}(T+M)$ to more sparse encoding, denoted as $\hat{\mathscr{S}}(T+M)$. $\mathscr{S}(T+M)$ is simplified as $\mathscr{S}$. The g_i is the noise i.i.d samples drawn from Gumbel(0,1), $|V|$ is the vocabulary size, and τ is the temperature for sampling. The lower τ is, the closer the reparameterized $\hat{\mathscr{S}}_i$ is to one-hot encoding.

$$\hat{\mathscr{S}}_i = \frac{\exp((\log(\mathscr{S}_i)+g_i)/\tau)}{\sum_{j=1}^{|V|}\exp\big((\log(\mathscr{S}_j)+g_j\big)/\tau)} \quad \text{for } i = 1,\ldots,|T| \tag{2.1}$$

Message Decoding: To decode embedded M from the reparameterized distribution $\hat{\mathscr{S}}(T+M)$, REMARK-LLM first maps it into the embedding space using a linear layer $\mathbf{R}_e$, yielding $\mathscr{H}(T+M)$. The $\mathscr{H}(T+M)$ is the watermarked text representation in the embedding space. Then, the transformer-based extractor model $\mathbf{E}$ extracts messages from $\mathscr{H}(T+M)$ as $M' = \mathscr{E}(\mathscr{H}(T+M))$.

REMARK-LLM becomes robust by enforcing the extractor to learn the embeddings of the malicious transformations and decode the same messages M from those transformations as well. The transforms, including randomly dropping, adding, and replacing tokens in the watermarked distribution, are performed over $\hat{\mathscr{S}}(T+M)$ and get their corresponding distribution as $\hat{\mathscr{S}}_t(T+M)$. Similar to $\hat{\mathscr{S}}(T+M)$, the $\hat{\mathscr{S}}_t(T+M)$ is mapped to the embedding space and extracts messages as $M'_t = \mathscr{E}(\mathscr{H}_t(T+M))$.

End-to-end Training: The above modules are trained in an end-to-end manner, with objectives to (1) ensure the semantic similarity of the input text T and the watermarked distribution $S(T+M)$ and (2) ensure the watermark extraction of the input message M and decoded message M' and M'_t. The first objective is reflected by the semantic loss L_s and the second is reflected by the message recovery loss L_M.

(1) Semantic Loss: REMARK-LLM formulates the semantic loss L_S by minimizing the cross entropy loss between input token T and watermarked text distribution $\mathscr{S}(T+M)$ in Equation 2.2. To avoid overfitting, in every epoch, the input token sequence T is randomly masked via a mask sequence T_M. T_M is of the same size as T, where 1 means the token is

unmasked and 0 means the token is masked. $|V|$ is the size of vocabulary in $\mathscr{S}$ and $|T|$ is the number of tokens in the input text T.

$$L_S(T, \mathscr{S}(T \cdot T_M + M)) = -\frac{1}{|T|} \sum_{i=1}^{|T|} \sum_{j=1}^{|V|} T_{ij} \log(\mathscr{S}_{ij}(T \cdot T_M + M)) \tag{2.2}$$

(2) Message Recovery Loss: REMARK-LLM measures the message recovery loss L_M between the input message M and decoded message M' from the watermarked distribution using L_1 loss. Similarly, REMARK-LLM also includes the message recovery loss between the signatures decoded from malicious transformation M'_t and input message M. In Equation 2.3, the two losses are adjusted by the coefficients w_w and w_t.

$$L_M(M, M', M'_t) = w_w \sum_{i=1}^{|M|} |M_i - M'| + w_t \sum_{i=1}^{|M|} |M_i - M'_t| \tag{2.3}$$

(3) Training: When training the end-to-end framework, we include the above losses together as an objective function in Equation 2.4. This is reflected in REMARK-LLM, where (i) minimizing $\mathscr{L}_S$ means encouraging the watermarked texts to be semantically close to the input texts, and (ii) minimizing $\mathscr{L}_M$ to ensure the encoded messages can be successfully extracted from the watermarked texts. The w_1 and w_2 are the trade-off coefficients during training.

$$L = w_1 L_S + w_2 L_M \tag{2.4}$$

2.2.3 Watermark Insertion

With a trained REMARK-LLM, model proprietors can use our message encoding module to insert watermarks into the LLM-generated contents. The message encoding takes the LLM-generated text $\bar{T}$ and the message $\bar{M}$ as input and generates the watermarked distribution over the vocabulary as $\mathscr{S}(\bar{T} + \bar{M})$. Decoding the distribution $\mathscr{S}(\bar{T} + \bar{M})$ by simply taking its maximum index without considering the overall sentence structure will diminish the text coherence. To overcome this, an optimized beam search algorithm is introduced in Algorithm 1. It aims to

ensure coherence while maximizing the watermark extraction rates.

Algorithm 1: Optimized Beam Search for Watermark Insertion

Input: LLM-generated token sequence $\bar{T}$; temperature schedule $\{\tau_k\}_{k=1}^{K}$; beam size B; number of iterations K; watermark message $\bar{M}$

Output: Watermarked text $\bar{T}_{wm}$

```
max_accuracy ← 0;
T̄_wm ← None;
for k ← 1 to K do
    Initialize token mask T̄_M;
    Compute watermark-conditioned distribution 𝒮(T̄ · T̄_M + M̄);
    foreach 𝒮_i ∈ 𝒮(T̄ · T̄_M + M̄) do
        𝒮_noisy,i ← 𝒮_i + Gumbel(𝒮_i, τ_k);
    T_k ← BeamSearch(𝒮_noisy, B);
    foreach T_{k,i} ∈ T_k do
        a ← Accuracy(E(T_{k,i}), M̄);
        if a > max_accuracy then
            max_accuracy ← a;
            T̄_wm ← T_{k,i};
return T̄_wm
```

In every decoding step, a Gumbel-Softmax noise with temperature τ is added into the token distribution $\mathscr{S}_i$. Then, the beam search algorithm with beam size B produces B candidate sentences from the perturbed token distribution. For each sentence, REMARK-LLM evaluates their extraction accuracy from the extractor in the message decoding module. Based on the empirical evidence, we find a small beam size B results in readable watermarked texts, whereas the best-accuracy sentence has good watermark extractability The beam search is repeated for K iterations with different temperatures τ_k to obtain more diverse watermarked texts.

2.2.4 Watermark Extraction

REMARK-LLM extracts the watermark via the message decoding module. Given the watermarked text $\bar{T}_{wm}$, it is first mapped into the embedding space using $\mathbf{R}_e$. Then, $\mathbf{E}$ extracts the predicted message $\bar{M}'$ from $\bar{T}_{wm}$ and compares it with LLM proprietor inserted watermark $\bar{M}$ to claim ownership.

Watermark Strength The confidence in predicting if watermark signatures reside in the watermarked texts can be evaluated using the z-score. The larger the z-score is, the more robust protection the watermark can provide. Given a message sequence with length $|M|$, $|N|$ bits out of the message can be successfully detected. The message generation is random and follows binomial distributions as in AWT [1], where the probability for generating bit 0 is $p = 0.5$ and bit 1 is $1 - p = 0.5$. The mean of the message distribution can be calculated as $\mu = |M| \times p$, and the variance can be calculated as $\sigma^2 = |M| \times p \times (1 - p)$. We calculate the z-score of the binominal distribution in Equation 2.5.

$$z = \frac{|N| - \mu}{\sigma} \tag{2.5}$$

2.3 REMARK-LLM Evaluations and Results

In this section, we first introduce the experiment setup. Then, we demonstrate REMARK-LLM's effectiveness, efficiency, and transferability compared with prior arts. Next, we present an ablation study on the effectiveness of each component in REMARK-LLM. Finally, we analyze REMARK-LLM's robustness by evaluating its performance under a spectrum of watermark removal and detection attacks.

2.3.1 Experimental Setups

We first introduce the infrastructure for training REMARK-LLM. Then, we include REMARK-LLM's detailed architecture configurations and the REMARK-LLM's hyperparameters at both training and inference time.

Hardware Infrastructure Our code is implemented using PyTorch [235] version 1.9.0 and open-source transformer models from huggingface [233]. The training and inference of our watermarking models are performed on NVIDIA RTX A6000 GPUs with Ubuntu 20.04.5 LTS and AMD Ryzen Threadripper 3990X 64-Core Processors.

REMARK-LLM Architecture We include REMARK-LLM architecture details in Table 2.2. Our T5 model [190] is initialized using the official pre-trained checkpoints from huggingface [233]. The rest of the submodule models are trained from scratch. For the transformer-based models, *n_head* is the number of heads for attention modules, *N* is the number of layers, and *d_model* is the feedforward dimension.

Table 2.2. REMARK-LLM architecture details. ML is short for Message Length, TL is short for Token Length, ED is short for Tokenizer Embedding Dimension

SubModule	Backbones	Input Size	Output Size
Message $\mathbf{R}_m$	Linear	ML	512
Mapping $\mathbf{R}_e$	Linear	TL	ED
SubModule	**Backbones**	***n_head***	***N***	***d_model***
Seq2Seq $\mathbf{S}_e$	T5 Encoder	12	12	768
Seq2Seq $\mathbf{S}_d$	T5 Decoder	12	12	768
Extractor $\mathbf{E}$	Transformer	8	3	512

Watermark Insertion and Extraction For the optimized beam search algorithm in the inference time, the detailed hyperparameters for watermark insertion are in Table 2.3. We choose the texts with the highest watermark extraction accuracy to be the watermarked texts for output. For the watermark extraction, we use the trained $\mathbf{R}_e$ to map the one-hot token distribution to the embedding space and extract the watermarks via the extractor $\mathbf{E}$.

Table 2.3. REMARK-LLM hyperparameters during training/inference.

Training-time	Settings
Epoch, Batch size	300, 16
w_w, w_t w_1, w_2	0.7, 0.3, 1, 1
Maximum Token Size	80
Optimizer, Learning rate	AdamW, 3e-5
Gumbel Temperature	0.3
Mask Percentage	50%

Inference-time	Settings
Beam Width, Repeat	5, 5
Gumbel Temperature	{1, 1, 1.5, 1.5, 2}
Mask Percentage	50%

Datasets We use four datasets to benchmark the LLM-generated content watermarking performance. The HC3 [71] is the ChatGPT-generated response to questions from QA platforms (e.g.,

Quora and Stack Overflow). The Human Abstract [170] and ChatGPT Abstract [170] are the research abstracts written by human researchers and their rephrased version by GPT-3.5 Turbo. The WikiText-2 [155] is a collection of paragraphs extracted from verified Good and Featured articles on Wikipedia. The detailed statistics are summarized in Table 2.4. We randomly split 80% of the texts as training datasets, and the remaining 20% are test datasets for HC3, ChatGPT Abstract, and Human Abstract.

Table 2.4. Dataset to benchmark the watermarking performance.

Dataset	Train	Test	Data Source
HC3 [71]	19440	4860	LLM
WikiText-2 [155]	44800	4360	Human
ChatGPT Abstract [170]	8000	2000	LLM
Human Abstract [170]	8000	2000	Human

Evaluation Metrics We benchmark **Effectiveness** by the fraction of the inserted watermarks successfully extracted, reflected by the Watermark Extraction Rate. **Fidelity** is measured by (1) BERT-S [270], indicating the watermarked texts should be semantically similar to the original texts, and (2) BLEU-4 [181] that the watermarked texts should be coherent and consistent w.r.t. the original texts. We compute these metrics using Huggingface Evaluate [233] and the measurement details are as follows:

- ***Watermark Extraction Rate (WER)***: the percentage of the binary watermark message successfully extracted.
- ***BERT-S*** [270]: the BERT score cosine distance between the original and watermarked text.
- ***BLEU-4*** [181]: the number of n-grams(n=4) in the watermarked text that match the reference texts.

BLEU-4 is adopted from machine translation to measure the n-gram(n=4) matches between the translated and baseline texts. While the metric has no hard threshold, baselines in

prior work [245] indicate BLEU-4 of higher than 0.15 is considered a semantically coherent document transformation. **Efficiency** of the watermarking frameworks are measured from two aspects: (1) the time overhead for inserting watermarks and (2) the computation resources required for inserting watermarks, as reflected by the peak GPU memory.

Baselines We compare REMARK-LLM with four state-of-the-art LLM watermarking frameworks: CATER [74], KGW [103], EXP [106] and AWT [1]. The rule-based watermarking algorithm CATER [74] inserts conditional watermarks into the LLM-generated texts. It inserts watermarks by choosing words that minimize the distortion of overall word distributions while maximizing the change of conditional word selections. The inference-time watermarking approach KGW [103] inserts watermarks at the LLM decoding step. By dynamically splitting the vocabulary into green/red lists with the message signatures, it enforces the next token prediction to only sample from the green list. We employ the soft red list watermarking algorithm in KGW [103]. The hyperparameters and the prompt methodologies follow the default settings. EXP [106] tries to reduce the semantic distortion by proposing an exponential minimum sampling strategy at the LLM decoding stage. The neural-based watermarking approach AWT watermarks LLM-generated texts in an end-to-end manner. It trains a transformer-based encoder-decoder network that takes an input sentence and a binary message to produce a watermarked text. AWT is trained to preserve the semantics of the watermarked texts while ensuring signature extraction.

For fair comparisons, we compare REMARK-LLM with baselines at the text segment level with a fixed 80 token length following EXP [106] and AWT [1]. We use the long text sequences with a fixed 640 token length as a proof-of-concept showing REMARK-LLM can watermark longer sequences, which exceeds the maximum length studied in prior work. KGW [103] and EXP [106] use OPT-2.7B [269] as the backbone generator to insert watermarks. For CATER [74], we employ their open-sourced synonym tables in follow-up work [76]. For KGW, we follow their paper setting [102] and consider the watermarking to be successful if the z-score of the watermark extraction exceeds 4. For EXP, the z-scores are below 4 after watermarking, and we report the WER providing the same level of p-value strength as EXP in Table 2.6.

2.3.2 Watermarking Performance

Segment-level Watermarking

We summarize the segment-level watermarking performance in Table 2.5, where the unit segment length is 80 tokens following AWT [1]. For LLM-generated texts, REMARK-LLM and AWT are trained on the HC3 training set and report the performance on the test set of HC3 and ChatGPT Abstract. For human-written texts, REMARK-LLM and AWT are trained on the WikiText-2 training set and report the performance on the test set of WikiText-2 and Human Abstract. For KGW, we use the first 40 tokens as prompts, and the predicted next 80 tokens are as watermarked texts. The inserted signature length is increased from 4-bit to 16-bit. All the original LLM-generated texts' WER are 50%, and we skip them in the table.

Table 2.5. Segment-level watermarking comparison. The length of the segment is 80 tokens. Both REMARK-LLM and AWT are trained on HC3 and WikiText-2's training dataset and report the watermarking performance on the test dataset. The transferability is benchmarked by reporting the test accuracy on ChatGPT Abstract with HC3-trained frameworks and on Human Abstract with WikiText-2-trained frameworks. The best metric values are highlighted in bold text, and the second best metric values are underlined. Metric values that are highlighted in red suggest failure cases (low WER or high semantic distortion). A WER of 50% indicates watermark recovery failure. The WER for all unwatermarked texts generated by the original LLM stands at 50%.

Dataset	Methods	4 bits			8 bits			16 bits		
		WER(%) ↑	BERT-S ↑	BLEU-4 ↑	WER(%) ↑	BERT-S ↑	BLEU-4 ↑	WER(%) ↑	BERT-S ↑	BLEU-4 ↑
HC3	REMARK-LLM	97.01	0.93	0.43	95.59	0.92	0.45	73.46	**0.92**	**0.46**
	AWT [1]	96.32	**0.94**	**0.91**	74.08	**0.96**	**0.84**	50.00	-	-
	KGW [103]	**99.43**	0.62	0.01	**99.43**	0.62	0.01	**99.43**	0.62	0.01
WikiText-2	REMARK-LLM	**97.23**	0.92	0.33	89.57	0.89	0.23	76.37	**0.89**	**0.19**
	AWT [1]	97.04	**0.95**	**0.92**	78.18	**0.96**	**0.92**	50.00	-	-
	KGW [103]	97.07	0.65	0.01	**97.07**	0.65	0.01	**97.07**	0.65	0.01
ChatGPT Abstract	REMARK-LLM	96.98	0.91	0.30	93.53	0.91	0.29	73.80	**0.90**	**0.24**
	AWT [1]	83.81	**0.94**	**0.96**	62.28	**0.96**	**0.83**	50.00	-	-
	KGW [103]	**99.87**	0.63	0.01	**99.87**	0.63	0.01	**99.87**	0.63	0.01
Human Abstract	REMARK-LLM	96.85	0.89	0.26	88.85	0.88	0.20	75.81	**0.84**	**0.10**
	AWT [1]	71.39	**0.95**	**0.95**	63.78	**0.95**	**0.85**	50.00	-	-
	KGW [103]	**99.50**	0.68	0.01	**99.50**	0.68	0.01	**99.50**	0.68	0.01

Comparison with AWT [1]: (1) REMARK-LLM can insert more signature bits into the same text length without compromising the semantics. The AWT's WER dropped to $\sim 70\%$ when inserting 8 bits signatures into the token sequence and failed to insert watermarks larger than

16 bits. However, REMARK-LLM can extract more than 90% of the signature when 8 bits are inserted. (2) AWT demonstrates worse transferability compared with REMARK-LLM. When 4 bits signatures are inserted into 80 tokens, the WER of AWT trained on WikiText-2 drops 25.65% when inference on the Human Abstract dataset and drops 12.51% from HC3 to ChatGPT Abstract dataset. For REMARK-LLM, the WER only drops 0.38% and 0.03% from WikiText-2 to Human Abstract and from HC3 to ChatGPT Abstract, respectively. (3) AWT achieves higher BLEU-4 by replacing words with their synonyms but does not modify the syntactic structure of the sentences. This hinders AWT from being robust under more powerful rephrase attacks, as shown in Section 5.4.5. By applying masking strategies during REMARK-LLM training, it achieves similar semantic preservation as AWT, but more diverse outputs as reflected by lower BLEU-4.

Comparison with KGW [103]: (1) KGW demonstrates better WER compared with REMARK-LLM by inserting watermarking at the LLM decoding step. However, this is at the cost of compromising its semantics and coherence. (2) KGW's average semantics is dropped by 29% compared with REMARK-LLM and KGW's coherence score BLEU-4 is close to 0. This demonstrates the KGW significantly distorts the semantics of the original LLM-generated texts and adversely compromises the LLM-generated content quality.

Watermarking Long Sequences

The responses local users receive from the LLMs are generally long text sequences with multiple paragraphs. In this section, we investigate how effectively long sequences can be watermarked with the aforementioned frameworks. For the long sequence watermarking, we set the maximum token length to 640 as a proof-of-concept showing REMARK-LLM can watermark longer sequences, which exceeds the maximum length studied in prior work. As AWT performs watermarking at the segment level, we report the watermarking performance by dividing the long sequence into multiple segments and watermarking each segment individually. We stop watermarking later segments if the maximum length is smaller than 640 tokens. We

train both AWT and REMARK-LLM on the HC3 dataset and report the inference watermarking performance on all four datasets with the trained framework. The results are reported in Table 2.6. The WER from unwatermarked content generated by LLMs is 50%. Thus, we skip it in the table.

Table 2.6. Long text sequence watermarking comparison. The length of the sequence is 640 tokens. The frameworks are trained on HC3's training dataset with the watermarking performance reported on the test dataset. The transferability is benchmarked by reporting the test accuracy on the rest of the datasets with the trained frameworks. The best metric values are highlighted in bold text, and the second best metric values are underlined. Metric values that are highlighted in red suggest failure cases (low WER or high semantic distortion). The WER for all unwatermarked texts generated by the original LLM stands at 50%.

Dataset	Methods	16 bits			32 bits			64 bits		
		WER(%) ↑	BERT-S ↑	BLEU-4 ↑	WER(%) ↑	BERT-S ↑	BLEU-4 ↑	WER(%) ↑	BERT-S ↑	BLEU-4 ↑
HC3	REMARK-LLM	98.93	0.94	0.45	97.84	0.92	0.41	95.61	0.91	0.41
	AWT [1]	96.12	**0.98**	**0.95**	94.51	**0.98**	**0.93**	68.42	**0.95**	**0.82**
	KGW [103]	**99.57**	0.58	0.01	**99.57**	0.58	0.01	**99.57**	0.58	0.01
	EXP [106]	79.37	0.80	0.01	70.62	0.80	0.01	64.68	0.80	0.01
	CATER [74]	75.20	0.96	0.63	75.20	0.96	0.63	75.20	0.96	0.63
WikiText-2	REMARK-LLM	99.02	0.92	0.32	98.60	0.86	0.18	94.48	0.85	0.16
	AWT [1]	89.72	**0.97**	**0.95**	85.82	**0.95**	**0.93**	65.77	**0.96**	**0.85**
	KGW [103]	**99.13**	0.61	0.02	**99.13**	0.61	0.02	**99.13**	0.61	0.02
	EXP [106]	79.37	0.82	0.01	70.62	0.82	0.01	64.68	0.82	0.01
	CATER [74]	53.10	0.94	0.75	53.10	0.94	0.75	53.10	0.94	0.75
ChatGPT Abstract	REMARK-LLM	98.24	0.92	0.33	98.55	0.90	0.27	95.04	0.89	0.27
	AWT [1]	88.26	**0.96**	**0.95**	80.62	**0.97**	**0.94**	62.39	**0.95**	**0.84**
	KGW [103]	**99.01**	0.61	0.01	**99.01**	0.61	0.01	**99.01**	0.61	0.01
	EXP [106]	79.37	0.80	0.01	70.62	0.80	0.01	64.68	0.80	0.01
	CATER [74]	75.50	0.93	0.64	75.50	0.93	0.64	75.50	0.93	0.64
Human Abstract	REMARK-LLM	98.56	0.91	0.31	98.71	0.88	0.16	95.39	0.87	0.15
	AWT [1]	86.43	**0.96**	**0.93**	77.21	**0.98**	**0.92**	63.52	**0.94**	**0.85**
	KGW [103]	**98.79**	0.69	0.01	**98.79**	0.69	0.01	**98.79**	0.69	0.01
	EXP [106]	79.37	0.81	0.01	70.62	0.81	0.01	64.68	0.81	0.01
	CATER [74]	82.00	0.95	0.54	82.00	0.95	0.54	82.00	0.95	0.54

Comparison with AWT [1]: (1) AWT successfully preserves the semantics and coherence of the LLM-generated contents. However, the WER drops by an average of 19.52% when the signature length is extended to 64 bits. Thereby, AWT's ability to embed stronger watermarks into the sequence is significantly hindered. (2) Similar to segment level watermarking, AWT exhibits worse transferability compared with REMARK-LLM. When inserting 32-bit signatures, AWT's WER dropped by 14% when watermarking the ChatGPT Abstract dataset using the watermarking model trained on HC3. However, the WER of REMARK-LLM from HC3 to ChatGPT Abstract showcased no accuracy drop.

Comparison with KGW [103]: (1) The semantic distortion introduced by KGW becomes

worse for longer text sequence. For the LLM-generated content, KGW's BERT-S drops by 35% compared with REMARK-LLM. This suggests that KGW could not preserve the original content quality generated by LLM, resulting in an ineffective watermark insertion. (2) Besides, the BLEU-4 of KGW is close to 0, which demonstrates KGW failed to maintain the coherence between the original LLM-generated and watermarked texts.

Comparison with EXP [106]: (1) While EXP maintains the semantics integrity for inference-time watermarking, the BERT-S and BLEU-4 are averagely 0.07 and 0.25 lower compared with 64-bit REMARK-LLM. This indicates neural-based REMARK-LLM preserve better semantics and coherence compared with EXP for longer text sequences. (2) EXP preserves the semantic at the cost of weaker watermark insertion. The average p-value EXP inserted into the watermarked texts is 9.9×10^{-3} and equals an average z-score of 2.36. However, the average z-score REMARK-LLM can provide at 64-bit signature is 7.12. Therefore, REMARK-LLM demonstrates better semantic preservation and stronger watermark insertion than EXP.

Comparison with CATER [74]: CATER achieves high semantic preservation by replacing words with their synonyms. However, such replacements are not generalizable toward new datasets because the candidate words can have high frequency on certain datasets but low frequency on the rest. As in Table 2.6, CATER has higher WER on HC3 and ChatGPT/Human Abstract datasets but low WER ($\sim$ 50%) on the WikiText-2 dataset. Note that the WERs of CATER are still 13% lower than REMARK-LLM even on the best-performing Human Abstract dataset.

2.3.3 Analysis and Visualizations

Watermarking Strength

The watermarking strength, measured by z-score, quantitatively evaluates the statistical significance and robustness of the watermarks embedded within the content generated by the LLM. Based on Null Hypothesis [11], a z-score threshold of 1.64 corresponds to a p-value of less than 0.05, implying a significant presence of the watermarks in the content. Following

KGW [103], a higher z-score of 4 ($p = 3 \times 10^{-5}$) indicates a greater alignment between the inserted and extracted signature bits. This alignment serves as stronger evidence that the watermarks belong to the owner of the LLM. We use the z-score from Section 2.2.4 to compute the watermarking strength for neural-based approach REMARK-LLM and AWT. The results are reported on the ChatGPT dataset with watermarking frameworks trained on HC3. The z-score of KGW [1] and EXP [2] is calculated using their original implementation.

From Figure 2.3, we find that (1) AWT, EXP, and CATER preserve the semantic integrity but fail to provide sufficient watermark strength for long text sequences. This makes these two approaches less suitable for real-world text ownership proof, where the adversary may argue the watermark insertion is coincidental. (2) While KGW effectively embeds watermarks to LLM-generated content, there is a noticeable semantic distortion and greatly degrading LLM quality. (3) REMARK-LLM demonstrates the merits of the aforementioned approaches and ensures both semantic fidelity and watermarking strength. The average z-score of REMARK-LLM is 7.12, corresponding to an average one-side p-value of 5.4×10^{-13}. This provides adequate watermarking proof for a text sequence with 640 tokens. When operating LLMs like GPT-4 at their zenith, the generated contents will be a maximum of 8.2k tokens. REMARK-LLM can provide a z-score of 25.20, corresponding to a one-tail p-value of smaller than 1.59×10^{-130}, all while maintaining semantic integrity. As a result, REMARK-LLM gives ample watermark strength to LLM proprietors.

Watermarking Efficiency

We measure the efficiency of inserting 8-bit watermarks into 100 samples of 80-token LLM-generated texts by reporting the average wall-clock time required in seconds and peak GPU memory utilization in GB. These results are summarized in Table 2.7. For REMARK-LLM and AWT, we report the inference time for the watermarking framework to generate the watermarked texts. For KGW and EXP, we report the time difference between the inference time with and

[1] https://github.com/jwkirchenbauer/lm-watermarking
[2] https://github.com/jthickstun/watermark

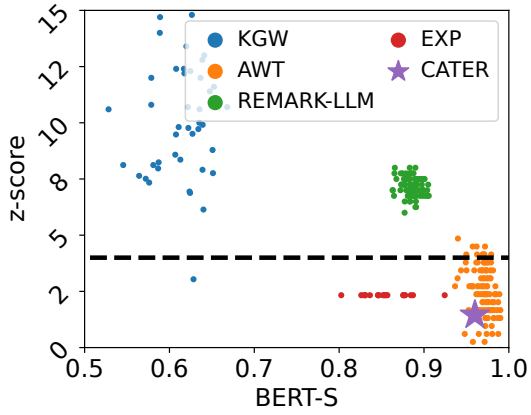


Figure 2.3. Watermarking strength and semantic preservation comparison of different watermarking frameworks. The threshold for a strong watermark insertion is a z-score of 4, represented as the black dotted line.

without watermark insertion.

The results indicate that inserting the same number of bit signatures into the LLM-generated texts, KGW requires nearly 2 $\times$ more time compared with neural-based REMARK-LLM and AWT. The additional overhead stems from the green/red list splitting over the vocabulary for every next token prediction. For EXP, which aims to improve the inference-time watermarking's semantic preservation, the time overhead becomes even larger, which takes 14$\times$ longer than REMARK-LLM to watermark the same texts. This greatly constraints such approaches from being widely applied in real-world LLM APIs.

For the computation resources, KGW and EXP require 2.1$\times$ more GPU memory compared with REMARK-LLM and 4.5$\times$ compared with AWT. The required computation resources for neural-based REMARK-LLM and AWT are agnostic to the LLM type, whereas KGW and EXP require more GPU memory as LLM size grows. Therefore, the neural-based approaches are more computationally efficient compared with inference-time frameworks. For the rule-based CATER, the textual transformations are performed on the CPU, and the GPU memory consumption is 0.

Table 2.7. Efficiency (time and memory) comparison among watermarking frameworks.

Methods	Time (s)	Memory(GB)
REMARK-LLM	1.21	5.83
AWT [1]	1.14	2.72
KGW [103]	2.36	12.21
EXP [106]	17.50	12.21
CATER [74]	1.02	0

Watermarking Integrity

We evaluate REMARK-LLM's watermarking integrity by running its message decoding module on non-watermarked texts. Then, we calculate the WERs by comparing the decoded signature with the randomly generated encoding signature. We benchmark the WERs on the four datasets' test set and report the results in Table 2.8. The close to 50% WER on non-watermarked texts demonstrates REMARK-LLM's integrity.

Table 2.8. Watermarking integrity on different datasets.

Datasets	HC3	WikiText-2	ChatGPT Abstract	Human Abstract
WER	49.9%	49.9%	50.0%	50.1%

Watermarking Different LLM Architectures

We have shown REMARK-LLM trained on the ChatGPT-generated HC3 dataset can successfully watermark human written texts and GPT-3.5 Turbo-generated ChatGPT Abstract in Section 2.5. We also investigate whether trained REMARK-LLM can watermark unseen texts generated by other LLM families in Table 2.9. We use 2k instruction prompts from Alpaca Dataset [231] and watermark responses from three state-of-the-art open-source LLMs: OPT [269], OpenOrca [163], and LLaMA-2 [237]. Those models are trained with different architectures and different training datasets. The prompts for generating those responses do not overlap with REMARK-LLM's training dataset HC3.

The watermarking is performed for long text sequences by using the instruction prompts as input and limiting the LLMs to predict the next 640 tokens. REMARK-LLM insert 64 bits into

their generated texts and report the watermarking performance in Table 2.9. We find REMARK-LLM can successfully insert and extract watermarks from texts generated by different LLM architectures. REMARK-LLM also keeps high-quality semantic preservation and coherence. Our results indicate that once REMARK-LLM is trained on the large LLM-generated corpus like HC3, the trained REMARK-LLM is agnostic to the LLM architectures and data sources. REMARK-LLM can thereby be used in real-world LLM watermarking scenarios.

Table 2.9. REMARK-LLM performance in watermarking texts generated by different LLM architectures.

LLMs	WER(%)	BERT-S	BLEU-4
OPT-2.7B [269]	93.42	0.91	0.34
OpenOrca-7B [163]	93.70	0.92	0.35
LLaMA-2-7B [237]	91.18	0.91	0.39

Watermarking Examples

We present the original LLM-generated and watermarked texts in Table 2.10 and highlight their differences. The first two texts are taken from the HC3 test set, and the last two are taken from the ChatGPT Abstract test set. All texts are embedded with 8-bit message signatures using REMARK-LLM trained on HC3's training set at the segment level. For all the watermarked messages presented in Table 2.10, the watermarks are successfully extracted with 100% WER.

From Table 2.10, we find that the watermarked texts are readable and possess the same semantics as the original texts. REMARK-LLM not only learns to replace the word with their synonyms but also adds/deletes words or replaces other non-synonyms to ensure coherence and preserve the semantic structure. For example, in the first example, "itself" is added after "space"; in the second example, "may" is replaced by "factors"; and in the third example, "by a suitable" is deleted. Those changes do not affect the overall quality of the watermarked texts but help to accommodate longer watermark signature sequences.

Table 2.10. Watermarked text examples. All of the watermarked texts achieve 100% WER. The first two examples are randomly taken from the HC3 test set, and the last two are randomly taken from the ChatGPT Abstract test set. The edited words are highlighted in yellow.

Original Text	Watermarked Text
It can be hard to explain it in simple terms. But I'll do my best! During inflation, the universe expands at an incredibly fast rate. But it's important to note that this expansion is not like the movement of objects through space.	It can be hard directly explain exactly in simple terms. But I'll try my best! During time, the universe expands at an infinite incredibly fast rate. But it's important to understand that this expansion is not about the movement of objects through space itself.
In the context of financial investments, "headwinds" refer to negative factors that can potentially hinder the performance of an investment. These may include economic conditions, regulatory changes, market trends, or other external factors that can work against the investment.	In the context of stocks investing, "headwinds" refer for negative factors that can potentially impact the value of an investment. These factors include economic impacts, regulatory issues, market conditions, and other external factors that work against the investment.
The paper discusses Colombeau's generalized function on arbitrary manifolds. We first define the space of Colombeau's generalized functions by quotienting out by a suitable ideal endowed with a ring structure.	The paper introduced Colombeau's generalization function on arbitrary manifolds. We first study the space of Colombeau's generalized functions by particle out ideal endowed with a ring structure.
This paper presents a novel methodology for constructing super throats using non trivial scalar fields. By introducing these fields, we are able to achieve unprecedented control over the dynamics of the throats.	This research presents a novel approach for constructing super throats through non trivial scalar fields. By utilizing these fields, we are able to obtain precise control over the dynamics of the throats.

2.3.4 Ablation Studies

In this section, we study how different hyperparameter choices affect the performance of REMARK-LLM during both training and inference time. We first analyze how different training coefficients affect the watermarking performance. The results in this section are all trained on the HC3 dataset with 8 bits inserted in an 80-token segment. The inference hyperparameters follow the default settings from Section 2.3.1 and are reported on the ChatGPT Abstract dataset to avoid overfitting.

Effectiveness of Watermark Backbones

We use the default training settings but change the Seq2Seq Model's backbone from T5-base to T5-small and T5-large. For the two replaced models, T5-small has 8 attention heads, 6 layers, and 512 feedforward dimensions, whereas T5-large has 16 attention heads, 24 layers, and 1024 feedforward dimensions.

From Table 2.11, as the backbone Seq2Seq model becomes larger, REMARK-LLM gets

better text coherence and better watermark extraction rates. This is because larger backbone models have more parameters to learn how to add watermarks in the feature space and achieve better performance. This improvement is significant when the model is switched from T5-small to T5-base, and is marginal when the model is changed from T5-base to T5-large.

Table 2.11. The effectiveness of different model backbones in REMARK-LLM performance.

Backbone	WER(%)	BERT-S	BLEU-4
T5-small	91.60	0.89	0.25
T5-base	93.53	0.91	0.29
T5-large	93.77	0.91	0.32

Effectiveness of Masking Percentages and Gumbel Temperatures

We compare how different masking percentages and the Gumbel-Softmax temperatures affect the overall text coherence and watermark extraction accuracy. The results are summarized in Table 2.12, where we change the input masking percentage from 30% to 70% and the Gumbel-Softmax temperatures from 0.1 to 0.5. We find that: (1) As the masking percentage becomes larger, at the same Gumbel-Softmax temperature, the coherence between LLM-generated texts and the original texts worsens, whereas the watermarking extraction rates become higher. Given that REMARK-LLM has more masked space, it introduces more alterations to the original texts, thereby can accommodate more signatures. (2) As the Gumbel-Softmax temperature becomes larger, at the same masking percentage, the coherence of the texts becomes worse, but more watermark signature bits are extracted from the watermarked texts. However, if the temperature rises to 0.5, REMARK-LLM fails to learn the one-hot encoding from Seq2Seq output and achieves worse watermark extractions. Therefore, we trade off the semantic preservation and the watermarking extraction rates during REMARK-LLM training. By default, we choose a masking percentage of 50% and a Gumbel-Softmax temperature of 0.3.

Table 2.12. The effectiveness of different masking percentages and gumbel noises in REMARK-LLM performance.

Masking	Temp	WER(%)	BERT-S	BLEU-4
30%	0.1	89.78	0.91	0.34
	0.3	89.97	0.91	0.31
	0.5	83.09	0.91	0.31
50%	0.1	91.18	0.91	0.31
	0.3	93.53	0.91	0.29
	0.5	85.43	0.90	0.29
70%	0.1	94.74	0.89	0.26
	0.3	94.98	0.87	0.17
	0.5	87.30	0.86	0.17

Effectiveness of Loss Coefficients

We include the watermarking performance under different loss coefficients in Table 2.13. The w_1 semantic loss coefficient increases from 0.3 to 0.7, whereas the w_2 message recovery loss coefficient decreases from 0.7 to 0.3. From here, we find that larger w_1 indicates better semantic coherence between the original and watermarked texts and a larger w_2 results in higher watermark extraction rates. Therefore, we choose $w_1 = 0.5$ and $w_2 = 0.5$ to obtain a balance between the text semantic coherence and watermark extractions.

Table 2.13. The effectiveness of different loss coefficients in REMARK-LLM performance.

w_1	w_2	WER(%)	BERT-S	BLEU-4
0.3	0.7	94.78	0.89	0.25
0.5	0.5	93.53	0.91	0.29
0.7	0.3	91.76	0.92	0.33

Next, we study how different coefficients in the inference time affect the REMARK-LLM performance. The training hyperparameters follow the default settings from Section 2.3.1. We change the token masking percentage in Algorithm 1. The results are summarized in Table 2.14, where we change the masking input percentage from 0.3 to 0.7.

From here, we find that as the masking percentage grows, REMARK-LLM tends to get better WER. However, the coherence and semantic preservation of the texts are compromised,

as reflected by the lower BLEU-4 and BERT-S. Therefore, we choose the masking percentage during inference to be 50% to ensure semantic preservation and high watermark extraction rates.

Table 2.14. The effectiveness of different inference hyperparameters in REMARK-LLM performance.

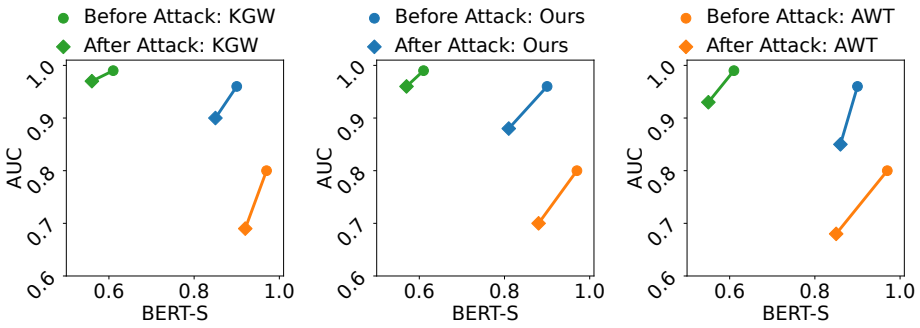
Masking	WER(%)	BERT-S	BLEU-4
30%	91.73	0.93	0.45
50%	93.53	0.91	0.29
70%	96.22	0.86	0.12

2.3.5 Attack Evaluations

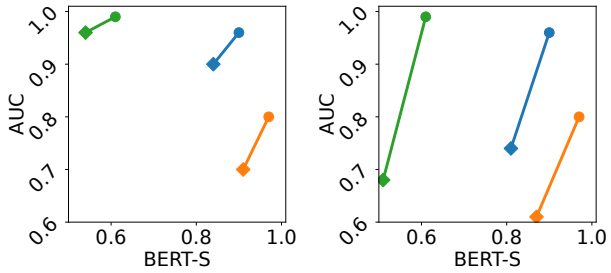
In this section, we evaluate the watermarking frameworks' **Robustness** under watermark removal attacks and **Undetectability** under watermark detection attacks.

We first demonstrate the resiliency of REMARK-LLM under watermark removal attacks. The adversary attempts to remove the signatures in the watermarked texts by performing the following attack: (1) **Text Deletion Attack**: randomly delete words in the watermarked texts with a probability of 6%; (2) **Text Addition Attack**: randomly add words in the watermarked texts with a probability of 6%; (3) **Text Replacement Attack**: randomly replace words with their synonyms by leveraging Word2Vec [156] pre-trained on Google News corpus [157] (3 billion running words); (4) **Text Rephrase Attack**: using open-source NLP tool T5-large [190] to rephrase the watermarked texts with prompt "paraphrase the texts:" and accept the rephrase if the BERT-S between the watermarked and rephrased sentence is higher than 0.85; and (5) **Re-watermark Attack**: training a local version of REMARK-LLM with WikiText-2 dataset, and re-watermark the texts with new signatures. We report REMARK-LLM's performance under the aforementioned attacks using REMARK-LLM trained on the HC3 dataset and report the attack performance on the ChatGPT dataset.

The adversary's goal is to remove the signature and maintain the semantics and readability of the resultant texts. Therefore, we do not consider the emojis or special characters attack



(a) Text Addition Attack **(b)** Text Replacement Attack **(c)** Text Rephrase Attack



(d) Text Deletion Attack **(e)** Re-watermark Attack

Figure 2.4. Watermarking performance under attacks, including watermark extraction measured by AUC and semantic coherence measured by BERT-S. The attacks are performed on the ChatGPT Abstract dataset with frameworks trained on the HC3 dataset. KGW [103] is the inference-time watermarking framework. AWT [1] is the neural-based watermark framework. From left to right, we study text edit attacks (deletion, addition, and replacement), text rephrase attacks and re-watermark attacks.

proposed in KGW [103], which adds additional symbols into the watermarked contents to compromise the overall text readability. Furthermore, the adversary uses LLM-generated texts to reduce the time for batch-generating spam content. Therefore, we do not consider the scenario where extra human labor is involved in the attack to rewrite the texts for watermark removal, which deviates from the adversary's initial objectives.

Text Addition Attack

From Figure 2.4a, we find (1) REMARK-LLM maintains high watermark extraction rates, yielding an average AUC of 0.90 and average BERT-S of 0.90 before attack. (2) Similar to the text deletion attack, AWT's average AUC degrades to 0.69, which is 23% lower compared

with REMARK-LLM and exhibits worse resilience toward text addition attack. (3) While KGW also demonstrates an average AUC of 0.97 after attacks, it fails to maintain semantic coherence in generated texts before and after attacks. This indicates that by including malicious transformations during training, REMARK-LLM becomes robust to unseen threats and maintains semantic coherence during watermarking.

Text Replacement Attack

The replacement attack poses a more significant threat to watermark extraction than the prior two approaches, as depicted in Figure 2.4b. (1) Firstly, AWT and REMARK-LLM watermarking extraction relies on combinations of textual tokens to decode the inserted messages from their respective feature space. By replacing words with synonyms, some of the features used for watermark extraction are compromised, thus resulting in worse AUC. However, the replacement attack does come with a cost of 4-5% BERT-S degradations. (2) REMARK-LLM maintains an AUC of 0.88 with a 0.90 BERT-S score, demonstrating its resilience under such attacks. In contrast, AWT's AUC degrades to 0.70, and KGW's BERT-S degrades from 0.61 to 0.57. This demonstrates that REMARK-LLM maintains a good trade-off between robustness and semantic preservation of watermarked content.

Text Rephrase Attack

Figure 2.4c demonstrates that text rephrase attack results in stronger WER degradation compared to text editing attacks. (1) Our findings show that REMARK-LLM remains resistant to rephrase attacks and consistently maintains an AUC of 0.85. This translates to an average z-score of 5.6 for long text sequences, highlighting the robustness of REMARK-LLM in establishing ownership proof. (2) Compared to baselines, KGW's maintains an AUC of 0.92, but its BERT-S drops to 0.61 even before the attack. (3) Conversely, while AWT preserves semantic information before attacks, its AUC drops to 0.68 after the attack.

Text Deletion Attack

From Figure 2.4d, we find that (1) REMARK-LLM achieves an AUC of 0.90 under Text Deletion Attack. This demonstrates that 90% of watermarking and non-watermarking texts can be successfully classified, providing sufficient ownership proof for LLM proprietors. (2) While KGW achieves 0.96 AUC, the semantic preservation evaluated by BERT-S even before the attack is 0.61. Therefore, KGW failed to provide effective watermarking to LLM-generated content. (3) AWT successfully maintains the semantics with BERT-S larger than 0.90, but the AUC drops to 0.70, demonstrating worse resilience towards text deletion attacks.

Re-watermark Attack

We highlight that re-watermarking performance in Figure 2.4e. Re-watermarking degrades the WER by rephrasing the text with REMARK-LLM and tries to break the text features used for watermark verification. Therefore, the WER for all three methods significantly degrade. We find that REMARK-LLM maintains an AUC of 0.74, whereas KGW degrades to 0.68 and AWT degrades to 0.61, demonstrating REMARK-LLM's robustness to re-watermarking attacks.

Next, we include more security evaluations on REMARK-LLM. Instead of removing the watermark, the adversary intends to know if watermarks are presented in specific texts. One of the most straightforward ways is to use the message decoding module to detect if the watermarks can be successfully extracted. However, when the adversary is a malicious end-user, he/she will not have access to REMARK-LLM's trained components. To detect the watermark presence, the adversary uses (1) statistical analysis of watermarked texts and (2) machine-learning models to classify watermarked and non-watermarked texts.

Statistical analysis

We show top-word distribution from the original LLM-generated texts and the watermarked texts in Figure 2.5. The texts are taken from the ChatGPT abstract dataset with REMARK-LLM train on HC3. From here, we can find the distributions are close, meaning the adversaries cannot distinguish if texts are watermarked or not by simply analyzing the word

frequency distributions.

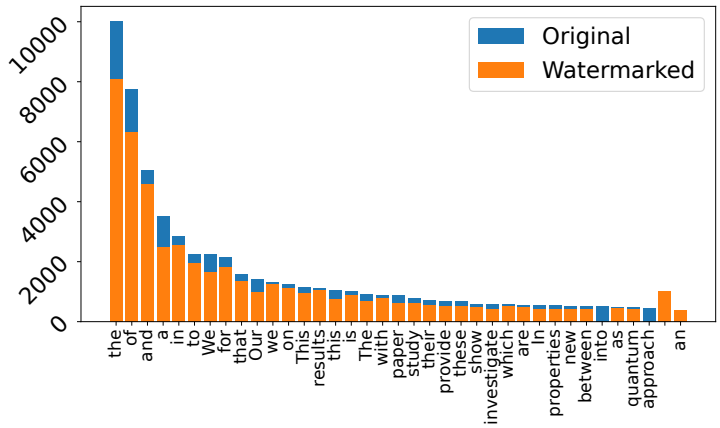


Figure 2.5. Word frequency distribution of original LLM-generated texts and corresponding watermarked texts.

Machine-learning classification

We use machine learning models to classify 2k samples with 1k watermarked and 1k LLM-generated texts. Those texts are non-paired, meaning the watermarked version of the LLM-generated texts is not shown in the training dataset. This is a common setting in which the adversary cannot access original LLM-generated texts. The adversary can only request the LLM API for watermarked texts and search unwatermarked texts online. We use another 500 LLM-generated texts and their corresponding watermarked texts to form the test set. The performance of different machine learning models is shown in Table 2.15. We train a 3-layer transformer [233] (8 heads with 512 dimensions), a BERT-base [52], and a BERT-large [52] on the training dataset and report the detection performance on the test dataset.

From Table 2.15, we can find that the accuracy is 50%, and the F1-Score is 0 for all three models. This suggests that machine-learning models cannot confidently classify whether the texts are watermarked or not. Therefore, the watermark detection attacks fail to detect whether the LLM-generated contents are watermarked or not.

Table 2.15. Classification performance on watermarked and non-watermarked texts. This shows the watermark detection attacks failed to detect whether the LLM-generated contents are watermarked.

Model	Acc. (%)	F1-Score
Transformer [233]	50.00	0
BERT-base [52]	50.00	0
BERT-large [52]	50.00	0

2.3.6 Overall Evaluations

In the previous experiments, we evaluate different watermarking frameworks' performance in terms of (i) effectiveness, (ii) fidelity, (iii) efficiency, (iv) robustness, and (v) undetectability. Their corresponding results are summarized in Figure 2.6. The effectiveness is measured by WER, fidelity is measured by BERT-S, and the robustness is measured by the average AUC after removal attacks on the ChatGPT dataset as in Table 2.6 and Figure 2.4. Undetectability is measured by 1-F1-score with ML-based classification and efficiency is the normalization of average WM insertion time in Table 2.7. Higher numbers are desirable for all metrics. We can observe that prior works demonstrate very sensitive trade-offs among all the criteria. However, REMARK-LLM maintains high performance over all the requirements, making it an ideal toolkit for real-world watermarking applications.

2.4 Conclusion

In this chapter, I present REMARK-LLM, a robust and efficient framework for watermarking text contents generated by LLMs. REMARK-LLM trains message encoding, reparameterization, and message decoding modules jointly, with the aim to accommodate the watermark signatures into the LLM-generated content while ensuring semantic coherence. The message encoding module facilitates the embedding of watermarks into LLM-generated content, while the decoding module extracts messages from these watermarked texts. Comprehensive experiments have demonstrated that REMARK-LLM can embed up to 64 binary bits within 640-token

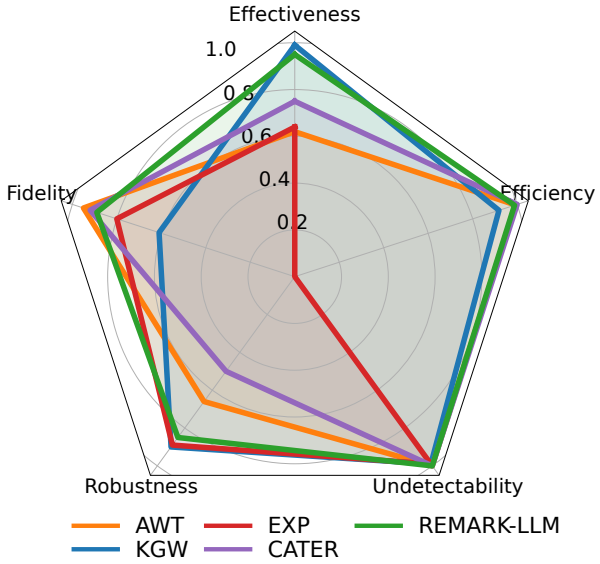


Figure 2.6. Overall evaluations of different frameworks. REMARK-LLM maintains good performance over all the criteria for watermarking, whereas other baselines demonstrate sensitive trade-offs.

texts without compromising semantics and coherence. Furthermore, REMARK-LLM exhibits transferability in watermarking unseen data sources and LLM architectures without additional fine-tuning and showcases resilience against various watermark detection and removal attacks.

2.5 Acknowledgements

Chapter 2 is a reprint of the material as it appears in: Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar, "REMARK-LLM: A robust and efficient watermarking framework for generative large language models," in *USENIX Security Symposium*, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Secure Code Watermarking with ML Crypto Co-design

The AI-empowered code-generation LLMs, such as GitHub Copilot [63], Qwen2.5-Coder [84], and Code Llama [199], generate high-quality code via user instructions. They assist software engineers with agile development and reduce production costs [227, 26]. Developing such powerful models requires substantially more effort compared to natural languages, e.g., designing specialized tokenization modules [119, 199], acquiring high-quality code training data [143, 185], and enhancing the model's reasoning ability [277, 84]. Nevertheless, AI-generated code may be used for malicious purposes and raise ethical and legal concerns, e.g., plagiarizing code that violates academic integrity [42, 228], contributing vulnerable code to open-source repositories [180, 62], deploying code in commercial software without acquiring licenses from LLM owners that infringes intellectual property [254, 238], etc.

Watermarking is suggested as a key solution to regulate LLM-generated content by embedding invisible signatures onto the code [85, 135]. The watermark insertion is formulated as an optimization problem that requires a signature insertion to preserve artifact quality while being identifiable by watermark decoding modules. The existing watermarking solutions fall into two approaches: (i) inference-based watermarking and (ii) neural-based watermarking. Inference-based watermarking [109, 172] encodes watermarks by splitting vocabulary into green/red lists on high-entropy tokens and decoding the next token primarily from the green list.

Such methods manipulate watermark insertion at the token level and do not consider syntactic validity, which can introduce compilation errors and corrupt the code functionality. Neural-based watermarking SrcMarker [254] employs a neural network to encode watermarks in both syntactic and variable name feature spaces for more robust watermarks. Its shallow transformer-based network architecture trained from scratch, however, limits SrcMarker's code feature extraction ability to provide higher watermarking strength and results in reduced detectability.

To verify the ownership, existing schemes require LLM owners to disclose their signatures to third-party arbitrators for legal verification [96, 130], which overlooks the reusability of watermarks. If owners want to use the same artifact, they must re-encode new watermarks to avoid signature forgery or exposure during the verification process. Such approach is practical for images or text, where the space of viable watermarking signatures is large. However, for low-entropy code snippets, the limited watermarking space makes it difficult to generate alternative signatures for re-encoding without losing functionality or reducing watermark resilience.

RoSeMary is the first-of-its-kind code LLM watermarking framework using ML/Crypto co-design that (i) balances detectability, fidelity, and robustness with novel ML-based watermarking architecture design, and (ii) enables watermarked code reusability with customized zero-knowledge proof protocols to keep signatures confidential during efficient verification. The Seq-to-Seq CodeT5 [239], pre-trained on millions of high-quality code snippets, is used as the watermark insertion backbone to extract better code features to fuse with watermarks and enhance watermark detectability. Then, it predicts a set of probabilities over the available syntactic transformations and variables to rename, which are applied to the original code to obtain the watermarked one. A transformer decoder is used for watermark extraction. The watermark encoder and decoder are trained end-to-end to (i) preserve functionality by minimizing the code feature loss between original and watermarked code; and (ii) ensure detectability and robustness by minimizing the message extraction loss between the encoded and extracted message from both the watermarked and the adversarially modified code. As such, RoSeMary strengthens the tri-objective for better watermarking performance.

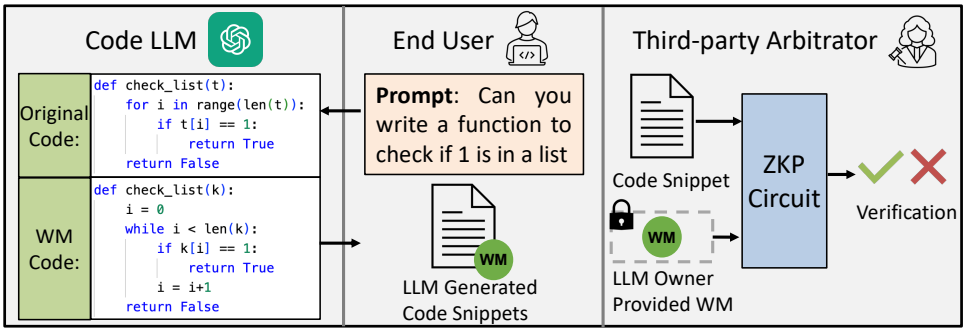


Figure 3.1. Overview of watermark insertion and extraction. The code LLM owner watermarks the code before distributing it to end users. The third-party arbitrator leverages zero-knowledge proofs to verify the ownership without requiring the owner to expose the encoded watermark, and provides legal credibility for potential IP misuse [122].

As in Figure 3.1, during RoSeMary's deployment, the trained watermark encoder embeds the owner's signature into the LLM-generated code and distributes the watermarked one to users. Non-interactive zero-knowledge proofs (ZKPs) are used to verify ownership, allowing the computation-intensive proof generation to be performed once by the LLM owner while enabling an efficient and verifiable proof for subsequent watermark verifications by the arbitrator. If a code snippet is suspected to be LLM-generated, users can submit the code to an arbitrator for legal inspection [96, 130] and request the LLM owner to input their signature to a ZKP circuit as a private input, which remains undisclosed to all parties except the owner. RoSeMary quantizes the watermark decoder to a smaller size and integrates both the watermark decoding and extraction rate computation into a unified computational graph for streamlined verification. Hence, it enables efficient watermark source verification while keeping signatures private. In brief, the contributions are summarized as follows:

- Introduction of RoSeMary, the first code LLM watermarking framework that leverages ML/Crypto co-design to search for high-quality signatures for low-entropy code data, and ensures watermark reusability by keeping it confidential in legal verification with zero-knowledge proofs.

- Enhancing the detectability, fidelity, and robustness by leveraging a pre-trained CodeT5

for better feature extraction and minimizing the code feature loss and message extraction loss between encoded and extracted signatures.

- Customizing and co-optimizing zero-knowledge proof circuit with watermark decoding network to enable efficient ownership verification without revealing encoded signatures.
- Performing evaluations on extensive code benchmarks, demonstrating RoSeMary (i) achieves 0.97 detection AUROC while preserving the code functionality and showing resilience against attacks, and (ii) enables efficient and secure watermark verification within 120ms using zero-knowledge proofs (ZKPs).

3.1 Background on Code Watermarking and Zero Knowledge Proofs

We first introduce the related work of LLM-generated code watermarking in Section 3.1.1, then review the background of zero-knowledge proofs (ZKPs) and optimizations for efficient verification in Section 3.1.2.

3.1.1 Watermarking Large Language Model-generated Code

Compared to natural language, watermarking code needs to preserve both its semantics and functionality. Prior work can be methodologically categorized into two approaches [265]: (i) inference-based watermarking [109, 172], and (ii) neural-based watermarking [254, 238]. The inference-based watermarking [109] encodes signatures during the LLM inference stage. It splits vocabulary into green/red lists only on high-entropy tokens and restricts LLM decoding to predict the next token from the green list. However, it manipulates watermarking insertion at the token level, which may violate syntactic validity and corrupt code functionality. Neural-based code watermarking approach [254] tries to maintain code functionality by encoding watermarks into the code using a neural network. It leverages a dual-channel network to embed watermarks on the code feature space and decodes a set of probabilities over potential syntactic transformations, as

well as the variable to be renamed. Nevertheless, SrcMarker [254] employs a shallow transformer trained from scratch for watermark insertion/extraction, which limits code feature extractability and reduces watermark detectability.

There is another line of work that employs rule-based methods like ACW [120, 110] to watermark code. It maintains a transformation table containing the transformation ID and the rule to transform the code. For each code segment, rule-based watermarking applies available transformations on the original code to form the watermark and obtains the watermarked snippets. The watermarks can be vulnerable to removal attacks if the adversary has prior knowledge of the transformation table. Hence, we do not consider them in this paper.

Due to the low-entropy nature of code, the space for high-quality watermarks is constrained. Once a watermark is disclosed for legal verification [96, 130], re-embedding new signatures may compromise code functionality and resilience. Existing solutions primarily focus on improving watermark quality, but neglect secure verification mechanisms that protect the owner's signature and avoid re-encoding watermarks that may harm code usability.

3.1.2 Zero-knowledge Proofs

Zero-knowledge Proofs (ZKPs) are a cryptographic primitive that allows a prover to prove knowledge of a secret value w to a verifier. In a standard ZKP scheme, the prover $\mathscr{P}$ convinces a verifier $\mathscr{V}$ that w is a valid private input such that $y = \mathscr{C}(x, w)$, in which $\mathscr{C}$ is an arbitrary computation and x and y are public inputs and outputs, respectively. In general, ZKPs are extremely useful in computations where verification of outputs is costly (e.g. machine learning), as ZKPs allow users to verify a small proof rather than repeating the computation themselves [248]. In most ZK schemes, the majority of the computation lies in the setup and proving phases, as any computation $\mathscr{C}$ must be properly encoded in a way that ensures efficient processing during proof generation. In the context of ZK machine learning, for example, the layers, activation functions, and parameters, must all be represented as *circuits*. This process, called *arithmetization*, generally involves the conversion of the computations into arithmetic

operations that can be efficiently performed over a finite field [162]. The setup and arithmetization phases of ZKP protocols are typically where the cryptographic elements are injected to ensure the privacy of w [72].

Zero-knowledge proof generation can be performed in an interactive or non-interactive manner, depending on the application. One of the main drawbacks of interactive schemes is that they limit proofs to *designated-verifier* settings, meaning proof generation, which is the most computationally heavy process in ZKP workflows, must be repeated for every new verifier. Non-interactive ZKPs allow for the *publicly verifiable* setting, meaning that once a proof is generated attesting correct computation or valid data, it can be verified by any third party. Generally, non-interactive ZKP schemes can be represented with the three following algorithms:

- $(\mathscr{VK}, \mathscr{PK}) \leftarrow \text{Setup}(\mathscr{C})$: A trusted third party, when trusted setup is needed, or $\mathscr{V}$ (with publicly verifiable randomness) runs a setup procedure to generate a prover key $\mathscr{PK}$ and verifier key $\mathscr{VK}$.

- $\pi \leftarrow \text{Prove}(\mathscr{PK}, \mathscr{C}, x, y, w)$: $\mathscr{P}$ generates proof π to convince $\mathscr{V}$ that w is a valid witness. A malicious $\mathscr{P}$ cannot generate a valid proof without knowledge of w. Alongside this, π does not reveal anything about w.

- $1/0 \leftarrow \text{Verify}(\mathscr{VK}, \mathscr{C}, x, y, \pi)$: $\mathscr{V}$ accepts or rejects proof π. $\mathscr{V}$ cannot be convinced by an invalid proof due to soundness property of ZKPs [230].

The most notable non-interactive ZKP scheme is Groth16-based zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs), which generate succinct proofs that are of constant size 128 bytes [68]. Due to their succinctness, verification of zk-SNARKs is also very fast - in the order of milliseconds. The main drawback of zk-SNARKs that operate in the Groth16 proof system is the reliance on a computationally heavy trusted setup process, done by a trusted third party, in the presence of every new computation $\mathscr{C}$. This approach is best suited for applications in which $\mathscr{C}$ is relatively static. ZKROWNN [217] shows the feasibility of Groth16

zk-SNARKs for watermark verification in deep neural networks (DNN), requiring only low communication and runtime for a user to verify a proof. However, its primary goal is to protect the watermarks of deep neural networks for IP protection of the models, rather than protecting the watermarks embedded in the LLM-generated data, which is different from RoSeMary.

Although Groth16-based zk-SNARKs work well for computation on the scale of DNNs, their performance begins to falter as $\mathscr{C}$ grows, as they require heavy computation on the prover side to ensure succinctness. RoSeMary utilizes the Halo2 proof system [262] to build efficient zk-SNARKs at a real-world scale, with support for dynamic $\mathscr{C}$. Halo2 utilizes a *universal* and *updatable* setup process, such that trusted setup does not have to be performed for every new $\mathscr{C}$. Halo2 does not enforce constant size proofs. Instead, this proof system produces larger proofs, generally in the range of tens to hundreds of kilobytes, as a tradeoff for higher prover scalability.

3.2 RoSeMary Design

3.2.1 Problem Formulation

Scenario and Motivation

Due to the **constrained high-quality watermark space** available in low-entropy code data, revealing signatures during legal verification and re-encode new ones to avoid forgery and tampering could otherwise degrade code usability. As in Figure 3.1, RoSeMary encodes high-quality signatures onto the LLM-generated code before distributing it to users. A third-party arbitrator decides if the suspect code is from the owner utilizing a ZKP circuit. The owner's signature is a private input to the circuit, ensuring it remains confidential and known only to the owner. RoSeMary maintains the usability of the watermarked code throughout its life cycle with ML/Crypto co-design: (i) introducing a novel watermarking architecture that embeds signatures into code while preserving functionality, enabling high detectability and robustness against

adversarial modifications; (ii) employing ZKPs for secure verification without revealing the watermarkw, thereby eliminating the need of re-encoding watermarks for code reuse.

Threat Model

We consider the adversary to be a malicious end user aiming to remove or forge the watermark to deactivate the ownership proof. He/She can manipulate the watermarked code or discover the signature from the semi-honest third-party arbitrator. However, the adversary cannot control the RoSeMary's watermark insertion or have access to RoSeMary's network parameters or signatures. This setting is consistent with many prior arts [109, 254].

We consider the following attacks following previous work [109, 254]: (i) Variable-rename Attack: the adversary randomly replaces variable names with semantically similar alternatives to remove the watermark while preserving code readability; (ii) Refactor Attack: the adversary uses a large language model to refactor the code in batches. This applies a combination of edits, including variable renaming and structural changes such as reformatting, control-flow restructuring, and adding wrappers (e.g., try/catch blocks); and (iii) Forgery Attack: the adversary counterfeits another set of watermarks and falsely claims a non-LLM-generated code belongs to the owner.

Note that defenses against (i) adaptive attacks [150, 53], where adversaries iteratively modify content based on feedback from the arbitrator to remove watermarks, and (ii) spoofing attacks [202, 64], which alter LLM-generated content into undesired formats while preserving the watermark with the arbitrator's feedback, are orthogonal to RoSeMary's objective to protect watermarks' confidentiality. These attacks typically request verification tens to hundreds of times with similar content and can be mitigated via well-designed authentication protocols [96, 179], such as query rate limiting or anomaly detection. Thus, integrating these complementary defenses can further enhance the overall security of RoSeMary.

3.2.2 RoSeMary Architecture

RoSeMary consists of a watermark insertion and a watermark extraction module. As shown in Figure 3.2, the watermark insertion backbone **S** takes the watermark message M and the code T as input and generates (i) a probability over the syntactic transformations and (ii) variable name distribution over the vocabulary. Then, the code is watermarked by performing the transformations to get the watermarked code $S(T,M)$. Then, a watermark decoder extracts the message M' from the watermarked code $S(T,M)$.

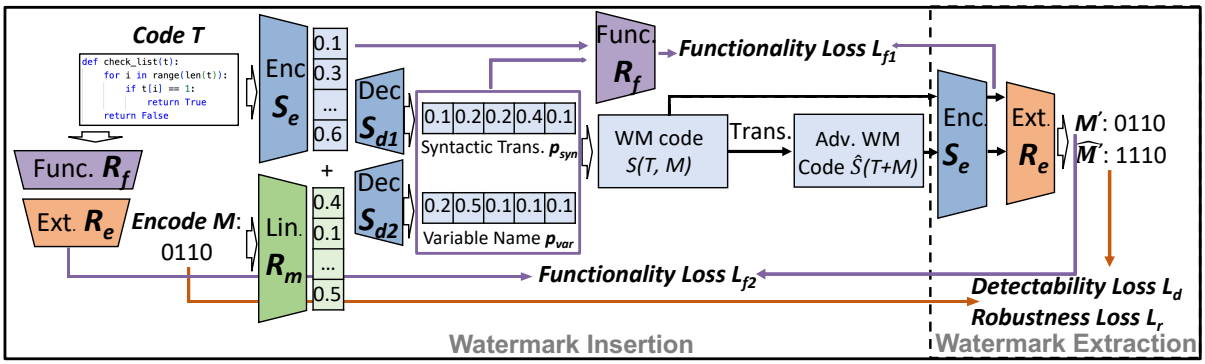


Figure 3.2. RoSeMary watermarking overview. The Watermark Insertion takes the original code and watermark message as input and fuses their features by CodeT5's encoder $\mathbf{S}_e$. Two sets of decoders $\mathbf{S}_{d1}$ and $\mathbf{S}_{d2}$ predict the probability over the available syntactic transformations and the renamed variable over the vocabulary. Then, the Watermark Extraction module decodes watermarks from the syntactic-transformed and variable-renamed watermarked code $S(T,M)$, as well as its adversarial transformation $\hat{S}(T,M)$. The two parts are trained jointly to ensure (i) functionality-invariant by minimizing functionality loss L_f and (ii) accurate and robust message decoding by minimizing detectability loss L_d and robustness loss L_r.

Watermark Insertion The watermark insertion employs the CodeT5 [239], pre-trained on millions of high-quality code files, as the backbone **S** for watermark encoding. The encoder $\mathbf{S}_e$ extracts the code feature and fuses with the message M's feature extracted by $\mathbf{R}_m$. The decoder $\mathbf{S}_{d1}$ and $\mathbf{S}_{d2}$ decodes two sets of probabilities over the syntactic transformations as p_{syn} and variable token distributions p_{var}. Then, RoSeMary obtains the watermarked code $S(T,M)$ by executing the predicted transformations from $argmax(p_{syn})$ and $argmax(p_{var})$.

To approximate realistic post-processing edits without hard-coding a specific refactor-

ing engine into the training loop, we generate adversarial variants by sampling alternative structure-preserving edit decisions from the model's transformation distributions. As shown in Equation 3.1, the AST-level transformation probability distribution p_{syn} and variable-renaming probability distribution p_{var} are perturbed to obtain $\hat{p}_{syn}$ and $\hat{p}_{var}$. It encourages RoSeMary to explore nearby but different edit sequences (e.g., alternative loop/condition forms, formatting-preserving AST rewrites, and different semantically valid renamings). Applying these sampled edits yields an adversarially transformed code $\hat{S}(T,M)$ that preserves functionality but stresses watermark recovery, and we train the extractor to recover M from both $S(T,M)$ and $\hat{S}(T,M)$. Note that this procedure does not aim to fully simulate arbitrary refactoring attacks, but provides a lightweight, differentiable way to cover a broad neighborhood of functionality preserving transformations that helps RoSeMary to decode robust signatures.

$$\begin{aligned} \hat{p}_{syn} &= p_{syn} + \varepsilon_{syn} \\ \hat{p}_{var} &= p_{var} + \varepsilon_{var}, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_p^2) \end{aligned} \tag{3.1}$$

Watermark Extraction The watermark extraction decodes messages from the watermarked code $S(T,M)$. The encoder $\mathbf{S}_e$, with shared parameters and architecture as the watermark insertion, is used to extract code features. Then, a shallow decoder $\mathbf{R}_e$ is used to recover message M'. Similarly, the message $\hat{M}'$ is decoded from the adversarial example $\hat{S}(T,M)$.

Watermarking Strength We measure the watermarking strength under the null hypothesis using a one-sided z-test on the total number of matching bits in Equation 3.2. The higher the z-score is, the more confident an owner can claim the code has been watermarked. For an artifact with F functions, RoSeMary embeds watermark M in each function. M is a binary sequence whose message generation is random, and follows binomial distributions. The probability for generating bit 0 is $p = 0.5$, and bit 1 is $1 - p = 0.5$. The mean of the message distribution is $\mu = F \times |M| \times p$, and the variance is $\sigma^2 = F \times |M| \times p \times (1-p)$. The watermark extraction accuracy is q, thus the expected matching bit is $q \times F \times |M|$.

$$z = \frac{q \times F \times |M| - \mu}{\sigma} \tag{3.2}$$

3.2.3 RoSeMary End-to-End Training

We guide the training by minimizing the loss L in Equation 3.3. RoSeMary is trained to meet three criteria: (i) Functionality-invariant: the functionality of watermarked code $S(T,M)$ remains the same as the input code T as L_f; (ii) Detectability: the decoded message M' matches the encoded M for successful detection as L_d; (iii) Robustness: the adversarial sample $\hat{S}(T,M)$'s decoded message $\hat{M}'$ matches the encoded M for robust detection as L_r.

$$L = w_f L_f + w_d L_d + w_r L_r \tag{3.3}$$

Functionality Loss Performing transformations over T results in non-differentiable watermarked $S(T,M)$. Inspired by SrcMarker [254], we employ $\mathbf{R}_f$ to approximate the watermark insertion/extraction. The $\mathbf{R}_f$ encourages the code functionality feature to be close during training. This is achieved by minimizing the mean square error (MSE) [7] L_{f1} between original cod feature $\mathbf{R}_f(\mathbf{S}_e, p_{syn}, p_{var})$ and watermarked code feature $\mathbf{S}_e(S(T,M))$ as in Equation 3.4. We also ensure the $\mathbf{R}_f$'s approximation is correct for watermark extraction by minimizing the binary cross entropy (BCE) loss [200] L_{f2} between $\mathbf{R}_f(T)$ and predicted $M' = \mathbf{R}_e(\mathbf{S}_e(S(T,M)))$.

$$\begin{aligned} L_f = MSE(\mathbf{R}_f(\mathbf{S}_e, p_{syn}, p_{var}), \mathbf{S}_e(S(T,M))) + \\ BCE(\mathbf{R}_e(\mathbf{R}_f(T)), \mathbf{R}_e(\mathbf{S}_e(S(T,M)))) \end{aligned} \tag{3.4}$$

Detectability Loss RoSeMary minimize the BCE loss between M and M' in Equation 3.5 to recover correct message in watermark extraction.

$$L_d = BCE(M, M') \tag{3.5}$$

Robustness Loss To enable robust message recovery over adversarial transformations, the

watermark extraction also decodes the malicious message $\hat{M}'$ over $\hat{S}(T,M)$ and minimizes the BCE loss between M and $\hat{M}'$ in Equation 3.6.

$$L_r = BCE(M,\hat{M}') \tag{3.6}$$

3.2.4 Secure Watermark Verification

The core problem with watermark extraction is the requirement to reveal the watermarked information to prove ownership of the code. This presents a significant challenge because each time a signature is exposed, the data must undergo re-encoding watermarks to prevent adversaries from discovering, altering, or erasing the exposed signature. Utilizing zero-knowledge proofs (ZKPs) we can solve this problem. We present a unique watermark extraction scheme, built using non-interactive ZKPs, that can efficiently prove that code has been generated using a proprietary LLM, without revealing what the original watermark was. Our solution generates verifiable proofs, such that one proof can be generated and universally verified to prove that a code snippet was generated from a proprietary LLM.

We utilize Halo2-based zk-SNARKs [262], a class of non-interactive ZKPs that offer high scalability and fast verification time. Our proposed system benefits from the fact that proof generation only has to be done once, and, as proof generation is the slowest aspect of Halo2-based zk-SNARKs, we do not need to view this as a bottleneck. Due to the computational overhead of ZKPs, our approach includes non-interactive ZKP-specific optimizations, such as custom quantization, to ensure that the operation is runtime and memory-efficient.

To verify copyrights, the semi-honest legal arbitrator starts by mapping the watermarked code $\mathscr{S}(T,M)$ into its embedding space using the fixed feature extractor $\mathbf{S}_e$. This is done by running a feed-forward process on $\mathbf{S}_e$ with input $\mathscr{S}(T,M)$, which results in a feature vector $\mathscr{S}(T,M)_{\mathsf{embed}}$ that represents the watermarked code in the correct embedding space. A high-level approach towards our zero-knowledge watermark extraction scheme can be seen in Algorithm 2.

The public inputs from the verifier (legal arbitrator) to the verification include $\mathscr{S}(T,M)_{\mathsf{embed}}$ and a target bit error rate (BER) θ. These public inputs do not reveal any sensitive information about the proprietary LLM or watermarking scheme. Besides, the legal arbitrator provides a public digest of the submitted code $h(S(T,M))$, where the prover (LLM owner) checks $h(S(T,M))$ against the submitted code and re-computes $\mathbf{S}_e(S(T,M))$ off-chain to confirm it equals the public $\mathscr{S}(T,M)_{\mathsf{embed}}$.

Then, the LLM owner begins the ZKP generation process. The circuit is instantiated with a fixed shallow decoder $\mathbf{R_e}$ at the setup time such that the prover cannot modify $\mathbf{R_e}$ without invalidating the proof. The original signature M is taken in as private inputs. With all computation represented as a zero-knowledge circuit, the trained watermark extraction module $\mathbf{R}_e$ decodes the signature M' from the watermarked code by performing our custom *zkFeedForward* function on $\mathbf{R_e}$, with the input set to $\mathscr{S}(T,M)_{\mathsf{embed}}$. This results in an extracted signature M' of the same length as M. Within the same zero-knowledge circuit, the BER between the extracted signature M' and the original signature M that the LLM owner provides is calculated. This is done with our provided custom *zkBER* function, which returns 1 if the BER between M' and M is less than θ, or else it returns 0. The resulting proof π will only be valid if the extracted signature M' has a low enough BER compared to the original signature M. This proof π can be sent to any verifier $\mathscr{V}$ to prove that the inspected watermarked code was a result of the owner's proprietary LLM.

A majority of the computational burden lies in the *zkFeedForward*, which requires custom optimization of the shallow decoder to ensure efficient operation when translated to ZK computation. Specifically, $\mathbf{R_e}$ is made up of batch normalization, fully-connected, ReLU, and dropout layers. To implement and run *zkFeedForward*, we use a customized version of the EZKL Rust package [278]. EZKL accepts a computational graph as input, allowing us to optimize our computation before converting it to the correct input format. We provide four custom optimizations and capabilities to ensure efficient proof generation, while maintaining small proof size and fast verification:

❶ We lower the memory requirement for non-linear layers by adding support for polyno-

mial approximations, which is an important technique in privacy-preserving applications. We approximate ReLU using $\sigma(x) = x^2 + x$, which is shown to closely replicate the ReLU [6].

❷ We quantize parameters into Bfloat16, a 16-bit floating point format that reduces the memory requirement for proof generation [23], while keeping watermark extraction accuracy.

❸ We add support for a highly efficient, zero-knowledge bit error rate calculation circuit based on the Halo2 proof system to represent *zkBER*. This is done using the bitwise AND operator to calculate the number of bits that differ between the M' and M.

❹ We add a composability layer that allows for efficient combination of Halo2-based and EZKL circuits (e.g. *zkFeedForward* and *zkBER*) for representation in a single computational graph.

Using these optimizations, we are able to build an efficient ZK watermark extraction and verification scheme with small proofs and fast verification that cleanly integrates into RoSeMary's end-to-end workflow.

Algorithm 2: ZK WM Extraction and Verification

Input: Public values: $\mathscr{S}(T,M)_{\mathsf{embed}}, \theta$; Private input: M
Output: Verification result *valid_BER*
$M' \leftarrow zkFeedForward(\mathbf{R}_e, \mathbf{S}_e(\mathscr{S}(T,M)_{\mathsf{embed}}))$;
$valid_BER \leftarrow zkBER(M, M', \theta)$;
return *valid_BER*

3.3 RoSeMary Evaluations and Results

3.3.1 Experiment Setup

Dataset: We use HumanEval [34], MBXP [13] (MBPP for Python, MBCPP for C++, and MBJP for Java), and EvalPlus [132] as the target benchmark to evaluate RoSeMary's performance. All of the datasets have instruction prompts for code generation, human-written canonical solutions, and test cases for functionality evaluation.

Evaluation Metrics: We assess the frameworks' performance from: (i) ***Detectability***: detecting watermarked versus non-watermarked codes based on their respective z-scores, where ground-

truth labels are set to 1 for watermarked and 0 for non-watermarked samples. AUROC is the area under the receiver operating characteristic curve, TPR is the true positive rates, and FPR is the false positive rates; (ii) ***Fidelity***: the percentage of code passed unit tests as Pass Rate.

Baselines: We compare RoSeMary with state-of-the-art natural language watermarking baselines: (i) ***KGW*** [103] is an inference-based watermarking scheme for natural language. It encodes watermarks at the LLM decoding stage by splitting the vocabulary into green/red lists and guides the decoding to primarily select tokens from the green list; (ii) ***REMARK-LLM*** [265] is a neural-based watermarking scheme for LLM-generated texts. We leverage CodeT5 as the watermark insertion backbone and take both the original code and watermarking signature as input. A watermark extraction module is used to decode the message, which is trained end-to-end with the insertion module to encourage close semantics and successful message extraction. State-of-the-art code watermarking baselines: (iii) ***SWEET*** [109] is an inference-based watermarking scheme for LLM-generated code. It optimizes KGW's decoding by setting an entropy threshold and encodes watermarks only toward high-entropy token decoding; (iv) ***SrcMarker*** [254]: is a neural-based watermarking scheme for LLM-generated code. It employs shadow transformers for watermark insertion/extraction. The watermark insertion generates both syntactic and variable rename probabilities to transform the original code into watermarked code.

Note that we skip baselines that encode watermarks with manually engineered transformation rules like ACW [110] and CodeMark [225], which are fragile to systematic code refactoring attacks [109].

Syntactic Transformations for p_{syn}: We follow SrcMarker's [254] syntactic transformations to construct p_{syn} for C++ and Java. SrcMarker [254] did not watermark Python, and we added the following syntactic transformations: (i) Naming style: choice of naming conventions: PascalCase, camelCase, snake_case, _underscore_init, or ALL_CAPS; (ii) Operator substitution: choice between regular assignment (e.g., $i = i + 1$) and augmented assignment (e.g., $i++$) for operations; (iii) Loop type: choice between using a `for` loop or a `while` loop; (iv) Nested conditions: use of merged conditions versus explicitly nested conditions; and (v) Parentheses in

conditions: option to include or omit parentheses in `if` or `while` conditions. Note that developing more syntactic transformations is orthogonal to RoSeMary's contribution in leveraging pre-trained CodeT5 to extract better code features, and can be combined to further improve its performance.

Implementation Details : Our code is implemented using PyTorch [235]. The training and inference of our watermarking models are performed on NVIDIA RTX A6000 GPUs with Ubuntu 20.04.5 LTS and Intel(R) Xeon(R) Gold 6338 CPU. RoSeMary is pre-trained on CodeSearchNet [86], which collects the open-source non-fork repositories from GitHub and cleans the dataset for executable functions. RoSeMary is trained for 20 epochs with the batch size set to 16, token length set to 512, $\sigma_p = 0.1$, $(w_f, w_d, w_r) = (1, 1, 0.05)$. The watermark bit length set to 4, which provides sufficient watermarking strength according to Section 3.3.2. The neural networks are optimized by an AdamW [140] optimizer with the learning rate set to 5e-5. $\mathbf{R}_f$, $\mathbf{R}_m$, $\mathbf{R}_e$, $\mathbf{S}_{d1}$, and $\mathbf{S}_{d2}$ are all linear layers, which map input feature vectors of dimension 2304, 4, 768, 1536, and 1536 to outputs of dimension 768, 768, 4, 320, and 32100, respectively. We pre-train SrcMarker [254] and REMARK-LLM [265] on the same dataset and report their respective performance. KGW [103] and SWEET [109] encode watermarks during inference. To ensure fair comparisons, we use Qwen2.5-Coder-14B [84] as the code generation model and report the conditional pass rate of watermarked code restricted to instances where the corresponding non-watermarked output is functional.

3.3.2 RoSeMary Performance

The RoSeMary's watermarking performance is in Table 3.1. We highlight that RoSeMary is able to provide high detectability while maintaining code functionality invariant.

Compared to KGW [102]: KGW watermarks LLM-generated code by promoting token decoding from the green list of the vocabulary. We relax the green/red list split ratio γ to 0.5 and set the constant δ added on green tokens' probabilities to 3 to guide watermark insertion on green lists while ensuring Code LLM generates compilable code. As syntactic tokens (e.g., `for` and

Table 3.1. RoSeMary performance on watermarking HumanEval, MBPP, and EvalPlus datasets when comparing with natural language watermarking KGW [103] and REMARK-LLM [265]; code watermarking SWEET [109] and SrcMarker [253]. The best metric values are highlighted in bold text, the second best metric values are underlined, and grey means failed watermark insertion (0% pass rate).

Method	HUMANEVAL [34]				MBPP [13]				EVALPLUS [132]			
	Pass Rate ↑	AUROC ↑	TPR ↑	FPR ↓	Pass Rate ↑	AUROC ↑	TPR ↑	FPR ↓	Pass Rate ↑	AUROC ↑	TPR ↑	FPR ↓
non-WM Code	100%	0.50	-	-	100%	0.50	-	-	100%	0.50	-	-
Natural Language												
KGW	42.62%	0.82	0.56	0.03	57.30%	0.78	0.43	0.03	54.02%	0.73	0.35	0.05
REMARK-LLM	0%	0.97	0.88	0.04	0%	0.98	0.89	0.05	0%	0.98	0.96	0.05
Code												
SWEET	82.53%	0.87	0.59	**0.02**	90.00%	0.86	0.47	0.05	84.90%	0.86	0.52	**0.04**
SrcMarker	95.12%	0.90	0.76	0.07	**97.95%**	0.91	0.76	0.06	**95.57%**	0.92	0.81	0.07
RoSeMary	**95.12%**	**0.97**	**0.98**	0.06	97.64%	**0.97**	**0.99**	**0.05**	95.39%	**0.97**	**0.98**	0.06

`while`) are sensitive to alterations, restricting the watermark insertion at the token-level results in an average of 48.69% pass rate drop to achieve an average of 0.78 AUROC for watermark detection.

Compared to REMARK-LLM [265]: REMARK-LLM primarily replaces words with their synonyms or changes the sentence syntax for watermark insertion. It was able to learn code semantics by leveraging CodeT5 [239] as the watermark insertion backbone and fine-tuning on code datasets. However, it transforms code without syntactic constraints, making the watermarked code non-compileable. As such, while reaching significant detectability, REMARK-LLM has low pass rates that corrupted the watermarked codes' functionality.

Compared to SWEET [109]: SWEET improves over KGW by restricting watermark decoding on high-entropy tokens for better detectability and fidelity. However, encoding watermarks at the token level loses the global view of the code and results in potential syntactic invalidity. As such, it demonstrates a 10.24% drop in pass rate and an average of 10.99% lower AUROC for watermark detection compared to RoSeMary.

Compared to SrcMarker [253]: SrcMarker uses shallow transformers trained from scratch for watermark insertion, which limits its code feature extraction ability. RoSeMary, on the other hand, leverages CodeT5 [239] as the backbone to extract better code features for improved watermark insertion performance, which results in an average of 6.59% higher AUROC scores

and 26.61% higher TPR among all benchmarks than SrcMarker. Besides, both RoSeMary and SrcMarker insert watermarks by applying syntactic transformations and variable renaming that preserve compilation correctness, resulting in less than a 5% drop in pass rate.

Watermarking Strength To quantify statistical evidence against the null hypothesis of no watermark presence, we compute a one-sided z-score from the aggregated bit matches across the dataset following Equation 3.2. Using a one-tailed significance level of 0.05, the rejection threshold is $z \geq 1.645$. When F is set to 1, the average z is 1.92, which already exceeds this threshold. Therefore, it provides statistically significant evidence of watermark presence and demonstrates that RoSeMary yields sufficient watermarking strength to prove ownership.

The watermark would be further strengthened when multiple functions are stacked, which is a more realistic setting when a feature requires multiple code functions to implement. Say we watermark a file with $F = 5$ and RoSeMary encode 4-bit signatures on every function. Then, the z-score would be boosted to 4.29 with the one-tailed significance level to be 1.8×10^{-5}, providing much stronger support for ownership verification.

3.3.3 Multi-lingual Code Watermarking

In Table 3.2, we compare RoSeMary's ability to watermark other coding languages (C++ and Java) with the best-performing baseline SrcMarker. The pre-trained CodeT5 model effectively extracts code features from different coding languages for high-quality watermark insertion. As such, RoSeMary consistently achieves high detectability and functionality-preservation after watermarking. RoSeMary averagely improves AUROC by 3.78% and TPR by 18.90% over SrcMarker [254], while maintaining a comparable FPR. Notably, both methods achieve over 98% pass rate, demonstrating that watermarking introduces minimal disruption to code functionality. .

3.3.4 Zero Knowledge Watermark Verification Overhead

We benchmark our ZK watermark extraction and verification by describing algorithm 2 in a computational graph that can be easily translated into a zero-knowledge circuit. The

Table 3.2. Multi-lingual code watermarking performance.

Metrics	MBCPP (C++)		MBJP (Java)	
	SrcMarker	RoSeMary	SrcMarker	RoSeMary
Pass Rate ↑	98.69%	**98.95%**	98.69%	**98.69%**
AUROC ↑	0.93	**0.96**	0.92	**0.96**
TPR ↑	0.80	**0.98**	0.84	**0.97**
FPR ↓	**0.06**	0.07	**0.06**	0.07

computation is done using the EZKL framework [278] in conjunction with the Halo2 proof system [262], resulting in a small zk-SNARK proof that any third-party arbitrator can easily verify. Our approach ensures that no information is leaked about the parameters of the decoder $\mathbf{R_e}$ and the watermarking methodology, including the original watermark M. The verification overhead is in Table 3.3. Generating the zk-SNARK proof only takes the prover $\mathscr{P}$ **6.79 seconds**, while only requiring a maximum of approximately **2.79 GB** of RAM. While this is significantly slower than standard inference, we highlight that this is a fully privacy-preserving solution, and, more importantly, proof generation only has to be done once. This process results in a proof π of size **18.75 KB**, which can be transmitted to as many verifiers as necessary. We benchmarked the verification over 25 examples. This proof can be verified by any verifier $\mathscr{V}$ in an average of **120 milliseconds**, while only requiring a maximum of approximately **227.88 MB** of RAM. All the verifier needs to verify a proof is the proof π and the verifier key $\mathscr{VK}$, which is only **511 KB** in our setup. This results in a required communication cost of less than a megabyte. As the proof generation time is amortized due to it only being performed once, RoSeMary's watermark extraction and verification scheme is an extremely communication and runtime-efficient solution that ensures the security of watermarks that are applied to LLM-generated code.

Table 3.3. Zero-knowledge proof generation (Proof Gen.) and watermark verification (WM Verif.) overhead.

	Frequency	Comm. Size	RAM	Time
Proof Gen.	Once	Proof Size: 18.75 KB	2.79 GB	6.79s
WM Verif.	Every WM	$\mathscr{V}$ Key Size: 511 KB	227.88 MB	120ms

3.3.5 Robustness Evaluations

We assume the adversary is an end-user who adopts LLM-generated code for malicious purposes. Their goal is to preserve the code's functionality while removing the embedded signatures to evade detection. The adversary has access only to the watermarked code and cannot manipulate the LLM owner's code generation or watermarking process. They may discover the signatures if verification is performed without a ZKP circuit, but when the signature is provided as a private input within the ZKP circuit, it remains hidden from the adversary. As a proof of concept, we evaluate attack performance on the MBPP dataset [13].

Watermark Removal Attack Following Section 3.2.1, we consider: (i) **Variable-Rename Attack** (VA): the adversary randomly renames variables using semantically similar words to preserve both functionality and readability. We modify 10%, 30%, and 50% of the variables in each function. (ii) **Refactor Attack** (RA): the adversary refactors the watermarked code using open-source code LLMs, which is able to provide comparable accuracy but more code refactoring variants compared to traditional code refactoring tools [177]. Here, we instruct Qwen2.5-Coder-32B-Instruct [84] model to refactor the code using the prompt: "You are Qwen Coder. You are a helpful assistant that performs code refactoring to remove watermark. Please only return the refactored code." RA is a compositional transformation that can simultaneously apply multiple semantics-preserving edits (e.g., reformatting, control-flow restructuring, try/catch wrapping, and variable renaming) while attempting to keep the original behavior unchanged. We use RA as a practical proxy for real-world post-processing pipelines that developers may apply before reuse the watermarked code.

The attack results are shown in Figure 3.3. We compare RoSeMary with SrcMarker [253] and SWEET [109]. RoSeMary maintains an AUROC above 0.93 under variable-rename attacks even when 50% of variable names are replaced, whereas SrcMarker and SWEET show AUROC decreases of 0.07 and 0.21, respectively. Under refactor attacks, RoSeMary achieves an AUROC

of 0.73 and demonstrates strong robustness. This robustness arises mainly from two factors: (i) leveraging CodeT5 [239] for improved feature extraction to enhance detectability, and (ii) adversarial training, which enables the extraction module to learn malicious transformations and accurately recover watermark signatures.

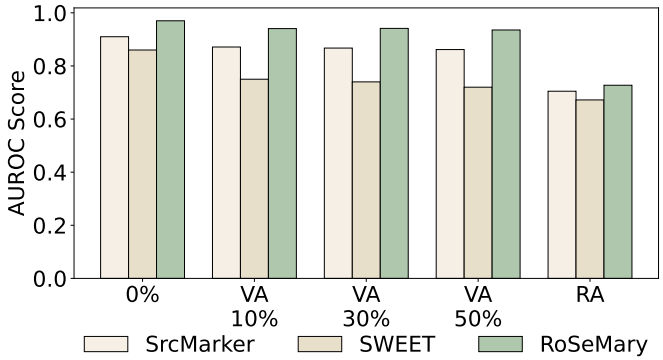


Figure 3.3. Robustness evaluation results under different attacks.

Watermark Forgery Attack RoSeMary is also resilient to **forgery attacks**, where an adversary attempts to produce a proof that validates an ownership claim without possessing the owner's secret signature. In our threat model, end users only have access to the released watermarked code, while the decoding module, seed, and signature remain confidential to the owner. Moreover, verification is performed via zero-knowledge proofs, where the signature is provided as a private input to the circuit and is not disclosed to the arbitrator or any third party. As a result, an adversary cannot learn the signature from the verification, nor can a malicious arbitrator exploit the verification process to impersonate the owner and generate false ownership claims.

3.3.6 Ablation Study and Analysis

Watermark Insertion Overhead The time taken for watermark insertion is in Table 3.4, which is the average overhead for encoding signatures onto 50 examples from the MBPP [13] dataset. RoSeMary embeds watermarks 90% faster than SWEET's inference-based approach, which requires entropy calculation for every token and splits the vocabulary into green/red lists for

high-entropy tokens. Compared to SrcMarker, while RoSeMary introduces more complex architectures for better watermarking performance, the additional watermark insertion overhead is less than 0.01s per sample. As such, RoSeMary's watermark insertion is efficient.

Table 3.4. Different frameworks' watermark insertion overhead.

Method	SWEET	SrcMarker	RoSeMary
Time (s)	0.234	0.021	0.027

Impact of Training Loss Weights The impact of different loss weight choices on watermark performance is in Table 3.5. We train RoSeMary on the CodeSearchNet [86] and evaluate its performance on MBPP [13]. Weighting more on the detectability loss w_d leads to better detection scores (AUROC, TPR, and FPR) than the robustness loss (w_r), as excessive focus on robustness against malicious modifications may hinder extraction performance on unaltered code. Additionally, increasing the weight of the functionality loss (w_f) maintains a comparable pass rate to the other configurations.

Table 3.5. Impact of loss weights on RoSeMary's performance.

(w_f, w_d, w_r)	Pass Rate ↑	AUROC ↑	TPR ↑	FPR ↓
(0.8, 0.1, 0.1)	97.74%	0.95	0.93	0.07
(0.1, 0.8, 0.1)	**97.74%**	**0.97**	**0.98**	**0.05**
(0.1, 0.1, 0.8)	97.74%	0.97	0.95	0.05

Impact of Decoder $\mathbf{S}_{d1}$ and $\mathbf{S}_{d2}$ We analyze the impact of training with multiple decoders in Table 3.6. We pre-train RoSeMary on the CodeSearchNet [86], with either $\mathbf{S}_{d1}$ or $\mathbf{S}_{d2}$, and evaluate the performance on MBPP [13]. As seen, encoding watermarks with sole syntactic transformations ($\mathbf{S}_{d1}$) or variable name transformations ($\mathbf{S}_{d2}$) results in close pass rates but degraded detectability, as less information can be embedded onto the code for watermark insertion. Besides, encoding only syntactic transformations results in higher detectability. Because such transformations carry more information for the signature feature to encode into the watermarked code.

Table 3.6. Impact of decoders on RoSeMary's performance.

S_{d1}	S_{d2}	Pass Rate ↑	AUROC ↑	TPR ↑	FPR ↓
✓	✗	98.05%	0.81	0.36	0.06
✗	✓	**98.77%**	0.73	0.24	0.06
✓	✓	97.64%	**0.97**	**0.99**	**0.05**

Watermark Examples We show a few watermark examples in Figure 3.4, where Listing 3.1,3.3 are original code and Listing 3.2,3.4 are watermarked one. RoSeMary will transform the variable names and syntactic structure with meaningful content while maintaining the functionality invariant. For example, Listing 3.1's two if statements (line 6 and line 7) are merged into one (line 6) in Listing 3.2. The function name is updated from `get_closest_vowel` to `getClosestVowel`. In Listing 3.4, the variable name `i` is changed to `item`, and parentheses are added for `(t>0)`.

Listing 3.1. non-WM code #1

```
def get_closest_vowel(word):
    if len(word) < 3:
        return ""
    vowels = {"a", "e", "i", "o", "u", "A", "E", 'O', 'U', 'I'}
    for i in range(len(word)-2, 0, -1):
        if word[i] in vowels:
            if (word[i+1] not in vowels) and (word[i-1] not in vowels):
                return word[i]
    return ""
```

Listing 3.2. WM code #1

```
def getClosestVowel(dates):
    if (len(dates) < 3):
        return ""
    vowels = {"a", "e", "i", "o", "u", "A", "E", 'O', 'U', 'I'}
    for i in range(len(dates)-2, 0, -1):
        if (dates[i] in vowels and (dates[i+1] not in vowels) and (dates[i-1]
                not in vowels)):
            return dates[i]
    return ""
```

Listing 3.3. non-WM code #2

```
def histogram(test):
    dict1={}
    list1=test.split(" ")
    t=0
    for i in list1:
        if(list1.count(i)>t) and i!='':
            t=list1.count(i)
        if t>0:
            for i in list1:
                if(list1.count(i)==t):
                    dict1[i]=t
            return dict1
```

Listing 3.4. WM code #2

```
def histogram(test):
    dict1={}
    list1=test.split(" ")
    t=0
    for item in list1:
        if ((list1.count(item)>t) and item !=''):
            t=list1.count(item)
        if (t>0):
            for item in list1:
                if(list1.count(item)==t):
                    dict1[item]=t
    return dict1
```

Figure 3.4. Watermarked examples randomly selected from HumanEval and MBPP dataset. The Listing 3.1,3.3 are the original code (non-WM code), and the Listing 3.2,3.4 show the watermarked one (WM code), where all watermarks are successfully extracted.

3.4 Conclusion

In this chapter, I present RoSeMary, the first-of-its-kind ML/Crypto codesign framework to secure high-quality code LLM watermark verification. It trains the watermark insertion and

extraction modules end-to-end, aiming to ensure the watermarked codes' functionality-invariant, while maintaining the detectability of the watermark in the adversarial environment. To guarantee signature confidentiality and preserve code usability, RoSeMary customizes and co-optimizes non-interactive zero-knowledge proofs for secure watermark verification, which allows third-party legal arbitrators to verify ownership without revealing the signature to them. Extensive evaluations across various coding benchmarks demonstrate that RoSeMary effectively preserves code fidelity, achieves high detectability and robustness, and supports efficient zero-knowledge verification.

3.5 Acknowledgements

Chapter 3 is a reprint of the material as it appears in: Ruisi Zhang, Neusha Javidnia, Nojan Sheybani, and Farinaz Koushanfar, "Robust and Secure Code Watermarking for Large Language Models via ML/Crypto Codesign," in *arXiv preprint arXiv:2502.02068*, 2025. The dissertation author was the primary investigator and author of this paper.

Part II

Secure Authentication and Attestation for Edge AI Deployment

Chapter 4

Model Authentication with Edge LLMs Watermarking

Deploying large language models (LLM) on resource-constrained edge platforms entails model compression [219, 89, 251] to reduce the model memory size and bandwidth. The compressed and embedded LLMs [189] reduce cost and energy for inference while enhancing local data privacy protection. Optimizing for the most compressed LLM within a quality bound is computationally costly, and thus, the resulting models become valuable intellectual property (IP) for the owners. Concurrently, unlike black-box cloud-based LLM APIs [210], compressed models deployed on edge devices grant full model access to end-users. The shift in the access paradigm introduced new security challenges for model copyrights and made the deployed models prone to IP theft attacks. Therefore, it is crucial to devise techniques to protect embedded model proprietors' IPs.

Watermarking protects the LLM proprietor's IP by inserting unique signatures onto the model parameters. Prior solutions fall into two approaches: (i) *training-time watermarking*, and, (ii) *post-training watermarking*. During model training, training-time watermarking inserts signatures onto the parameter weight distributions [45, 33] or as backdoors [212, 117] to generate unintended outputs via trojan activations. Such methods, however, are computationally heavy and hard to scale to the larger models.

Post-training watermarking is introduced in SpecMark [30] and adapted for LLMs in Qiu *et.al.* [114] to insert signatures onto the trained model weights. SpecMark converts model weights to the discrete cosine transform domain to encode signatures. Qiu *et.al.* [114] co-optimizes the full-precision and quantized LLMs to ensure watermarks as backdoors are in the full-precision model, and the quantized LLM lacks such malfunctions. These techniques successfully watermark full-precision models with dense weight distributions, where small additives can serve as watermarks. However, embedded and quantized [1] LLMs bring new challenges, in which weight distributions are discrete and sparse, allowing fewer alternatives for watermark insertion.

This chapter presents EmMark, a robust watermarking framework for protecting the IP of embedded LLMs deployed in resource-constraint edge devices. EmMark encompasses a *watermark insertion* stage and a *watermark extraction* stage. The *watermark insertion* encodes signatures into the quantized LLM by a novel scoring function. The function assesses each weight parameter from two aspects: (i) quality preservation by the sensitivity of weight parameters to signature insertion, and, (ii) robustness by watermarking on saliency weight channels critical to the LLM quality. The first item is evaluated by the absolute value of weight parameters, where larger values are less sensitive to addition and deletion during signature insertion. The second item is reflected from the full-precision model activation distribution [125], where weight channels with larger activations are more salient. Leveraging the parameter scores, EmMark randomly selects a subset of best-performing candidates to encode the watermark signatures before model deployment.

The *watermark extraction* reproduces the scoring function to obtain the watermark weight locations using the random seed, original model weights, and full precision model activations. Then, the model proprietor queries the watermarked model and decodes the signatures at the watermark weight locations. The ownership can be claimed by comparing the encoded and

[1] Note that the most significant aspect of compressed embedded models from the perspective of the watermark is quantization, which substantially changed weight distributions and data types. Thus, in the remainder of the paper, we use the terms compressed and quantized models interchangeably.

decoded signatures.

The deployed watermarked LLM is resilient to watermark removal and forging attacks, aiming to remove or counterfeit the signature to misappropriate the IP. EmMark defends the watermark removal attacks by encoding on the salient region. The adversary has to perturb a larger portion of salient weight parameters to remove watermarks, leading to LLM quality compromises. The watermark insertion is also confidential. Adversaries without access to the full-precision model cannot reproduce the model activation for parameter scoring and, thereby, cannot obtain the watermark weight location for signature counterfeiting. In brief, our contributions are summarized as follows:

- Introduction of EmMark, a robust watermarking framework for embedded quantized LLMs deployed in resource-constraint edge devices. The *watermark insertion* stage leverages a novel scoring function to encode signatures with both quality preservation and robustness; The *watermark extraction* stage decodes signatures and asserts ownership.
- Proof-of-concept experiments on embedded LLMs from OPT and LLaMA-2 families show EmMark achieves 100% watermark extraction without quality degradations.
- Analysis of EmMark capacity indicates up to 100-bit signatures can be inserted per layer into low-bit embedded LLMs without quality deterioration.
- Extensive evaluations of EmMark under various watermark removal and forging attacks demonstrate its resiliency.

4.1 Background and Related Work

In this section, we first introduce the related work for large language model quantization. Then, we present literature on watermarking machine learning models.

4.1.1 Large Language Model Quantization

On-device LLMs lessen the cost, power, and network requirements for running the powerful generative models while enhancing local data's privacy [189]. Different model compression algorithms, like pruning [151, 224], and quantization [125, 247], are applied to reduce the model size and fit it into the resource constraints in the target platforms. One of the most vital steps of deploying the model to the embedded platform for better inference speed is quantization [125, 247] that maps a full-precision FP32/FP16 model onto lower-precision INT8/INT4.

Due to the considerable fine-tuning overheads, training-aware quantizations are hard to be applied to LLMs. To address this, post-training quantization is commonly used to quantize LLMs without introducing significant computation burderns. Given the floating point tensor $\mathbf{X}$, the number of bits N to quantize, Equation 4.1 depicts how $\mathbf{X}$ is quantized into $\overline{\mathbf{X}}$ with quantization step size Δ. In LLM quantization, the tensor $\mathbf{X}$ can be activations and/or weights, depending on the constraints in the target platform.

$$\overline{\mathbf{X}} = \text{Round}\left(\frac{\mathbf{X}}{\Delta}\right), \quad \Delta = \frac{\max(|\mathbf{X}|)}{2^{N-1}-1} \tag{4.1}$$

The post-training quantization goes in two directions [257]: (1) INT8 quantization [50, 247], where activation and/or weights are quantized into INT8; (2) Low-bit quantization [57, 125], where activation and/or weights are quantized to low bits like INT4. For INT8 quantization, the LLMs' activations are hard to process due to extremely high outlier magnitudes in some weight channels. `Llm.int8()` [50] uses mixed-precision decomposition to isolate the outlier activations into a float16 matrix multiplication. The rest of the parameters use INT8 computation. Outlier Suppression [241] improves the scheme by applying non-scaling LayerNorm and token-wise clipping to reduce outliers. SmoothQuant [247] enhances the INT8 quantization using a mathematically equivalent transformation to migrate high-magnitude activations to low-magnitude weights. For Low-bit quantization, GPTQ [57] uses second-order methods to obtain a closed-form solution for the low-bit quantization optimization. However, it overfits

the calibration dataset, and has bad generalization to new dataset distributions at the inference time. AWQ [125] improves low-bit quantization by identifying the salient weights in LLMs and rescaling the salient weights before quantization.

4.1.2 Machine Learning Model Watermarking

Machine learning model watermarking refers to adding digital signatures onto the model parameters to enable ownership proof. Prior arts insert watermarks during the model training/fine-tuning stage. DeepSign [45] and DeepMarks [33] encode watermarks onto the probability density function (pdf) of models' weight distributions. It is implemented by adding the watermark signatures as an additional regularization loss term during model training. Follow-up work also suggests adding backdoors during model training as watermarks [212, 117]. The models' abnormal behaviors under certain trojan activations serve as watermarks to prove ownership. While the inserted watermarks are robust to potential attacks, the insertion process requires significant computation resources and is hard to scale up to LLMs.

More recent post-training watermarking inserts watermarks on the full-precision trained model parameters. SpecMark [30] inserts watermarks on the audio models by first converting the parameters to the discrete cosine transform (DCT) domain. Then, it inserts signatures at the high-frequency region of model parameters. Qiu *et.al.* [114] proposed to watermark LLMs by co-optimizing the full-precision and quantized LLMs. Its objective is to ensure the quantized LLM works normally, whereas the full-precision model is watermarked by pre-defined backdoors.

All the aforementioned watermarking frameworks are designed for full-precision FP16 or FP32 models. No previous work explored watermarking embedded quantized LLMs on edge devices. Therefore, we present EmMark as the first watermarking framework for embedded LLM and protecting owners' IP in the edge.

4.2 Threat Model

4.2.1 Motivation and Scenario

Deploying generative large language models in the edge fuels the broader applications for mobile users and IoT devices. For cloud LLM APIs like ChatGPT and GPT-4 [210], users only have black-box access to the LLMs. However, end-users in the edge devices have full access to the LLMs locally. The switch in the security model leads to new threats to the LLM copyrights. To defend against potential IP thefts, there is an urgent need to devise robust watermarking frameworks to protect embedded LLM owners' IP in the edge.

The embedded LLMs deployed in the edge devices are compressed for better hardware efficiency. The most significant aspect of the compressed model from the watermarking perspective is quantization, which maps the full-precision weights into INT8/INT4 for reduced memory size and bandwidth. The sparse weight distributions introduced new challenges for model watermarking compared with full-precision models.

EmMark inserts owners' signatures onto the compressed and quantized LLM before model deployment. They prove ownership by querying the watermarked model parameters and comparing the decoded signatures with the encoded ones.

4.2.2 Watermarking Criteria

An ideal watermarking framework for edge LLMs should meet the following criteria [30]. By taking these criteria into the design, EmMark emerges as a robust watermarking framework for embedded LLM IP protection on resource-constraint edge devices.

- **Fidelity**: the watermarks should be successfully inserted into the models while preserving model quality.
- **Robustness**: the watermarks should withstand various removal or forging attacks.

- **Efficiency**: the watermark insertion should be efficient both in terms of time and computation overheads.

4.2.3 Watermarking Threats

We assume the adversary in the edge devices has full access to the watermarked embedded LLM parameters and has knowledge of watermark insertion algorithms. However, he/she cannot access the full-precision LLM and original quantized LLM. The adversary also does not know the owners' signatures or random seeds for watermark parameter selections.

Potential threats to the embedded models include (i) parameter overwriting attacks [21], where other values replace model parameters; (ii) re-watermarking attacks [45], where adversary corrupts the original watermark by embedding new signatures; and (iii) forging attacks [21], where the adversary counterfeits fake watermarks from watermarked model and claim the model belongs to him.

Note that parameter pruning and fine-tuning attacks cannot be applied to embedded LLM. Firstly, pruning the compressed model results in model ability breakdown and generates outputs as NaN (Not a Number). Besides, fine-tuning quantized model like QLoRA [51] does not change quantized weights but adds additional linear low-rank adaptators to learn new features.

4.3 EmMark Method

EmMark's global flow is depicted in Figure 4.1, comprising two major steps, namely, *watermark insertion* and *watermark extraction*. The watermark insertion encodes watermark signatures into the compressed and quantized LLMs before deployment. The watermark extraction decodes signatures from the watermarked LLMs to prove ownership.

4.3.1 *Watermark Insertion*

EmMark takes the original N-bit compressed and quantized LLM M and the signature sequence $B = \{b_1, b_2, ..., b_{|B|}\}$ as input. In B, each element $b_i \in \{-1, 1\}$. The watermarks are

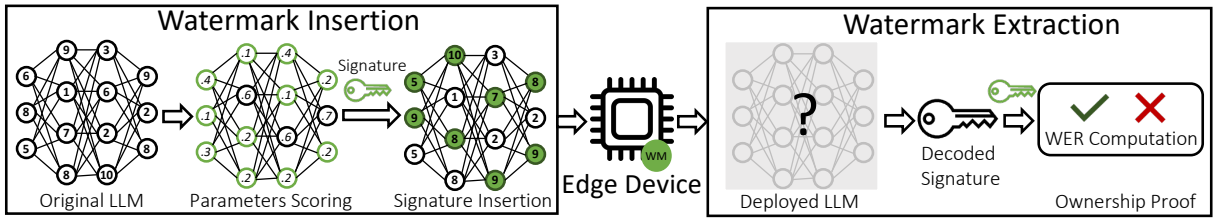


Figure 4.1. EmMark watermarking overview. The watermark insertion encodes signatures into the original LLM before deployment. The watermark extraction decodes the signatures from the deployed LLM and proves ownership. The green circles in Parameters Scoring are candidate watermark locations, and the green weights in Signature Insertion are watermarked weights. The bold value is the model weight parameters, and the italics value is the corresponding scores **S**.

inserted into M's weights $\mathscr{W}$. The activation of corresponding full-precision LLM in each weight channel is $\mathscr{A}_f$.

Parameters Scoring As mentioned in Section 4.2, the watermarked LLM shall: (i) preserve original models' quality; and (ii) be robust against removal and forging attacks. We search for quantized weight parameters with such qualities by Equation 4.2, where $\mathbf{S}_q$ evaluates the quality preservation and $\mathbf{S}_r$ assesses the robustness. The two scores are weighted using coefficients α and β ($\alpha, \beta \geq 0$).

$$\mathbf{S} = \alpha \mathbf{S}_q + \beta \mathbf{S}_r \tag{4.2}$$

For i-th quantized weight parameter $\mathscr{W}_i$, we measure its corresponding $\mathbf{S}_q$ and $\mathbf{S}_r$ to accommodate signature b_j as follows. The first quality score $\mathbf{S}_q$ is defined in Equation 4.3. Weight parameters with larger absolute values are less sensitive to slight changes (additions/deletions) from watermark insertion. Thus, it results in better quality preservation. Note that $\mathscr{W}_i$ in the minimum and maximum quantization level is set to 0 before scoring.A smaller $\mathbf{S}_q$ indicates the weight is less sensitive to signature insertions.

$$\mathbf{S}_q = |\frac{b_j}{\mathscr{W}_i}| \tag{4.3}$$

The score $\mathbf{S}_r$ in Equation 4.4 measures the robustness of each quantized weight param-

eter. It defends against (i) removal attacks by watermarking on the salient region. To remove watermarks, the adversary has to perturb a larger fraction of saliency weights, resulting in LLM performance degradations. (ii) forging attacks by scoring with full-precision model. Adversary does not have access to the full-precision model, and cannot reproduce the score $\mathbf{S}_r$ for signature counterfeiting.

The weight parameter saliency has strong correlations with the activation magnitudes. The larger activations process more incoming features, and the corresponding weight channels are more sailent [125]. Inspired by this, we formulate the saliency of the weight parameter in each channel as the normalization of current channel magnitude $|\mathscr{A}_{fi}|$. A smaller $\mathbf{S}_r$ indicates the weight channel contributes more to the LLM quality.

$$\mathbf{S}_r = |\frac{\max(\mathscr{A}_f)}{\mathscr{A}_{fi} - \min(\mathscr{A}_f)}| \tag{4.4}$$

EmMark scores each quantized weight parameter using Equation 4.2, and obtain the scores for each $\mathscr{W}$. For the i-th weight parameter $\mathscr{W}_i$, a smaller score means the position is better for watermark insertion. Therefore, for a n quantization layer model, we choose B_c smallest candidate weight parameters from $\mathscr{W}$ in every quantization layer as the candidate location for watermark insertion. Here, $|B| \ll |B_c| \times n$.

Signature Insertion To ensure the even signatures distribution, for a n quantization layer model, we insert $\frac{|B|}{n}$ signatures into each layer. To maintain the secrecy of inserted signatures, EmMark randomly choose $\frac{|B|}{n}$ weight parameters out of the $|B_c|$ candidates in the current layer using random seed d. EmMark obtains the watermark weight locations L. The insertion of watermarks follows Equation 4.5, where the signatures are encoded into the quantized weights. The watermark consists of (i) signature sequence B; (ii) the random seed d, the original quantized weight $\mathscr{W}$, full-precision activation $\mathscr{A}_f$, and α, β coefficients for location L reproduction.

$$\mathscr{W}'[L_i] = \mathscr{W}[L_i] + b_i \quad \text{for } i \in [1, |B|] \tag{4.5}$$

4.3.2 *Watermark Extraction*

Ownership Proof EmMark reproduces the watermark weight locations L with the random seed d, quantized model weights $\mathscr{W}$, full-precision activation $\mathscr{A}_f$, and α, β coefficients. At L, EmMark compares the extracted weight $\mathscr{W}'[L]$ with the original weight $\mathscr{W}[L]$ and gets their difference $\Delta\mathscr{W}[L]$ using Equation 4.6. Model owners can assert ownership by comparing $\Delta\mathscr{W}[L]$ with inserted signature sequence B.

$$\Delta\mathscr{W}[L] = \mathscr{W}'[L] - \mathscr{W}[L] \tag{4.6}$$

The watermark extraction rates are computed using Equation 4.7, where $|B|$ is the length of the inserted signature, and $|B|'$ is the number of matching signature bits.

$$\%WER = 100 \times \frac{|B|'}{|B|} \tag{4.7}$$

Watermarking strength Equation 4.8 evaluates the probability that a non-watermarked model matches the inserted signatures by chance. k is the number of matching bits between the owner's and non-watermarked model's signatures. $|B|$ is the signature length. The signature generation follows the Rademacher distribution, and each bit has an equal probability of 0.5 to be 1 or -1.

$$P_c = \sum_{i=k}^{|B|} \begin{pmatrix} |B| \\ i \end{pmatrix} 0.5^{|B|} \tag{4.8}$$

4.4 EmMark Evaluations and Results

4.4.1 Experiment Setup

Target Model We use LLaMA-2 [237] and OPT [269] family models with parameter sizes ranging from 125 million to 70 billion as the target LLM. The models are compressed and quantized by Smoothquant [247] to INT8 for OPT family, by `LLM.int8()` [50] to INT8 for LLaMA-2 family, and by AWQ [125] to INT4. Note that EmMark is agnostic to quantization algorithms. We use the the frameworks as a proof-of-concept showing EmMark's performance. We watermark on top of the official model quantization instances [242].

Watermark Parameters For the target model, we insert 300-bit signatures per INT8 quantized layer and 40-bit per INT4 quantized layer. It yields a minimum watermarking strength of 9.09 $\times 10^{-13}$ for each layer, and 9.09 $\times 10^{-13n}$ for n quantized layer LLM following Equation 7.16. Such watermarking strength provides sufficient protection to the target models. The coefficients α and β are set to 0.5 and 0.5. The seed for signature insertion is 100. For model size smaller than 6.7 billion, $\frac{|B_c| \times n}{|B|} = 50$. For larger LLMs, $\frac{|B_c| \times n}{|B|} = 60$.

Baselines We compare EmMark with RandomWM where watermarks are inserted in random indexes and SpecMark where watermarks are inserted into the high-frequency region transformed by DCT. SpecMark is designed for full precision models, and we apply the transformation to the quantized weights.

Evaluation Metrics The watermarked LLM performance is evaluated using *Perplexity (PPL)* for text fluency on WikiText Dataset [155], and, *Zero-shot Accuracy (Zero-shot Acc)* for token prediction on the mean of LAMBADA, HellaSwag, PIQA, and WinoGrande Datasets [60]; the watermark extraction performance is evaluated using the percentage of signatures successfully extracted as *Watermark Extraction Rate (WER)*; the watermark efficiency is evaluated using the *Insertion Time (Time)* and required *GPU Memory (Memory)*.

4.4.2 EmMark's Fidelity

Different watermarking frameworks' performance is in Table 4.1. The first and the second cells are the perplexity and zero-shot accuracy degradation for accommodating the signatures. The third cell is the watermark extraction rates.

Table 4.1. Watermarked embedded large language models performance. The first and second cell is the LLM performance measured by perplexity (PPL) and zero-shot accuracy (Zero-shot Acc). The third cell is the watermark extraction rate (WER). The best metric values are in bold, and the text in grey means failed watermark insertion (0% WER). $\bar{\Delta}$ is the average performance degradation compared with non-watermarked models.

Metrics	Method	INT8 Quantization										INT4 Quantization									
	Model	OPT						LLaMA-2			$\bar{\Delta}$	OPT						LLaMA-2			$\bar{\Delta}$
		125M	1.3B	2.7B	6.7B	13B	30B	7B	13B	70B		125M	1.3B	2.7B	6.7B	13B	30B	7B	13B	70B	
PPL ↓	w/o WM	48.72	19.51	18.25	22.35	16.41	22.30	9.26	8.26	4.93	0	33.97	16.83	14.61	12.43	11.60	9.77	9.47	8.41	4.94	0
	SpecMark [30]	48.72	19.51	18.25	22.35	16.41	22.30	9.26	8.26	4.93	0	33.97	16.83	14.61	12.43	11.60	9.77	9.47	8.41	4.94	0
	RandomWM	48.75	19.56	18.28	22.43	16.39	22.37	9.26	8.31	4.93	+0.03	34.03	17.56	27.88	14.07	13.16	10.77	9.48	8.43	4.95	+2.29
	EmMark	**48.71**	**19.49**	**18.21**	**22.34**	**16.34**	**22.28**	**9.26**	**8.26**	**4.93**	**0**	**33.96**	**16.83**	**14.61**	**12.43**	**11.60**	**9.75**	**9.47**	**8.41**	**4.94**	**0**
Zero-shot Acc (%) ↑	w/o WM	44.68	56.66	61.28	65.36	65.43	67.59	67.19	69.05	73.65	0	44.47	57.63	61.37	64.52	65.21	67.87	67.26	68.89	73.76	0
	SpecMark [30]	44.68	56.66	61.28	65.36	65.43	67.59	67.19	69.05	73.65	0	44.47	57.63	61.37	64.52	65.21	67.87	67.26	68.89	73.76	0
	RandomWM	44.60	56.64	61.14	65.06	65.47	67.56	66.98	69.03	73.21	-0.13	44.32	57.58	60.22	64.45	65.11	67.75	67.16	68.74	73.62	-0.23
	EmMark	**45.34**	**56.70**	**61.35**	**65.50**	**65.43**	**67.59**	**67.19**	**69.05**	**73.65**	**0**	**44.49**	**57.79**	**61.38**	**64.53**	**65.23**	**67.87**	**67.29**	**68.90**	**73.76**	**0**
WER (%) ↑	SpecMark [30]	0	0	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	-
	RandomWM	100	100	100	100	100	100	100	100	100	-	100	100	100	100	100	100	100	100	100	-
	EmMark	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	-	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	-

Comparison with RandomWM: While RandomWM performs on par with the non-watermarked models in INT8 quantization, it failed to preserve lower-bit quantization models quality. Its performance drops significantly in INT4 quantization, where the degradation is 2.29 and 0.23%, respectively, for perplexity and zero-shot accuracy. As a result, the watermarked LLM has worse text fluency. In contrast, EmMark introduced no performance degradations compared with non-watermarked LLMs in both INT8 and INT4 quantization. By strategically scoring quantized parameters, EmMark preserved the model quality during watermark insertion.

Comparison with SpecMark: As seen from Table 4.1, SpecMark failed to watermark embedded LLM and achieved 0% WER in both INT8 and INT4 quantization. The weight distributions in embedded LLMs are sparse, and the small additions/deletions from the DCT domain cannot change the discrete parameters in the weight domain. As a result, SpecMark failed to watermark such LLMs. EmMark, however, successfully inserted signatures into the embedded model without introducing additional quality deterioration.

4.4.3 EmMark's Efficiency

The efficiency is evaluated by the required time (Time) and GPU memory (Memory) for watermark insertion on the OPT family models [269]. In Table 4.2, we report the average time/memory for signature insertion per quantization layer. For both INT8 and INT4 quantization, the time taken to watermark one quantization layer is less than 0.4s. The insertion takes less than 10 minutes for the largest OPT-30B model. All of EmMark's components are performed on CPUs, and no additional GPU memory is required for watermark insertion. As such, the efficient and lightweight watermarking scheme makes EmMark scalable to LLMs with larger sizes.

Table 4.2. EmMark's watermarking efficiency.

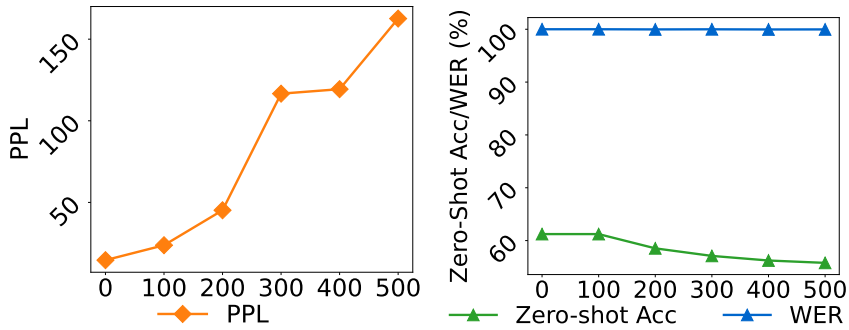
Quantization	Time (s)	Memory (GB)
INT8	0.4	0
INT4	0.3	0

4.4.4 Robustness

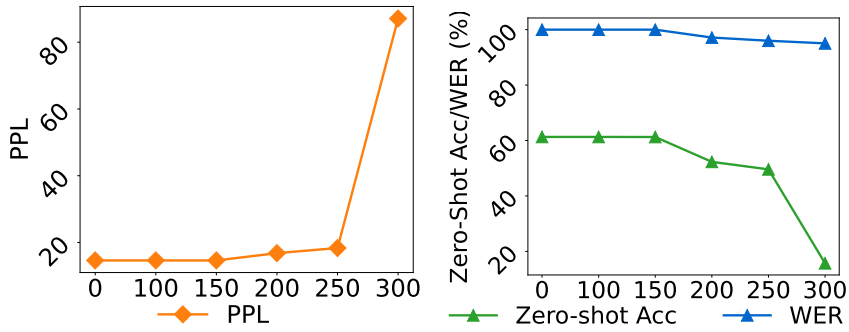
We evaluate EmMark's robustness by performing the following attacks: (i) parameter overwriting attacks, (ii) re-watermark attacks, and (iii) forging attacks. We use the OPT-2.7B [269] model quantized to INT4 by AWQ [125] as the target model. The embedded model is already compressed and quantized. Therefore, additional pruning attacks will result in model ability breakdown. Current fine-tuning on the quantized models uses LoRA-based approaches [51] to add low-rank adaptors to learn new dataset features. Such methods, however, will not change the quantized model weight parameters, and cannot be used to remove signatures.

Parameter Overwriting Attacks The adversary removes the watermark by randomly adding one bit to the parameter weights in the watermarked model. The attacked model performance (perplexity and zero-shot accuracy) and watermark extraction rates are shown in Figure 4.2a. The number of overwritten parameters in each quantized layer is increased from 100 to 500 with a constant gap of 100. As seen, the attacked model performance drops as more bits are overwritten. The threshold renders significant model quality degradations is at 300-bit, where the

PPL is over 100 and indicates bad generated text fluency. Nevertheless, EmMark demonstrates its robustness under such attacks and maintains over 99% WER.



(a) Parameter Overwriting Attacks



(b) Re-watermark Attacks

Figure 4.2. EmMark's performance under parameter overwriting and re-watermarking attacks. The left subplot evaluates Perplexity (PPL), and the right subplot depicts Zero-shot Accuracy and Watermark Extraction Rate (WER).

Re-watermark Attacks The adversary knows EmMark's general watermark insertion algorithm. However, he/she cannot access the model owners' signatures or random seeds. The adversary tries to break the watermark by perturbing parameters potentially used for watermarking. The watermark coefficients α and β and seed differ from the watermark insertion stage and are set to 1, 1.5, and 22, respectively. The activation for scoring $\mathbf{S}_r$ is obtained from the quantized LLM instead of the full-precision one. The quantization model performance (perplexity and zero-shot accuracy) and watermark extraction rates are shown in Figure 4.2b. The number of perturbed parameters in each quantized layer is increased from 100 to 300 with a constant gap of 50.

Figure 4.2b shows EmMark maintains its high watermark extraction rates under re-watermark attacks. The threshold for bad LLM quality is at 300 bits, where the zero-shot accuracy is below 20% and yields bad token prediction performance. However, the watermarked model still maintains over 95% watermark extraction rates, providing sufficient IP protection to the embedded LLM.

Forging Attacks The adversary does not remove the LLM owners' watermark. Instead, he/she claims the model ownership by faking another set of watermarks. It is achieved by (i) counterfeiting the watermark weight locations L_a with a fake signature sequence; (ii) re-watermarking on top of the watermarked embedded LLM by a counterfeited full-precision model activations and insertion hyperparameters. EmMark is resilient to the two settings with a confidential full-precision model's activation that the adversary does not have access to.

In the first setting, the watermark weight locations are a reproduction of the full-precision model's activations and a set of hyperparameters (random seed and scoring coefficients). Counterfeiting the location L_a without such reproduction validation process leads to failed forging attacks. In the second setting, while the adversary proves ownership of the re-watermarked model, the inserted signatures from owners are still highly extractable as seen from the ***Re-watermark Attacks*** section. Notably, matching the owners' signatures by coincidence is nearly impossible and the probability goes as low as 9.09×10^{-13n} for n quantization layer LLM following Equation 7.16. Here, n=192 for OPT-2.7B, and larger as the model size grows. As such, the owners are able to claim ownership under forging attacks and protect the IP.

4.4.5 Ablation Study and Analysis

We provide further ablation study and analysis of EmMark's capabilities. The OPT-2.7B [269] quantized by AWQ [125] to INT4 is the target model.

Effectiveness of Watermark Coefficients We analyze how different α and β choices affect EmMark's watermarking performance in Table 4.3. α is changed from 0 to 1, and β is changed from 1 to 0. The maximum watermark signature length is set to 100. As shown, as the coefficient

β becomes larger, EmMark tends to choose bits in the saliency channel over bits with larger values. While it ensures strong IP protection, the watermarked LLM's quality is compromised.

Table 4.3. Effectiveness of different insertion coefficients.

(α, β)	PPL ↓	Zero-shot Acc (%) ↑	WER (%) ↑
(1, 0)	14.61	61.36	100
(0.5, 0.5)	14.61	61.36	100
(0, 1)	14.65	61.25	100

Watermark Capacities We show the maximum signature length can be inserted into the embedded LLM without compromising its quality in Figure 4.3. We increase the inserted length per quantization layer from 50 to 200 with a constant gap of 50. As seen, the threshold for maintaining watermarked LLM performance is at 100 bit. It corresponds to a watermark strength of 1.57×10^{-30} per layer and 1.57×10^{-5760} for OPT-2.7B model following Equation 7.16, providing sufficient protection to the model IP.

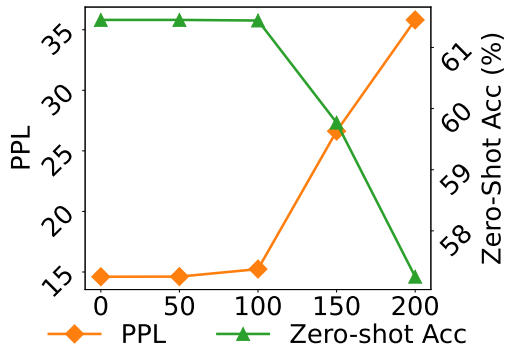


Figure 4.3. EmMark's watermark performance with inserted signature lengths increased from 50-bit to 200-bit. All of the watermarks are successfully extracted.

Watermark Integrity To ensure integrity, EmMark shall only prove ownership of watermarked models and generate low WER on non-watermarked ones. In Table 4.4, the first model is the non-watermarked OPT-2.7B quantized by AWQ [125], the second model is fine-tuned on a 4k subset of the Alpaca dataset [231] before AWQ [125] quantization, the third model is fine-tuned on WikiText dataset [155] before AWQ [125] quantization, and the fourth model is non-watermarked OPT-2.7B quantized by GPTQ [57]. The EmMark successfully extracts all

signatures from the watermarked model and fails to prove ownership of non-watermarked ones. As a result, EmMark demonstrates its integrity.

Table 4.4. EmMark's integrity evaluation on watermarked and non-watermarked models.

Model	WM	non-WM 1	non-WM 2	non-WM 3	non-WM 4
WER (%)	100	0	0	0	0

4.5 Conclusion

We present EmMark, a novel watermarking framework for embedded large language models. It inserts signatures into the compressed and quantized models while demonstrating quality preservation and robustness. As such, it tackles the security challenges arising from deploying large language models in edge devices and protects the IP of model owners. Extensive evaluations on OPT and LLaMA-2 family models show EmMark successfully inserts 100% signatures into the LLM with no performance compromises.

4.6 Acknowledgements

Chapter 4 is a reprint of the material as it appears in: Ruisi Zhang, and Farinaz Koushanfar, "EmMark: Robust Watermarks for IP Protection of Embedded Quantized Large Language Models," in *ACM/IEEE Design Automation Conference*, 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Hardware Authentication with Edge LLMs Attestation

On-device large language models (LLMs) [249, 136, 259] have been widely deployed in consumer electronic devices to enable local model inference with reduced latency and enhanced user privacy. Delivering high-quality user experiences requires substantial effort in both model compression and hardware architecture optimizations [12, 189]. However, such local inference introduces an underexplored challenge: untrusted third parties along the supply chain (e.g., resellers) often have access to the device and its rich execution environment (REE) [31, 266]. This opens the door to potential unauthorized misuse, where the pre-installed, vendor-optimized LLMs can be replaced with illegal and unauthorized models. Such tampering not only infringes on the hardware vendor's intellectual property (IP) by running undesired models on the platform, but also erodes its reputation and customer trust if devices exhibit degraded performance, unsafe behaviors, or inconsistent user experiences.

Attestation [31] is introduced to verify the legitimacy of the models within trusted execution environment (TEE), where model's origin is securely authenticated by matching a decoded watermark (WM) against a device-specific signature. Only watermarked models that have passed attestation are permitted to perform inference in rich execution environment (REE), while unauthorized ones are blocked. However, existing attestation techniques are primarily

designed for smaller neural networks with millions of parameters, and scaling them to billion-parameter LLMs remains highly challenging. The bottlenecks mainly come from: (i) **hardware constraints**: on-device TEE, such as Arm TrustZone [184], only provides 10–32 MB of secure memory, which is insufficient to load LLMs' parameters and activations, let alone perform efficient attestation; (ii) **watermark verification efficiency and robustness**: the watermark verification shall be computationally lightweight to enable timely attestation, while remaining robust against attacks.

This chapter presents AttestLLM, the first attestation framework to protect the hardware-level IP of on-device LLMs. It consists of two key components: offline watermarking and online attestation. **Offline watermarking** embeds quality-preserving and robust signatures onto the LLM. AttestLLM profiles the model and adaptively allocates the watermark budget based on each layer's performance sensitivity. The signatures are then stealthily encoded via an optimization-based algorithm on the trigger dataset, which preserves model quality, while robustly anchoring the watermark to each layer's output activations. AttestLLM enables independent layer-wise watermark verification by saving the input activation to LLM layer, and checking watermarks from its output activation. **Online attestation** efficiently verifies model legitimacy with system/algorithm co-design. It enlarges the limited secure memory of Arm TrustZone [184] via a virtualization-based enclave [48] and interleaves with REE inference to avoid throughput degradation. During attestation, AttestLLM dynamically samples a subset of LLM layers and decrypts watermarking keys and input activations from the trigger dataset for verification, ensuring sufficient attestation strength to resist evasion attempts. Those layers are attested in parallel, and such verification computations are overlapped with secure copy communication to further reduce attestation latency.

As detailed in Figure 5.1, the LLM, embedded with watermarks, is executed within the REE. During inference, ① attestation requests are issued periodically, and ② AttestLLM running in the TEE verifies the embedded watermarks. A hypervisor monitors and ensures the LLM executing in REE corresponds to the model verified in TEE. ③ If the watermark extraction

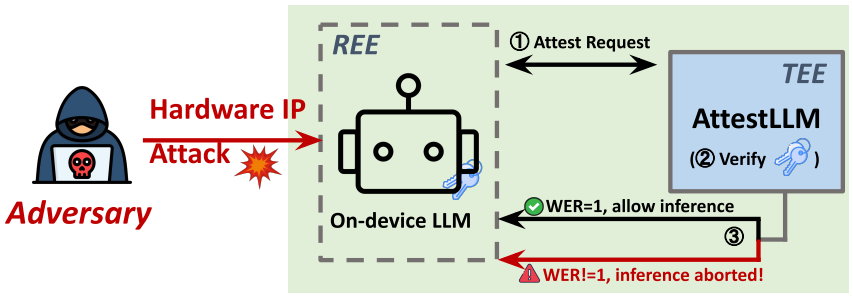


Figure 5.1. Attestation overview: AttestLLM in the TEE periodically attests models in the REE, ensuring that only authenticated LLMs are executed while blocking unauthorized ones.

rate (WER) equals 1, the model is authorized to run; otherwise, inference is blocked, thereby preventing the execution of unauthorized or tampered models. In summary, the contributions are as follows:

- We present AttestLLM, the first attestation framework to protect the hardware-level IP of device vendors, ensuring only authorized LLMs can execute on target platforms.
- We co-design algorithm/software/hardware to scale the attestation to billion-parameter LLMs with: (i) **quality-preserving and robust watermarking** to encode device-specific signatures into on-device LLMs; (ii) **efficient and scalable attestation** to minimize attestation overhead under resource-constrained hardware platforms.
- We evaluate AttestLLM on various on-device LLMs, showing: (i) fidelity preservation while ensuring 100% watermark extraction rate; (ii) efficient attestation on Arm-based edge device, which achieves 62× lower latency overhead and 30× lower energy overhead compared to the best prior hardware IP protection work. (iii) resists various replacement, forgery, and TEE system attacks.

5.1 Background and Related Work

5.1.1 On-device LLM Watermarking

To deploy LLMs on edge devices, researchers typically leverage various compression techniques, such as knowledge distillation [113, 246] and quantization [55, 137], to obtain a

compact and low-precision format to reduce the inference burden. One of the most significant steps is quantization, which maps the model from high-precision float32 [173] or bfloat16 [49] into int8 or lower. These model optimizations are often co-designed with target hardware platforms, optimizing kernel implementations [80, 66], memory layouts [263], and resource allocation [131] strategies to fully exploit hardware capabilities for faster inference, which constitute valuable IP.

Watermarking for protecting the intellectual property of LLMs goes in two directions: (i) parameter-based watermarking [45, 266, 261]: embedding unique identifiers directly into the model's parameters, enabling ownership verification when the model weights are accessible. It could be done during training [45] by adding an additional regularization term into the loss function, or post-hoc [266, 261] by weight sensitivity analysis to enable watermarking insertion without introducing quality degradations; and (ii) backdoor-based watermarking [114, 116, 115]: encoding secret triggers onto the LLM's token prediction probabilities, and produce distinctive behavior only when specific inputs are encountered. This is done typically by fine-tuning pre-trained language models with multi-task learning and specific triggers to enable robust watermark extraction even after downstream fine-tuning.

5.1.2 TEE for LLM IP Protection

TEE is a secure area within the main processor that ensures the confidentiality and integrity of code and data through hardware-enforced isolation [201]. Most edge devices adopt the ARM architecture [191], whose TEE, ARM TrustZone [184], enhances hardware security by dividing the processor into two isolated virtualized domains: the normal world (NW) for general-purpose workloads and the secure world (SW) for trusted applications. Armv8-A processors structure the privilege hierarchy into four exception levels (EL0-EL3): EL0 is for user-space applications; EL1 hosts the operating system kernel with access to system resources; EL2 is the hypervisor layer supporting virtualization, which can intercept and control OSes at EL1/EL0; and EL3 is reserved for highest-privilege firmware and the secure monitor that orchestrates

transitions between the secure and normal worlds.

Edge devices are constrained by limited memory and computational resources, which introduce significant overhead when executing end-to-end LLM inference entirely within TEEs. Prior work for protecting LLM intellectual property and ensuring model legitimacy fall into two categories: (i) TEE-shield inference [138, 272, 59] executes the entire model inside the TEE by partitioning weights into memory-fitting segments and running them sequentially. However, this approach incurs frequent world switches and secure memory operations, resulting in substantial communication and context-switching overhead; and (ii) TEE-based attestation [31], which performs model inference in the REE while verifying pre-embedded watermarks within the TEE. The state-of-the-art work, DeepAttest [31], is limited to attesting models with only millions of parameters on Intel SGX [40] and lacks support for ARM-based TEEs that dominate modern edge devices [191]. In this paper, AttestLLM introduces the first algorithm/software/hardware co-design approach that scales attestation to billion-parameter LLMs on Arm-based devices while maintaining both efficiency and robustness.

5.2 Threat Model

5.2.1 Scenario

As in Figure 5.1, hardware providers [136, 189, 12] embed LLMs directly onto edge devices to reduce network dependency, strengthen privacy, lower inference costs, and improve responsiveness. However, such on-device deployment introduces a critical security risk: third parties along the supply chain (e.g., resellers) may act as adversaries. With access to the device, they can repurpose compute resources for unapproved LLM execution or deploy models that violate safety and ethical standards. These threats not only compromise the vendor's hardware-level intellectual property (IP) but also damage its reputation and user trust if devices exhibit degraded performance or unsafe behaviors. AttestLLM enables hardware vendors to attest the authenticity of LLMs running in the REE and terminate execution of any unauthorized models.

5.2.2 Attackers' capability and potential threats

We consider the adversary cannot manipulate the secure trusted execution environment (TEE) on the device. The operating system (OS) is trusted and can provide information about the pages associated with the weights (e.g., page fault or modified) under hypervisor mode in ARM TrustZone. This is consistent with prior attestation work [31, 160]. The adversary has prior knowledge of the watermarking algorithm, but lacks access to the trigger dataset and watermark, which are securely encrypted and stored in a shuffled, randomized memory region. They aim to evade attestation to host another unauthorized model on the hardware device.

To achieve this, the adversaries may execute these attacks. (Detailed evaluations in Section 5.4.5): (i) **Watermark forgery attack**: The adversary forges the signature and attempts to host an illegitimate LLM that decodes the same device-specific signature in TEE to bypass attestation; (ii) **Model replacement attack**: The adversary has prior knowledge of the model architecture and attempts to host a similar LLM with different parameters; (iii) **TEE system attack**: The adversary bypasses attestation by running an unauthorized LLM in the REE while either halting attestation interactions with the TEE or dispatching the watermarked model only for TEE attestation. Note that defenses [69, 160] against side-channel attacks [275, 108], which aim to infer data or control flow within the TEE, are orthogonal to AttestLLM, and can be combined to strengthen its security.

5.3 AttestLLM Method

As in Figure 5.2, AttestLLM consists of two main components: (i) offline watermarking (Section 5.3.1), which encodes device-specific watermarks onto LLM while preserving model quality and robustness; and (ii) online attestation (Section 5.3.2), which securely verifies the watermarks within the hardware's protected enclave in TEE, ensuring only the owner's authorized LLM can be executed in REE.

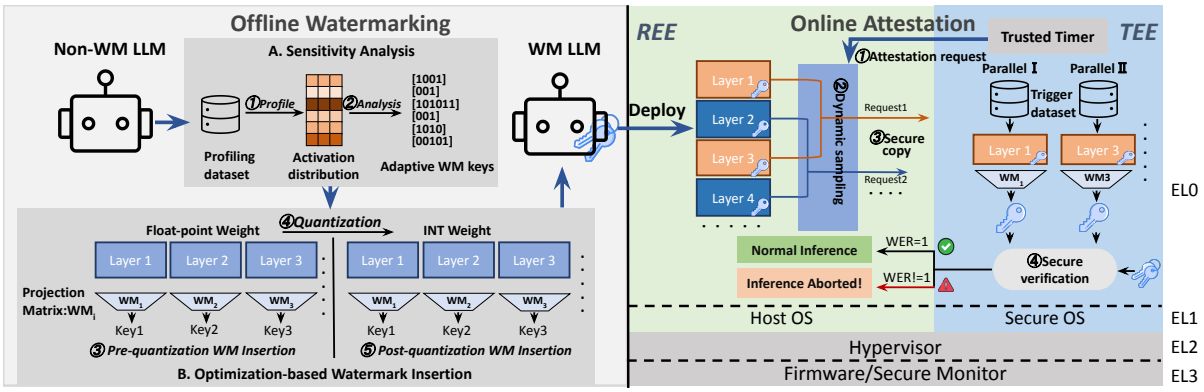


Figure 5.2. AttestLLM pipeline. Offline watermarking performs (A) sensitivity analysis to adaptively allocate watermark across layers, and (B) optimization-based watermark insertion to embed device-specific signatures with fidelity and robustness; Online attestation attests watermarked LLM on edge devices by periodically requesting attestation, and securely copying sampled LLM layers into the TEE for watermark verification. Only authorized LLM (WER=1) is allowed for normal inference.

5.3.1 Offline LLM Watermarking

Sensitivity Analysis Different transformer layers capture and process different stages of input text representations [73]. Some layers can accommodate more watermark bits without affecting model utility, whereas others are more sensitive and should receive fewer. Weight parameter saliency correlates strongly with activation magnitude, where channels with larger peak activations process more information and are therefore more salient [126]. Inspired by this, for an LLM M with $|M|$ layers and a watermark sequence $B = \{b_1, b_2, ..., b_{|B|}\}$, where $b_i \in \{0,1\}$, we allocate signature to layer i as B_i. As in Equation 5.1, B_i is inversely proportional to the layer's peak activation magnitude $\max|\hat{\mathscr{A}_i}|$, profiled by a profiling dataset D_{pro}, ensuring that more salient layers receive fewer signature bits and less salient layers absorb more.

$$B_i = |B| * \frac{1/\max|\hat{\mathscr{A}_i}|}{\sum_1^{|M|} 1/\max|\hat{\mathscr{A}_i}|} \tag{5.1}$$

Optimization-based Watermark Insertion To deploy an LLM on edge devices, the most significant step of model compression from the perspective of watermarking is quantization,

Algorithm 3: Quantization-aware WM embedding

Input: i-th layer M_i, activations from $(i-1)$-th layer A_{i-1}, signature B_i, projection matrix WM_i

Output: Watermarked and quantized transformer layer M_i'

$M_i' \leftarrow M_i$

for $epoch \leftarrow 1$ **to** $Epoch_{pre}$ **do** // Pre-quantization WM insertion

  $A_i \leftarrow M_i'(A_{i-1})$

  $B_i' \leftarrow \text{WM}_i(A_i)$

  $L \leftarrow L(M_i', \text{WM}_i, B_i, A_{i-1})$

  Update M_i', WM_i via gradient descent with learning rate η_{pre}

$M_i' \leftarrow \text{Quantize}(M_i')$

for $t \leftarrow 1$ **to** $Epoch_{post}$ **do** // Post-quantization WM insertion

  Select a subset Θ of parameters from M_i'

  Sample a perturbation p in the parameter space of Θ

  $L^+ \leftarrow L(M_i' + p, \text{WM}_i, B_i, A_{i-1})$

  $L^- \leftarrow L(M_i' - p, \text{WM}_i, B_i, A_{i-1})$

  $\hat{g} \leftarrow \frac{L^+ - L^-}{2} \cdot p$

  $M_i' \leftarrow M_i' - \eta_{\text{post}} \hat{g}$

which substantially changes weight distributions and data types. Thus, we devise a quantization-aware optimization-based watermarking insertion algorithm in Algorithm 3. Given the activation from the $(i-1)$-th transformer layer as $\mathscr{A}_{i-1}$ (on trigger dataset D_{tri}), the activation $\mathscr{A}_i$ from the i-th watermarked transformer layer M_i' is projected into the watermarking space by a projection matrix WM_i. The watermark embedding goal is to make the projected output approximate the ground-truth signature B_i. Besides, to maintain model fidelity, a penalty term $\alpha \|M_i' - M_i\|^2$ is added to the loss, constraining the updated parameters M_i' to remain close to the original M_i. These objectives are formulated in Equation 5.2.

$$L = \min_{M_i', WM_i} \left| WM_i\left(M_i'(\mathscr{A}_{i-1})\right) - B_i \right| + \alpha \|M_i' - M_i\|^2 \tag{5.2}$$

To achieve robust watermark embedding while preserving model fidelity, AttestLLM introduces a two-stage watermark insertion before and after quantization as in Algorithm 3. In the pre-quantization stage, the optimization is performed on the full-precision model using gradient-based methods. Both M_i' and WM_i are updated via backpropagation to approximate the target

signature B_i efficiently. After quantization, model weights become discrete and zeroth-order optimization [221] is employed to further enhance signature insertion. A subset of parameters Θ is perturbed along random directions for p in the space, and the corresponding losses L^+ and L^- are estimated using finite-difference evaluations [193]. These estimates are used to refine the quantized transformer layer M_i' to reduce the discrepancy between the decoded watermark signature B_i' and its target B_i. The model fidelity is kept by constraining LLM layer parameters' divergence and robustness is reinforced by encrypting trigger dataset D_{tri} and watermark keys to keep them confidential.

Watermark Verification The watermark insertion enables each LLM layer's watermark to be verified independently. Given the trigger dataset D_{tri}, AttestLLM saves the activation $\mathscr{A}$ to all watermarked layers. When verifying the i-th layer, it will retrieve $\mathscr{A}_{i-1}$ as an input to watermarked layer M_i'. The output $M_i'(\mathscr{A}_{i-1})$ is projected by WM_i to decode signature as B_i'. The watermark extraction rate is computed in Equation 5.3, where $|B_i|$ is the length of the inserted signature, and $|B_i'|$ is the number of matching signatures.

$$\%WER = 100 \times \frac{|B_i'|}{|B_i|} \tag{5.3}$$

5.3.2 Online Attestation with TEE

TEE Setup and Attestation Workflow AttestLLM enlarges the available secure memory using virtualization-based TEE [48]. At platform bootstrapping, hardware root-of-trust verifies the integrity of the bootloader, host kernel image, and device tree. During the early boot of the host kernel (before switching fully to the REE), the system reserves security-sensitive resources (e.g., I/O passthrough, enclave memory) that must be owned by the TEE. This ensures the REE cannot interfere with these reserved resources and TEE obtains enlarged memory for scalable attestation. After the hypervisor and memory partitioning are set up, the host kernel (which contains the

hypervisor and portions of the REE) and device tree are verified and then booted [111, 160].

During attestation, ①a trusted timer inside TEE sends signals to request attestation of a few selected LLM layers. Then, REE ②samples the layers and ③securely copies them in a read-only region via virtio-pmem [234], where they can be efficiently transferred into enclave pages in TEE. Inside TEE enclave, ④AttestLLM retrieves and decrypts the encrypted activations $\mathscr{A}$ using AES [4] to verify the watermark of received model layers. If verification succeeds (i.e., WER = 1), the model will be executed in REE; otherwise, the enclave raises an alert and enforces a device reset to block untrusted execution. Such attestation is performed periodically to ensure the runtime integrity of embedded LLMs.

To enhance security, AttestLLM uses these mechanisms to guard against online and offline data tampering. To avoid online tampering, (i) a trusted time tracks the attestation interval and aborts the REE application's execution if no response is received within the expected time window; (ii) the attest layer selection is generated randomly and securely inside the TEE, preventing an attacker from manipulating layer checking and thereby bypassing attestation of tampered layers; (iii) to avoid attacks on layer sampling and model execution, e.g., replacing sub-modules or adding LoRA adapters [81], AttestLLM under hypervisor mode monitors OS events and tracks OS pages allocated to the LLM program. If any modification to those pages is detected at runtime, a trigger is raised and execution is blocked. To avoid offline data tampering (i.e., modifications to stored data or model weights before execution), AttestLLM shuffles the weights when storing them in untrusted memory via memory-layout randomization [215, 142, 31]. This randomization relocates code and data segments and breaks persistent layouts, which significantly reduces the feasibility of brute-force searches for tampering with stored model weights and watermark keys.

AttestLLM reduces the attestation overhead by performing verification dynamically on a subset of LLM layers while ensuring attestation strength. The attestation overhead can be adjusted by an attestation interval f, which triggers the attestation request every f generated tokens. Based on demand, vendors can set a smaller f (e.g., 50) for privacy-critical scenarios,

while a larger f (e.g., 200-500) for latency-sensitive or conversational applications. At each attestation round, the system randomly samples k layers from $|M|$ total layers and securely copies them into TEE for watermark verification, where the attestation strength is formulated by evasion probability P_{evasion}. If an attacker tampers t layers, the probability of evading one attestation is $p = \frac{\binom{|M|-t}{k}}{\binom{|M|}{k}}$. After r attestation rounds, the evasion probability is in Equation 5.4. A sufficiently low evasion probability provides significant attestation strength to protect the device from hardware-level IP attacks.

$$P_{\text{evasion}} = \left(\frac{\binom{|M|-t}{k}}{\binom{|M|}{k}} \right)^r \tag{5.4}$$

Optimizations for Efficient Attestation To reduce latency under limited TEE resources, we design a pipelined and partially parallel verification flow. Each attestation round includes (i) *communication intensive* stage that secures copies of transformer layers and decrypts WM key, (ii) *computation intensive* stage that extracts and verifies watermarks. For m TEE threads, AttestLLM dedicates one thread for secure copying and decryption, while the rest are for computation of watermark extraction and verification. The secure copy of k_i layer can be pipelined and happen concurrently with the computations of $k_{i-m} - k_{i-1}$ layers. In addition, the $m-1$ compute threads can be further parallelized to process those $k_{i-m} - k_{i-1}$ layers concurrently. The verification will be early-exit and trigger signals to abort REE inference as soon as a mismatch is found to accelerate detection and avoid unnecessary computation.

5.4 AttestLLM Evaluations and Results

5.4.1 Experiment Setup

We evaluate AttestLLM on on-device LLMs, including Llama3 1B, 3B, and 8B [67], Qwen3 4B and 8B [252], and Phi-4 15B [3]. The models are quantized to 8-bit using TorchAO [236] and 4-bit using AWQ [126]. The watermark insertion (Section 5.3.1) is performed on a CPU/GPU server with 4 NVIDIA RTX 6000 GPUs. We use C4 [190] as D_{pro} and 64

samples from SimCSE dataset [61] as D_{tri}. In pre-quantization, we use the Adam optimizer with learning rate η_{pre} to 6e-6, regularization term α set to 1e-3, and optimize for 20 epochs. In post-quantization, the perturbation strength p is set to 2 and 20 for INT4 and INT8 quantization, η_{post} is 0.5 and 0.1 for INT4 and INT8 quantization, and optimize for 40 epochs.The signature length $|B|$ is 20 $\times|M|$. The online attestation(Section 5.3.2) is performed on an 8-core ARM CPU (4×Cortex-A76 + 4×Cortex-A55) with 16GB LPDDR4X memory and Android 13 as host OS. LLM inference runs in the REE, while attestation executes in a pKVM-protected enclave allocated 512MB of memory and running a minimal Microdroid kernel [65], configured with 4 TEE threads. For online attestation, we set $(|M|, k)$ to Llama3-1B $(16, 2)$, Llama3-3B $(28, 2)$, Llama3-8B $(32, 4)$, Qwen3-4B/8B $(36, 4)$, and Phi4-15B $(40, 6)$. We use a default attestation interval of $f = 100$ tokens and verify 16 random samples from D_{tri} per attestation round.

Evaluation Criteria We evaluate AttestLLM with: (1) **Watermark Extraction Rate** (WER): The percentage of watermark successfully extracted; (2) **Attest Pass Rate** (APR): The percentage of layers passing attestation (i.e., layer-wise WER = 1); (3) **Watermarked LLM Performance**: *Perplexity (PPL)* to evaluate generated text fluency on WikiText Dataset [155], and, *Zero-shot Accuracy (Zero-shot Acc)* for next-token prediction evaluation on the mean of LAMBADA, HellaSwag, PIQA, and WinoGrande Datasets [60]; (4) **Attestation Overhead**: *Latency overhead (%)* is the additional time required to generate one token compared to pure REE inference, and *Energy overhead (%)* is the additional energy consumed to generate one token compared to pure REE inference.

Baselines AttestLLM is the first attestation framework to protect on-device LLM's hardware IP. Nevertheless, we compare it with *TEE-shield inference* [272, 138] that enforces model integrity and protects hardware IP by partitioning the LLM to fit into secure memory and executing segment by segment inside the TEE.

Note that (i) we do not compare with DeepAttest [31], as it is designed for Intel SGX [40] rather than Arm TrustZone. Besides, it encodes watermarks when training the DNNs from scratch, which cannot be easily scaled to LLMs [266]; (ii) watermarking frameworks with intensive

verification overheads are excluded because attestation requires a co-design of the watermarking algorithm and TEE system optimizations. Methods that perform full-model inference [116, 115] to check pre-encoded backdoors or require heavy extraction computation [266] are therefore unsuitable for attestation.

5.4.2 Watermarking On-device LLMs

Table 5.1. AttestLLM's performance in watermarking on-device LLMs quantized to INT4 and INT8. $\bar{\Delta}$ is the average LLM performance degradation (PPL for perplexity; Zero-shot Acc for zero-shot accuracy) compared with non-wm LLMs.

Method	Metrics	INT4 Quantization							INT8 Quantization						
	Model	Llama 3			Qwen 3		Phi-4	$\bar{\Delta}$	Llama 3			Qwen 3		Phi-4	$\bar{\Delta}$
		1B	3B	8B	4B	8B	15B		1B	3B	8B	4B	8B	15B	
non-WM	PPL ↓	10.84	8.22	6.64	14.88	9.98	6.78	-	9.88	7.88	6.36	13.82	9.63	6.75	-
	Zero-shot Acc ↑	59.59%	67.08%	71.38%	62.09%	66.16%	71.90%	-	60.85%	67.45%	72.11%	62.59%	67.27%	72.89%	-
	WER ↑	55.50%	63.21%	63.68%	79.38%	79.41%	76.92%		71.42%	73.92%	67.71%	76.61%	81.33%	74.44%	
	APR ↑	0%	0%	0%	0%	0%	0%		0%	0%	0%	0%	0%	0%	
AttestLLM	PPL ↓	10.92	8.24	6.66	15.07	9.96	6.85	0.62%	9.92	7.89	6.37	13.72	9.62	6.76	0.07%
	Zero-shot Acc ↑	59.49%	67.10%	71.18%	61.97%	66.18%	71.89%	0.09%	61.31%	67.59%	72.04%	62.50%	67.66%	72.70%	0.15%
	WER ↑	100%	100%	100%	100%	100%	100%		100%	100%	100%	100%	100%	100%	
	APR ↑	100%	100%	100%	100%	100%	100%		100%	100%	100%	100%	100%	100%	

Attestation reliability We encode watermarks into on-device LLMs from Llama, Qwen, and Phi families following Section 5.3.1 and show the results in Table 5.1. The attest pass rate (APR) reflects whether each layer's watermark can be correctly extracted. If any layer fails to achieve a 100% layer-wise WER (i.e., APR lower than 100%), the system flags the mismatch and aborts REE inference. LLMs watermarked with AttestLLM can reliably pass the attestation with 100% WER and 100% APR, whereas the execution of non-watermarked ones will be aborted due to low WER and 0% APR.

Watermarked LLM fidelity Encoding watermarks onto LLMs will not degrade their quality and utility. We evaluate AttestLLM's performance on text generation fluency using perplexity and next-token prediction capability using zero-shot accuracy. As shown in Table 5.1, for INT4 quantization, compared to non-watermarked models, AttestLLM averagely degrades the Perplexity (PPL) by 0.62% and Zero-shot Accuracy by 0.09%. As for INT8 quantization, compared to non-watermarked models, AttestLLM degrades the PPL and Zero-shot Accuracy by 0.07% and 0.15%, respectively. These degradations are minimal and do not produce noticeable

differences when users interact with the watermarked models [126].

5.4.3 Attestation Overhead

As shown in Figure 5.3, we report the attestation latency and energy overhead of AttestLLM and TEE-shield inference compared to pure REE inference. TEE-shield inference suffers from substantial overhead, with average latency overhead of 472.8% for INT4 and 626.3% for INT8, and average energy overhead of 87.8% for INT4 and 247.2% for INT8. In contrast, AttestLLM imposes significantly lower costs over REE execution, with average overheads of 9.3% (INT4) and 8.3% (INT8) in terms of latency, and 7.3% (INT4) and 7.0% (INT8) for energy. As such, AttestLLM achieves an average of 62× lower latency overhead 30× lower energy overhead compared to TEE-shield inference.

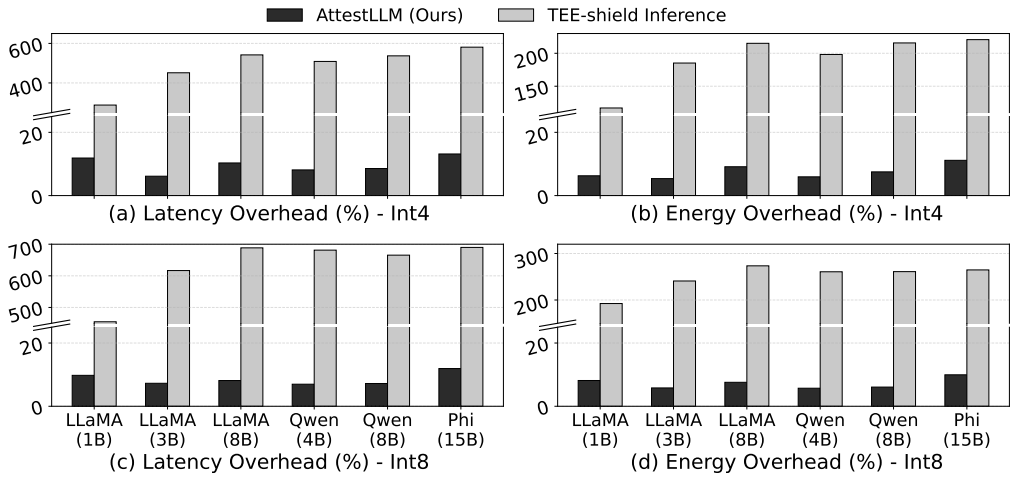


Figure 5.3. Attestation latency and energy overhead (%) of LLMs quantized into INT4 and INT8. Lower is better.

5.4.4 Attestation Strength

The attestation is performed while ensuring sufficient attestation strength with a low evasion probability P_{evasion}. For a conversational session producing 1000 tokens, with an attestation interval of f=100, AttestLLM will conduct r=10 attestation rounds. Given that the model weights are stored in shuffled memory regions, we assume an adversary performs a brute-force

search and tampers with 10% of transformer layers in a Qwen3-8B model, which is 4 out of 36 layers. With Equation 5.4, the evasion probability per attestation is 6.1×10^{-1}, and over a single conversational session, P_{evasion} drops to 7.1×10^{-3}. It ensures AttestLLM to confidently detect and abort the execution of unauthorized LLMs.

5.4.5 Robustness

We evaluate AttestLLM's robustness by performing the following attacks: (i) watermark forgery attack, and (ii) model replacement attack; (iii) TEE system attack. We use Llama3-8B [67] model quantized to INT8 as the target model. We assume the adversary has prior knowledge of the watermarking algorithm, but do not have access to the encrypted trigger dataset and watermark keys. They can attempt a brute-force search to tamper with certain layers in the shuffled memory region, but locating and modifying all layers would be prohibitively expensive.

Watermark forgery attack The adversary aims to host an unauthorized LLM but bypass the attestation by fooling the verification pipeline to decode the device-specific signatures. To achieve this, he/she needs to (i) know the watermark matrix WM and watermark signature B, which are kept confidential and encrypted; (ii) brute-force search all of the LLM's parameters in shuffled and randomized memory, and replace them with the malicious copy. All of this information can be expensive to search and recover, which ensures AttestLLM is resistant to watermark forgery attacks.

Model replacement attack In this setting, we consider two types of model replacement attacks (MPA): (i) partial model replacement attack; and (ii) full model replacement attack. In both cases, the adversary attempts to perform brute-force search and replaces model parameters with malicious copies. Since an exhaustive search in the shuffled memory region can be time-consuming, we consider partial model replacement as a more realistic threat, where $k = 4$ out of $|M| = 32$ layers are tampered. The full model replacement attack is a stress test to evaluate AttestLLM's attestation reliability when all of the parameters are tampered. As in Table 5.2,

we use zeroth-order optimization to fine-tune INT8 quantized Llama3-8B [67] model on the ToxicChat [128] dataset. For partial MPA, we only update the parameters of four randomly selected layers; for full MPA, we update the entire model. In the partial MPA case, four tampered layers fail attestation (i.e., layer-wise WER is not 100%), resulting in an attest pass rate of 87.50%. In such cases, the evasion probability after 10 rounds of attestation becomes $P_{\text{evasion}} = 3.5 \times 10^{-3}$, which ensures AttestLLM to confidently abort the execution of tampered LLM. In the full MPA case, all layers fail attestation because updating all of the model parameters alters their decoded watermarks. These results demonstrate that AttestLLM reliably detects both partial and full MPA, ensuring that unauthorized or tampered LLMs are blocked from execution.

Table 5.2. Model replacement attack (MPA) performance.

Attack	WM LLM (benign)	Partial MPA	Full MPA
WER ↑	100%	94.06%	84.84%
APR ↑	100%	87.50%	0%

TEE system attack In TEE system attack, the adversary uses device resources to host an illegitimate LLM in REE. He/She aims to bypass attestation by (i) halting the interaction with TEE; (ii) dispatching the watermarked LLM solely for attestation. They can be mitigated by these system-level security efforts: (i) during platform bootstrapping, the hardware root-of-trust verifies the integrity of the bootloader, host kernel image, and device tree, preventing the injection of malicious software before the system starts; (ii) A secure timer inside the TEE tracks attestation intervals, which cannot be tampered with by the adversary. If the TEE does not receive attestation interactions within the expected time window, the hypervisor halts the application executions in REE, preventing the adversary from running a separate LLM without TEE involvement.; (iii) The attested layer selection is generated randomly and securely inside TEE, which prevents the attacker from manipulating layer checking and bypassing the attestation of tampered layers; (iv) The hypervisor monitors OS-level page allocations to ensure that the executed model matches the attested one. This prevents the adversary from running an illegitimate LLM while dispatching

the watermarked model only for attestation.

5.4.6 Ablation Studies and Analysis

Watermark before/after quantization As in Table 5.3, when watermarking Llama3-1B [67] model quantized to INT4, post quantization WM insertion introduces more degradation because zeroth-order optimization must approximate gradients from a discrete weight distribution, which requires larger weight updates than gradient-based optimization. Pre-quantization WM insertion causes less quality degradation but suffers from reduced extraction accuracy, since quantization perturbs the activation outputs and weakens the watermark signal. By combining both, AttestLLM preserves both model quality and verification reliability.

Table 5.3. Watermark insertion before/after quantization.

WM Insertion Stage	PPL ↓	Zero-shot Acc ↑	WER ↑	APR ↑
Before Quantization	10.26	59.58%	97.19%	87.50%
After Quantization	10.84	58.51%	100%	100%
AttestLLM	9.91	60.90%	100%	100%

Attestation overhead with/without efficiency optimization Without optimization, secure copy and watermark verification execute sequentially, leaving TEE threads idle and thereby increasing attestation overhead. As shown in Table 5.4, applying pipelined secure copy time overlapping and parallel LLM layer verification significantly improves efficiency. For Qwen3-4B quantized to INT8, AttestLLM's latency overhead decreases from 8.9% to 6.9%, and its energy overhead drops from 7.2% to 5.7%, showing that these optimizations effectively reduce attestation cost.

Table 5.4. Attestation overhead with/without optimizations.

Efficiency Optimization	Latency ↓	Energy ↓
no Optimization	8.9%	7.2%
with Optimization	6.9%	5.7%

Attestation overhead with (f, k) choices f and k are tunable hyperparameters that balance

attestation strength and efficiency. We show the quantitative results for Qwen3-4B quantized to INT8 in Fig. 5.4. A larger f increases the attestation interval, reducing overhead but weakening security. Similarly, a smaller k means fewer transformer layers are verified per round, lowering overhead but also reducing attestation strength. Thus, we choose $f = 100$ and $k = 4$ for a reasonable trade-off between efficiency and security.

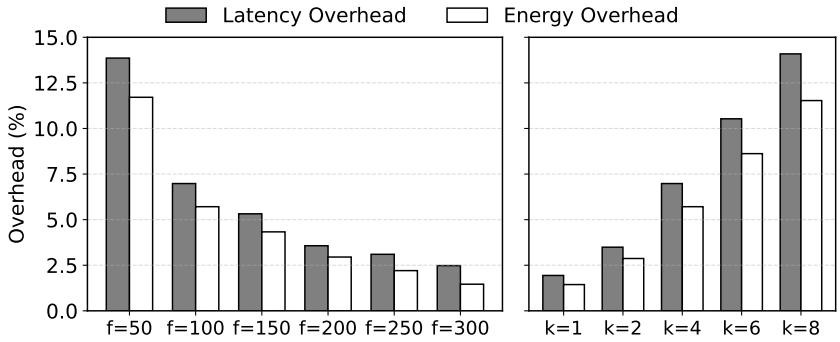


Figure 5.4. Attestation overhead with different (f, k) choices.

5.5 Conclusion

This chapter presents AttestLLM, the first attestation framework that co-designs algorithm/software/hardware to protect the hardware IP of device vendors with efficiency, scalability, and robustness. In offline watermarking, AttestLLM adaptively encodes device-specific signatures onto each LLM layer while preserving model quality and robustness. Online attestation securely verifies the watermarks inside TEE and employs several system-level optimizations to enlarge secure memory and reduce attestation overhead. Comprehensive evaluations of on-device LLMs demonstrate AttestLLM's fidelity, robustness, attestation reliability and efficiency.

5.6 Acknowledgements

Chapter 5 is a reprint of the material as it appears in: Ruisi Zhang, Yifei Zhao, Neusha Javidnia, Mengxin Zheng, and Farinaz Koushanfar, "AttestLLM: Efficient Attestation Framework

for Billion-scale On-device LLMs," in *ACM/IEEE Design Automation Conference*, 2026. The dissertation author was the primary investigator and author of this paper.

Part III

Intellectual Property Protection for Human–AI Co-Creations

Chapter 6

Scalable Combinatorial Optimization Artifact IP Protection

Combinatorial optimization [208, 105, 244], a fundamental paradigm in operations research and computer science, has a wide range of applications in various fields, including scientific discovery [43, 97], chip design [77, 158], synthetic biology [99, 166], etc. The optimization engines (solvers) try to find the best solution from a combinatorially large number of possibilities that adhere to problem constraints. Due to the vast number of potential candidates and the complexity of the problem constraints, exhaustive search and evaluation are infeasible and involve substantial computational overhead [78]. Thus, most common solvers use sophisticated heuristic-based algorithms [39, 218], or rely on emerging machine learning-based methods [152, 19, 209, 78] to explore the search space for quality solutions efficiently. Nonetheless, domain expertise is still required for solution validations and refinements to ensure performance in target scenarios [56, 226, 183]. Protecting the costly, high-quality solutions to complex combinatorial optimization problems is an open and pressing challenge.

Watermarking has emerged as a promising solution for securing such valuable IP artifacts [47] by embedding invisible ownership signatures into solutions that can later be used to prove authorship. However, existing efforts in combinatorial optimization watermarking remain narrow in scope and lack cross-domain generalizability. Early work in this area [243, 214, 153]

encodes signatures by imposing additional watermarking constraints on problem variables or constraint functions. These methods are typically tailored to a specific set of problems (e.g., MaxCut or SAT) and rely on enumerating satisfied watermarking constraints for verification. As such, the encoded watermarks are vulnerable to variable name transformation attacks and removal attacks that can significantly alter the solution. More recent approaches are tightly coupled with specialized solvers and domain-specific applications, such as protein folding [271] and chip design [267]. While these methods incorporate extensive domain knowledge and customized toolchain optimizations, they fall short of addressing the broader challenge of developing universal and transferable watermarking techniques for combinatorial optimization problems.

The first set of challenges lies in the absence of versatile formulations that can abstract watermarking objectives and support verification with cross-domain adaptability. Meanwhile, the ownership proofs must be embedded without compromising the quality of the underlying solutions. The second set of challenges arises with the recent advances in combinatorial optimization solvers [158, 78], which tackle large-scale problems through machine learning-based algorithms and distributed training paradigms to efficiently explore vast search spaces in parallel. In such settings, distributed watermarking methods are critical for scalable intellectual property protection. Moreover, the encoded watermarks shall demonstrate resilience against potential adversarial attempts to erase signatures, while preserving the integrity and quality of the solution.

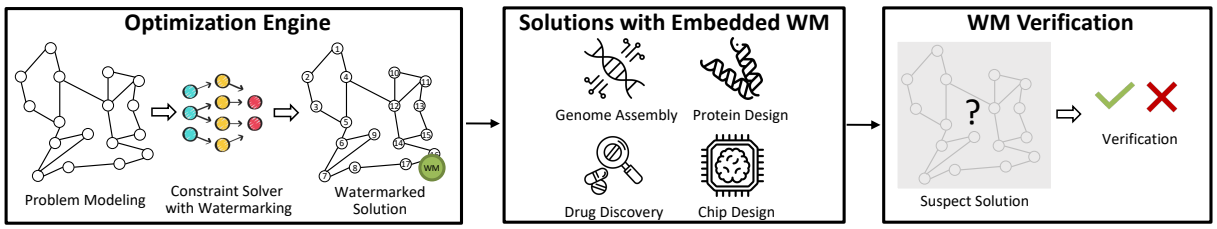


Figure 6.1. Overview of tracing constrained combinatorial solutions with OptiMarks. High-quality solutions to combinatorial optimization problems are valuable assets in domains such as biotechnology and chip design. OptiMarks embeds watermarks into these solutions to safeguard solver owner's intellectual property, trace solution distribution, and verify watermark provenance in a scalable and robust manner.

To address these challenges, this chapter introduces OptiMarks, a universal watermarking

framework to scalably and robustly embed ownership signatures into combinatorial optimization solutions. The overall flow is depicted in Figure 6.1. In the watermark insertion stage, we first model the constrained combinatorial optimization problem as a graph, with nodes as variables and edges as constraints. Nodes are then labeled using a graph isomorphism network (GIN) [250], which leverages graph structural information to yield unique and reliable identifiers even under malicious transformations. The watermark constraints, represented as an additional set of edges on the variable nodes, are jointly optimized with the original problem constraints. The watermarks are designed with three key properties: (i) Secrecy: construct by randomly selecting node pairs that are not connected by an edge in the original problem graph, making them difficult for adversaries to forge; (ii) Uniqueness: provide statistically significant evidence of the watermark presence in the solution, thereby enabling faithful ownership verification; (iii) Quality-preserving: ensure that the introduction of additional constraints does not degrade the solution quality. Afterwards, OptiMarks trains a watermark decoder using adversarial training to robustly extract a predefined error correction code [17] signature from both the watermarked solution and its adversarially modified variants. Once the solution is finalized, OptiMarks removes the watermark constraints before distributing it to the end users.

To verify ownership of a suspicious solution, OptiMarks first relabels the graph with the same GIN as in the insertion stage, and restores the watermark constraints. Next, a trained watermark decoder extracts the signature and compares it with the encoded ones for ownership verification. To accommodate the ever-growing scale of combinatorial optimization problems, OptiMarks employs a distributed watermarking approach, where watermark constraints are assigned to subgraphs partitioned across servers. These watermarks are embedded in parallel during the optimization process, and the watermark decoder is trained to extract the predefined messages from each server's subgraph. Thus, OptiMarks effectively protects the intellectual property of the large-scale solution in the distributed optimization settings. In summary, the contributions are as follows.

- We present OptiMarks, the first scalable and universal watermarking framework for constrained combinatorial optimization solvers, which protects the intellectual property and traces the distribution of resulting solutions.
- We highlight the novelty of OptiMarks in (1) **Quality preservation**: signature insertion does not degrade the performance of the watermarked products; (2) **Scalability**: supports watermarking for ultra-large optimization problems in distributed environments; (3) **Robustness**: resistance toward diverse attacks aimed at watermark removal.
- We conduct extensive experiments on various benchmark instances demonstrating: (1) OptiMarks successfully encodes watermarks onto a wide range of real-world combinatorial optimization problems, including protein prediction, drug discovery, and social recommendation; (2) OptiMarks achieves 100% watermark extraction accuracy while incurring only $\sim$0.36% performance degradation; (3) OptiMarks demonstrates resiliency under various malicious transformations and maintains over 80% watermark extraction rates.

6.1 Problem Formulation

6.1.1 Problem Formulation

Equation (6.1) shows a combinatorial optimization problem with an arbitrary cost function $f(x)$ over $x_i,\ i \in \mathcal{N}$, where $\mathcal{N} = \{1, \cdots, N\}$ are the indices of the optimization variable. The variables are constrained by K constraint functions, $c_k^K(x_{\mathcal{N}_k}),\ k \in \mathscr{K}$, where $\mathscr{K} = \{1, \cdots, K\}$ is the constraint indices. The watermark is encoded by enforcing an additional set of watermarking constraints, $c_j^{WM}(x_{\mathcal{N}_j}),\ j \in \mathscr{WM}$, where $\mathscr{WM}$ is the index set of the watermarking constraints. The original constraints c_k^K, together with the watermarking constraints c_j^{WM}, are mapped by the watermarking function f_{wm} into the watermarking space, a representation space designed for embedding and detecting watermarks, such that their transformed form remains close to the target watermark signature sequence M.

$$\min_{x_i, i \in \mathcal{N}} \quad f(x) \tag{6.1a}$$

$$s.t. \quad c_k^K(x_{\mathcal{N}_k}) \leq 0, \text{ for } k \in \mathcal{K} \tag{6.1b}$$

$$c_j^{WM}(x_{\mathcal{N}_j}) \leq 0, \text{ for } j \in \mathcal{W}\mathcal{M} \tag{6.1c}$$

$$f_{wm}(c^K, c^{WM}, M) \leq 0 \tag{6.1d}$$

$$x_i \in \{d_0, \cdots, d_v\}, \text{ for } i \in \mathcal{N} \tag{6.1e}$$

Solving the above combinatorial optimization problem is inherently challenging, as it often falls into the class of NP-hard problems due to discrete decision variables and potentially non-convex, non-linear objective and constraint functions. Previous work [78, 209] addresses this challenge by exploiting the structural properties of the problem and modeling it as a constraint (hyper)graph, where each optimization variable corresponds to a node and each constraint is represented as a (hyper)edge connecting the involved variables. This formulation captures high-order dependencies among variables and enables the use of (hyper)graph neural networks to learn effective and scalable solution strategies.

6.1.2 Watermarking Principles

To ensure the watermarked solution can be usable for real-world applications, the watermarking algorithm design follows these principles:

- **Quality-preserving**: the watermark insertion shall not degrade the final products' quality and incur minimal overhead to the optimization process;
- **Scalable and Efficient**: the watermark insertion and verification shall be efficient and scale to watermark ultra-large solutions;

- **Robustness**: the watermarked product shall withstand malicious transformations and removal attempts from the adversaries;
- **Integrity**: the watermark shall only appear in the watermarked product and not provide ownership proof for the non-watermarked ones.

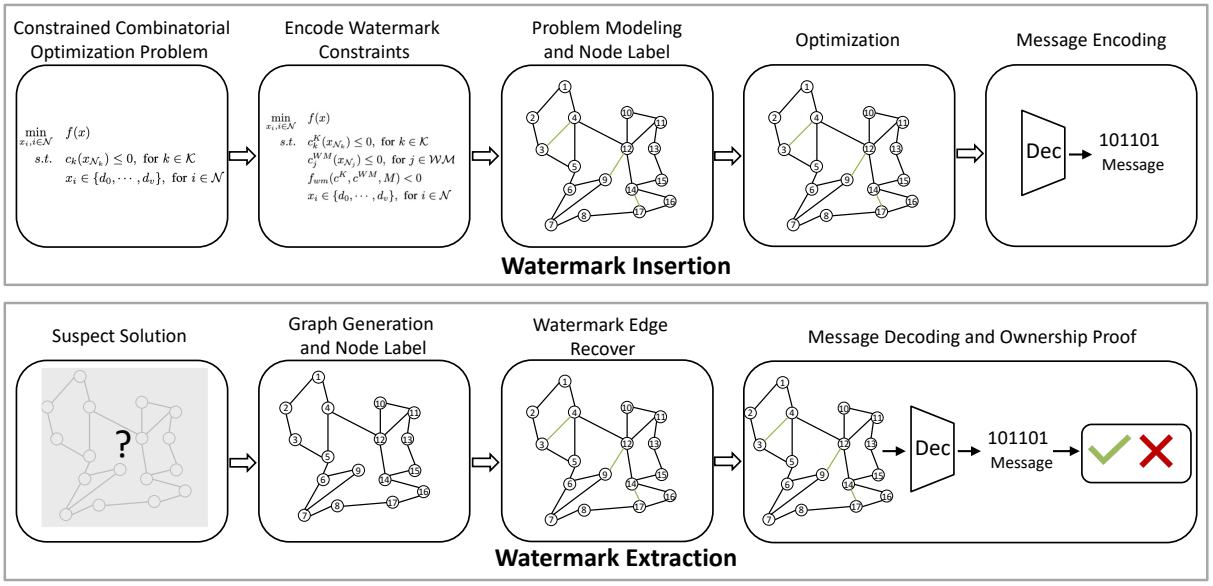


Figure 6.2. Overview of the constraint combinatorial optimization watermarking process. It consists of two stages, namely, watermark insertion and watermark extraction. OptiMarks first assigns the label to each node via its structural information. Green edges are added as the additional watermark constraints for the solver to satisfy to encode the owner's signatures. Then, a decoder is trained to decode the watermark messages from the watermarked solutions. Given a suspect solution, the owner leverages a decoder to decode embedded messages from the layout and compares them to the encoded ones for ownership proof.

6.2 Encode Robust Signatures with OptiMarks

The overview of OptiMarks is shown in Figure 6.2. In this section, we introduce the watermark insertion process, which is designed to (i) preserve solution fidelity while providing strong watermarking strength with provable constraint thresholds, and (ii) remain resilient to attacks through structure-aware node labeling against variable renaming attacks and adversarial training with error-correction code against removal attacks that significantly alter the solution.

6.2.1 Structure-aware Node Label

The objective is to assign each node a corresponding feature vector derived solely from the graph's structural information. This enables reliable node identification and signature decoding without relying on external information that can be easily forged or removed. Given a graph $\mathrm{G} = \{c^K\}$, we first construct node features F by encoding each node's degree and the degrees of its neighbors into a $|F|$-dimensional vector and sort them in descending order. During the construction, if the resulting vector is shorter than $|F|$, it is padded with zeros; if the vector is longer, it is truncated to length $|F|$.

With node features F, OptiMarks assigns each node a canonical and structure-intrinsic identifier through a Graph Isomorphism Network (GIN) [250]. GIN evaluates node significance by computing unique structural scores, analogous to the Weisfeiler–Lehman (WL) [216] test for graph isomorphism, while offering better scalability and computational efficiency. The GIN takes node features F as input. As in Equation 6.2, the node score for the v-th node is $h_v^{(k-1)}$ at the $k-1$-th layer. A learnable parameter ε and an $MLP^{(k)}$ layer are used to learn the mapping from $k-1$-th layer score $h_v^{(k-1)}$ to k-th layer score $h_v^{(k)}$ using $h_v^{(k-1)}$ and its neighboring node u's score $h_u^{(k-1)}$.

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\varepsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right) \tag{6.2}$$

The final score h_v of each node is unique and is hashed to a distinct value using SHA-1 [197]. We rank the resulting hash values in ascending order and assign the corresponding rank as the node's index label. Because the GIN parameters are kept confidential by the owner, adversaries cannot easily reproduce these node hash values for forgery.

6.2.2 Robust Watermark Insertion

As in Section 6.1.1, the watermark constraints c^{WM} are incorporated into the final solution

as an additional set of problem constraints. We first construct c^{WM} by randomly selecting pairs of nodes from G that are not included in the original constraint set c^K. The solver then jointly optimizes over both the original constraints c^K and the watermark constraints c^{WM} to obtain the final solution.

The solution is robustly projected into the watermark space by the transformation function f_{wm}, which encourages it to align closely with a predefined message M. As such, OptiMarks can decode robust signatures from the overall solution, rather than by enumerating satisfied watermark constraints as in prior work [243, 214, 153]. This design makes OptiMarks more resilient to removal attacks that substantially modify the solution. We leverage two mechanisms, namely, error correction codes [17] and adversarial training [265], to ensure reliable watermark recovery even under malicious tampering.

❶ *Error Correction Codes*: During adversarial tampering, part of the decoded message can be distorted and results in reduced watermark extraction rates. Error correction codes [44, 29] introduce controlled redundancy in message generation and enables message recovery even when a subset of the watermark bits is altered.

OptiMarks first defines a binary message $M = \{M_1, M_2, .., M_i\}$ ($M_i \in \{0,1\}$). This message is then encoded using a Bose–Chaudhuri–Hocquenghem (BCH) code into a longer codeword $\hat{M}$ with $|\hat{M}| \geq |M|$. As show in Equation 6.3, G is the generator matrix derived from the BCH generator polynomial. This matrix expands the $|M|$-bit message into a $|\hat{M}|$-bit redundant codeword by introducing structured parity bits. All operations are performed in the binary field $GF(2)$. d_{dmin} is the minimum Hamming distance between any two valid codewords, which determines the error-correction capability of the code. With $d_{\min} \geq 2t+1$, the generated code $\hat{M}$ can correct up to t bit errors.

$$\hat{M} = M \cdot G, \quad G \in \{0,1\}^{|M| \times |\hat{M}|}, \quad d_{\min} \geq 2t+1. \tag{6.3}$$

❷ *Adversarial Training*: Here, we leveraged a GNN-based message decoder D to project the

solution on graph G into the watermark space and align it with the encoded message $\hat{M}$. Similar to Section 6.2.1, OptiMarks aggregates each node's solution together with those of its neighbors to construct a solution feature vector X. D learns to recover the encoded message $\hat{M}$ from the watermarked graph $\mathrm{G}(c^K, c^{WM}, X)$ by minimizing the L_1 loss as shown in Equation 6.5. To enhance robustness against malicious solution tampering, OptiMarks introduces adversarial noise to the solution feature X and generates a perturbed $\tilde{X}$ that mimics such scenarios as in Equation 6.4, where δ denotes bounded adversarial perturbations and ε controls the perturbation magnitude.

$$\tilde{X} = X + \delta, \quad \|\delta\|_1 \leq \varepsilon, \tag{6.4}$$

D is jointly trained on both clean and perturbed features, enabling it to reliably decode the watermark even when the solution is partially modified. The overall training objective is given by Equation 6.5, where $G = \mathrm{G}(c^K, c^{WM}, X)$ and $\tilde{G} = \mathrm{G}(c^K, c^{WM}, \tilde{X})$ denote the clean and perturbed graph representations, respectively, and w_w and w_t balance the two objectives.

$$\begin{aligned} &\min_{\mathrm{D}} \; L_{\hat{M}}(\hat{M}, \mathrm{D}(G), \mathrm{D}(\tilde{G})) = \\ &w_w \textstyle\sum_{i=1}^{|\hat{M}|} \left|\hat{M}_i - \mathrm{D}(G)_i\right| + w_t \sum_{i=1}^{|\hat{M}|} \left|\hat{M}_i - \mathrm{D}(\tilde{G})_i\right|, \end{aligned} \tag{6.5}$$

These keys are stored for later verification: (i) the watermark constraints, including the structure-aware node IDs used to restore the watermarked graph $\mathrm{G}(c^K, c^{WM}, X)$; and (ii) the learned decoder D, the signature sequence $\hat{M}$, and the random seeds required for watermark decoding.

6.2.3 Provenance Guarantee

We analyze the statistical detectability of the embedded watermark by formulating ownership verification as a hypothesis testing problem that bounds the number of watermark constraints and the decoded signature length.

The following binary hypothesis test is used for verification

$$H_0 : \text{The solution is not watermarked,}$$

$$H_1 : \text{The solution contains the watermark.}$$

❶ *Encoded watermark constraints* For a graph G with N nodes (optimization variables) and K edges (problem constraints), watermark constraints is defined as $c^{WM} = \{c_1^{WM}, \ldots, c_m^{WM}\}$. Let

$$k = \sum_{i=1}^{m} 1\left[c_i^{WM} \text{ is satisfied in G}\right]$$

denote the number of watermark constraints satisfied by coincidence in G. Under the null hypothesis H_0 that the graph is generated without any watermark embedding, the watermark constraint has a coincidence probability

$$p = \frac{K}{\binom{|N|}{2}}.$$

Under the approximation that $c_1^{WM}, \ldots, c_m^{WM}$ behave as independent Bernoulli random variables with success probability p when the problem constraint size $K \gg m$, the total number of coincidental matches satisfies

$$k \sim \text{Binomial}(m, p)$$

under H_0. The one-sided p-value for observing at least k coincidental matches is then given by

$$p\text{-value} = \Pr_{H_0}[K \geq k] = \sum_{j=k}^{m} \binom{m}{j} p^j (1-p)^{m-j},$$

which quantifies the probability that k or more watermark constraints are satisfied purely by random chance. A watermark is declared present if this p-value falls below a pre-specified significance level α (e.g., $\alpha = 0.05$), indicating that the observed degree of constraint satisfaction

is statistically unlikely under H_0.

❷ *Encoded signature length*: Let the message $M = (m_1, m_2, \ldots, m_{|M|}) \in \{0,1\}^{|M|}$ denote the decoded signature from the decoder D. which are generated as independent random variables $m_i \sim \text{Bernoulli}(1/2)$. Let k denote the number of decoded bits that match the target signature, i.e.,

$$k = \sum_{i=1}^{|M|} 1[m_i = m_i'],$$

where m_i' is the i-th target bit in decoded message M'. Under H_0, k follows a binomial distribution,

$$k \sim \text{Binomial}(|M|, 1/2).$$

The statistical significance of the extracted watermark is quantified by the p-value, defined as the probability of observing at least k matches under H_0:

$$p\text{-value} = \Pr_{H_0}[K \geq k] = \sum_{j=k}^{|M|} \binom{|M|}{j} \left(\frac{1}{2}\right)^{|M|}. \tag{6.6}$$

A watermark is declared present if the computed p-value is below a predefined significance level α (e.g., $\alpha = 0.05$), indicating that the observed agreement between M and the target signature is unlikely to occur by random chance.

6.3 Verify Signatures from Suspect Solutions

OptiMarks's ownership verification is shown in Figure 6.2. First, the nodes are assigned canonical indices based on graph structural information following Section 6.2.1, yielding $\text{G} = \{c^K\}$. With the stored watermark keys, the corresponding watermark constraints are then incorporated to construct the verification graph $\text{G} = \{c^K, c^{WM}, X\}$. Next, the trained decoder D extracts the error correction code $\hat{M}'$ from G. To recover the original message, $\hat{M}'$ is first multiplied by the corresponding parity-check matrix $H \in \{0,1\}^{(|\hat{M}|-|M|)\times|\hat{M}|}$ to compute the

syndrome vection S following Equation 6.7, where all operations are performed over GF(2). The syndrome vector S characterizes the error pattern, where a non-zero value indicates a corrupted bit during distribution.

$$S = H \cdot (\hat{M}')^{\top} \tag{6.7}$$

With the syndrome vector S, the Berlekamp–Massey algorithm [205] is employed to iteratively process S and solve the key equation in Equation 6.8 for a minimal error-locator polynomial $\Lambda(x)$ that satisfies the algebraic relationship between $S(x)$, $\Lambda(x)$, and the error-evaluator polynomial $\Omega(x)$. Then, a structured finite-field Chien search [35] is used to identify the indices of erroneous bits in $\hat{M}'$ by evaluating $\Lambda(x)$ at all possible field elements to find its roots. The decoder corrects those bits by flipping their values to obtain the corrected codeword $\tilde{M}'$, and finally recovers the original message M' by discarding the parity coordinates.

$$\Lambda(x)\, S(x) \;\equiv\; \Omega(x) \pmod{x^{2t}} \tag{6.8}$$

The watermark extraction rate (WER) is computed to quantify the similarity between the original signature M and the decoded signature M' following Equation 6.9.

$$\%WER = 100 \times \frac{\sum_{i=1}^{|M|} 1[m_i = m'_i]}{|M|} \tag{6.9}$$

6.4 Distributed Watermark for Large-scale Solutions

As the constrained combinatorial optimization problems' size and complexity grow, distributed training is introduced to partition the global problem into smaller subproblems that can be processed in parallel across multiple servers [78], with each server i responsible for a subset of constraints denoted as $\mathrm{G}_i(c_i^K)$. In this setting, watermarking is applied independently to each local subset during optimization, and thus protects the integrity of the overall combinatorial

solution across the distributed system. This distributed watermarking mechanism ensures that the global solution retains ownership information even when it is assembled from locally computed artifacts.

6.4.1 Distributed Watermark Insertion

We demonstrate the distributed watermarking insertion process in Algorithm 4. Given a graph $\mathrm{G}(c^K)$ and a random seed d, the global graph is first partitioned into N subgraphs $\{\mathrm{G}_i(c_i^K)\}_{i=0}^{N-1}$, each of which is assigned to a corresponding server for processing. On each server, additional watermark constraints are encoded into the local subgraph, yielding $\mathrm{G}_i(c_i^K, c_i^{WM})$. These augmented constraint sets are then jointly solved by the distributed solver, which produces solution components X_i for each subgraph. The watermarked subgraphs are denoted as $\mathrm{G}_i(c_i^K, c_i^{WM}, X_i)$, which collectively embed ownership information into the distributed solution.

To encode a pre-defined message M_i to the i-th server, we first expand M_i into a longer, redundant codeword $\hat{M}_i$ using a binary BCH code following Section 6.2.2. With the BCH code $\hat{M}_i$, we train a watermark decoder D jointly over the solution subset $\mathrm{G}_i(c_i^K, c_i^{WM}, X_i)$ so that it learns a mapping from the augmented subgraph to the encoded codeword $\hat{M}_i$. The message $\{M_i\}_{i=0}^{N-1}$ on each subproblem, the jointly trained decoder D, the encoded watermark constraints $\{c_i^{WM}\}_{i=0}^{N-1}$ and the random seed d are saved as the watermark key.

6.4.2 Distributed Watermark Verification

To decode the watermark from a suspect solution, OptiMarks first reconstructs the global graph $\mathrm{G}(c^K, c^{WM}, X)$ by adding watermark constraints c^{WM} back to $\mathrm{G}(c^K)$. Then, OptiMarks partitions it into subgraphs using the same random seed d used during insertion. For each subgraph $\mathrm{G}_i(c_i^K, c_i^{WM}, X_i)$, the trained decoder D decodes the corresponding encoded codeword $\hat{M}_i'$. Following Section 6.3, $\hat{M}_i'$ is then passed through the BCH error-correction decoder to recover the original message M_i'. Finally, the recovered message $\{M_i'\}_{i=0}^{N-1}$ is compared with the original embedded message $\{M_i\}_{i=0}^{N-1}$ to establish ownership.

Algorithm 4: Distributed Watermark Insertion

Input: Problem constraints as a graph $\mathrm{G}(c^K)$; Combinatorial solver Solver; number of servers N; number of epochs EP; predefined messages $\{M_i\}_{i=0}^{N-1}$.

Output: Watermarked solutions $\{\mathrm{G}_i(c_i^K, c_i^{WM}, X_i)\}_{i=0}^{N-1}$ and trained message decoder D.

Partition $\mathrm{G}(c^K)$ into subgraphs $\{\mathrm{G}_i(c_i^K)\}_{i=0}^{N-1}$ using seed d
for $i \leftarrow 0$ **to** $N-1$ **do**
    // Encode watermark constraints and solve subproblems for each server
    Generate watermark constraints c_i^{WM} for $\mathrm{G}_i(c_i^K)$
    Solve $\mathrm{G}_i(c_i^K, c_i^{WM})$ with Solver
    Output X_i and form watermarked graph $\mathrm{G}_i(c_i^K, c_i^{WM}, X_i)$
for $ep \leftarrow 1$ **to** EP **do**
    // Train decoder D over multiple subsolutions
    for $i \leftarrow 0$ **to** $N-1$ **do in parallel**
        Predict encoded codeword $\hat{M}_i' \leftarrow \mathrm{D}(\mathrm{G}_i(c_i^K, c_i^{WM}, X_i))$
        Compute decoder loss comparing $\hat{M}_i'$ with BCH codeword $\hat{M}_i$
        Compute local gradients for D
    Aggregate gradients for D across servers
    Update and broadcast global decoder parameters
return $\{\mathrm{G}_i(c_i^K, c_i^{WM}, X_i)\}_{i=0}^{N-1}$ *and trained decoder* D

6.5 OptiMarks Evaluations and Results

We verify OptiMarks's efficacy across a spectrum of constrained combinatorial optimization problems, including hypergraph and graph MaxCut, Graph Maximum Independent Set (MIS), and Satisfiability (SAT). Furthermore, we validate the utility through several real-world applications, including protein design, drug discovery, and social recommendation. HypOp [78], a state-of-the-art constrained combinatorial optimization solver, is used as the watermarking target as a proof-of-concept. We perform the watermarking on a CPU/GPU server with 4 NVIDIA RTX 6000 GPUs, 256 GB RAM memory, and Intel Xeon CPUs connected via a 10Gbps Ethernet interface. For each problem instance, OptiMarks chooses the number of watermark constraints so that the corresponding watermark presence achieves a statistical significance of $\alpha = 0.05$. We fix the message length to $|M| = 4$, which yields a random-match p-value of $2^{-4} \approx 0.06$ when all bits match. To provide robustness against noise and partial violations, this 4-bit message

is encoded using a binary BCH error-correcting code that can correct up to $t = 1$ bit errors, resulting in an 7-bit $\hat{M}$ codeword.

OptiMarks is the first general-purpose watermarking toolkit for constraint combinatorial optimization problems. Previous work on optimization solution watermarking can be generally categorized into:

❶ **Edge-Constrained Watermarking** (ECW) [243, 214, 14] embeds ownership information by introducing additional watermark constraints into the optimization problem. Similar to OptiMarks, the solver must satisfy both the original problem constraints and the watermark constraints. However, it lacks a decoding mechanism and can only verify ownership by measuring how many embedded constraints are satisfied, rather than extracting a robust signature.

❷ **Variable-Constrained Watermarking** (VCW) [153, 188] embeds watermarks by introducing auxiliary variables and coupling them with original decision variables. This approach modifies the variable space directly, rather than the constraint structure, by forcing selected variables to satisfy prescribed relationships.

6.5.1 Experimental Results

Watermarking on Different Constrained Combinatorial Optimization Problems For MaxCut in Table 6.1, OptiMarks successfully encoded all of the watermark constraints into the solution and results in 0.35% performance degradation. The watermarked solution is mapped into the watermarking space by f_{wm} and yields 100% watermark extraction rate. In contrast, VCW enforces the label of nodes adhering to the constraints and degrades the quality by 1.06% for a watermark extraction rate of 46%. ECW has a 0.67% performance degradation with an average of 84% watermark extraction rates. It is because the solver may prioritize satisfying other problem constraints instead of the watermark constraints for better watermarked solution quality.

Similarly, for MaxInd in Table 6.2, OptiMarks results in $\sim 0.37\%$ performance degradations, whereas ECW degrades the quality by 0.82% and VCW degrades by 1.99%. The SAT problem is modeled as a hypergraph, in which the nodes are the variables and the hyperedges are

the constraints. ECW and VCW are tailored for graph watermarking and are unable to watermark hypergraphs. Thus, we skip them in Table 6.3. As seen, OptiMarks successfully watermarks the solution with negligible quality degradations.

Table 6.1. OptiMarks watermarking performance on MaxCut problem.

Graph	Nodes	Edges	Solution Quality ↑				Extraction Rate ↑		
			Clean	ECW [243]	VCW [153]	OptiMarks	ECW [243]	VCW [153]	OptiMarks
G14	800	4694	3045	3036	3008	3041	100%	52%	100%
G15	800	4661	3053	3018	2995	3027	100%	35%	100%
G22	2,000	19990	13112	13109	13056	13135	71%	58%	100%
G26	2,000	19990	13303	13103	13154	13225	50%	38%	100%
G70	10,000	9999	9307	9268	9249	9274	100%	45%	100%

Table 6.2. OptiMarks watermarking performance on MaxInd problem.

Graph	Nodes	Solution Quality ↑				Extraction Rate ↑		
		Clean	ECW [243]	VCW [153]	OptiMarks	ECW [243]	VCW [153]	OptiMarks
D3	1,500	633	627	613	629	100%	100%	100%
D3	2,500	1052	1039	1041	1044	100%	100%	100%
D3	3,500	1469	1462	1448	1462	100%	100%	100%
D3	4,500	1896	1889	1888	1889	100%	100%	100%
D5	1,500	460	455	435	460	100%	100%	100%
D5	2,500	862	855	851	858	100%	100%	100%
D5	3,500	1192	1183	1166	1192	100%	100%	100%
D5	4,500	1548	1534	1534	1542	100%	100%	100%

Table 6.3. OptiMarks watermarking performance on SAT problem.

SAT Dataset	Problems	Solution Quality ↑		Extraction Rate ↑
		Clean	OptiMarks	
uf 20-91	100	100	100	100%
uf 100-430	100	100	99	100%
uf 200-860	100	100	100	100%
uf 250-1065	100	98	97	100%

Watermarking Real-World Applications In protein structure prediction, MaxInd is used to help identify residues that could be mutated simultaneously without direct structural interference in directed evolution or rational design [166]. In Table 6.4, we use the protein structure DD dataset[161], where nodes are amino acids and edges represent the spatial proximity between them, as a proof-of-concept showing OptiMarks's watermarking performance. As seen, by

enforcing the real nodes' label to follow the virtual nodes, solutions watermarked by VCW, on average, degrade the performance by 2.61% and have an extraction rate of 48%. Both ECW and OptiMarks achieve 100% watermark extraction rates and degrade the solution quality by only 0.70% and 0.66%, respectively. Thus, both methods maintain the watermarking fidelity and can help the owner prove the ownership.

In drug discovery, hypergraph MaxCut is used to select a subset of substances for regulatory measures by dividing the dataset into two isolated groups of drugs. In Table 6.5, we validate the watermarking performance on National Drug Code (NDC) [20], in which the drugs are considered as nodes and the substances are represented as hyperedges. As seen, OptiMarks maintains a 100% watermark extraction rate with less than 0.01% solution quality degradation. We skip VCW and VCW here as they are designed to watermark graphs and are unable to watermark hypergraphs.

In social recommendation, MaxCut is used to partition users in a social graph for maximized inter-group disagreement, which can be used to identify opposing user communities and tailor recommendations accordingly. In Table 6.6, we use the public collaboration dataset COLLAB [161] as a proof-of-concept, showing OptiMarks's watermarking performance. For each graph, nodes represent researchers and edges indicate if two researchers have collaborated. As seen, the two baselines ECW and VCW degrade the solution quality by 0.33% and 1.05%, and have a watermark extraction rate of 49.67% and 60.33%, respectively. Due to the dense connection of the graph, ECW treats the watermark constraints similarly to other problem constraints, and the solver failed to satisfy them, resulting in low watermark extraction rates. By encoding watermarks on the node labels, VCW achieves higher watermarking extraction rates, but at a cost of 1.05% solution quality degradation. In contrast, OptiMarks maintains a 100% watermark extraction rate and degrades the solution quality by only 0.67%.

Distributed Watermarking We demonstrate OptiMarks's scalability by watermarking on the Stanford G22 [258] and OGB-Arxiv datasets [82]. Following [78], we train the hypergraph neural network on four servers collaboratively, and each server encodes its respective signatures.

Table 6.4. OptiMarks watermarking performance on protein prediction problem.

Graph	Nodes	Edges	Solution Quality ↑				Extraction Rate ↑		
			Clean	ECW [243]	VCW [153]	OptiMarks	ECW [243]	VCW [153]	OptiMarks
#5	4152	2064	1211	1205	1199	1205	100%	54%	100%
#20	5748	28534	1660	1650	1635	1653	100%	48%	100%
#195	2495	13128	708	691	680	691	100%	38%	100%
#241	1284	6606	358	354	345	354	100%	50%	100%
#293	1160	5484	343	347	333	347	100%	50%	100%

Table 6.5. OptiMarks watermarking performance on drug discovery problem.

Graph	Nodes	Edges	Solution Quality ↑		Extraction Rate ↑
			Clean	OptiMarks	
NDC	3767	29810	27523	27498	100%

Table 6.6. OptiMarks watermarking performance on social recommendation problem.

Graph	Nodes	Edges	Solution Quality ↑				Extraction Rate ↑		
			Clean	ECW [243]	VCW [153]	OptiMarks	ECW [243]	VCW [153]	OptiMarks
#176	419	39436	22856	22858	22704	22829	48%	71%	100%
#567	331	17594	10384	10285	10192	10207	60%	50%	100%
#833	388	38934	22780	22769	22636	22742	41%	60%	100%

As shown in Table 6.7, the distributed watermarking will not degrade the watermarking performance but reduces the watermarking time by 70.87% on the smaller Stanford G22 dataset. On the large OGB-Arxiv dataset, single-GPU training is unable to watermark the solution and results in out-of-memory errors. By encoding the signature on multiple machines concurrently, OptiMarks is able to watermark the combinatorial optimization solutions more efficiently.

Table 6.7. OptiMarks watermarking performance in distributed training settings.

Graph	Nodes	Edges	w/o WM Performance	WM Performance	WM Extraction Rate	Distributed WM Time (s)	Single GPU WM Time (s)
Stanford G22	2,000	19,990	13112	13091	100%	78.18	268.39
OGB-Arxiv	169,343	1,166,243	855382	854780	100%	839.29	OOM

6.5.2 Robustness Evaluation

We consider the adversary is an end user of the solver and seeks to exploit the high-quality optimized solution while removing the embedded signature to evade detection by the solver provider. The adversary has full access to and can manipulate the released solution. However, the adversary does not have access to (i) the watermark insertion process, including the

hyperparameters, embedded signatures, and watermark decoder parameters used for signature embedding, or (ii) the watermark verification procedure.

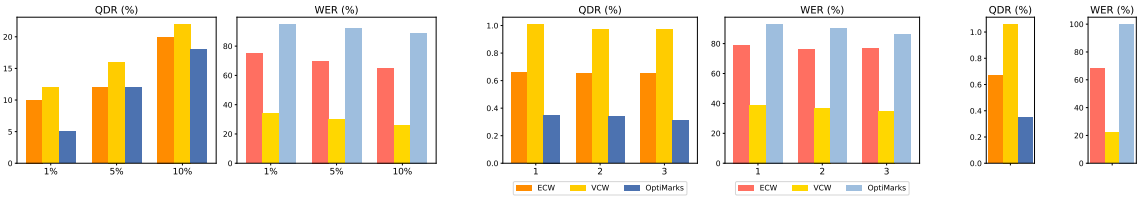
We consider the following attacks, with results detailed in Figure 6.3: (i) **Random Attacks**, where the node labels are randomly flipped to remove the encoded signatures; (ii) **Optimization Attacks**, where simulated annealing [104] is applied on top of the watermarked solutions to remove the signature, while maintaining solution quality; (iii) **Transformation Attacks**, where the adversary relabels the graph node ID to remove the signature. Here we use the MaxCut problem on the Stanford graph dataset [258] as a proof of concept showing OptiMarks's robustness.

Random Attacks: For random attacks, the adversary aims to remove encoded signature by randomly modifying the label of 1%, 5% and 10% nodes in the graph. The results are in Figure 6.3a. Although the solution quality degrades noticeably when 10% of the nodes are perturbed, OptiMarks still achieves an 80% watermark extraction rate. This demonstrates that OptiMarks can continue to provide reliable ownership verification even under substantial random corruption, benefiting from its adversarial training and error correction code mechanisms.

Optimization Attacks: For optimization attacks, the adversary aims to remove encoded signature while preserving solution quality by re-optimizing the solution using simulated annealing [104]. We show the results of running 1, 2, and 3 additional rounds of optimization in Figure 6.3b. As seen, OptiMarks maintains high watermark extractions rates because signatures are decoded from global graph patterns using a learned GNN and are reinforced through adversarial training, making them difficult to remove without significantly altering the solution.

Transformation Attacks: For transformation attacks, the adversary attempts to invalidate ownership verification by relabeling node indices in the graph, thereby disrupting signature decoding. As shown in Figure 6.3c, OptiMarks can reliably recover the embedded signature even after node indexes are permuted. This robustness arises because (i) OptiMarks relabels node indexes during signature decoding based on the intrinsic graph structure rather than raw indices, and (ii) signature decoding is performed holistically over the entire graph using a GNN,

rather than by checking individual watermark constraints. Consequently, OptiMarks achieves a 100% watermark extraction rate under transformation attacks.



(a) Random Attacks **(b)** Optimization Attacks **(c)** Trans. Attacks

Figure 6.3. OptiMarks performance under random, optimization, and transformation attacks. OptiMarks provides strong ownership proof even when the solution quality is compromised. QDR is quality degradation rate and WER is watermark extraction rate.

6.5.3 Analysis

In this section, we perform analysis regarding OptiMarks's capability and integrity. Here we use the MaxCut problem on the Stanford G22 graph [258] as a proof of concept.

Watermarking Integrity We analyze the integrity of OptiMarks, in which it shall only provide ownership proof for the watermarked solution and show low watermark extraction rates on non-watermarked ones. In Table 6.8, we provide the watermark extraction rates of both watermarked and non-watermarked MaxCut problem solutions over the Stanford G22 graph. non-WM1 is the HypOp's solution that is not being watermarked by OptiMarks. non-WM2 is the simulated annealing [104]'s solution without being watermarked by OptiMarks. As seen, OptiMarks demonstrates its integrity and only provides ownership for the solutions that have been watermarked.

Table 6.8. Watermarking integrity of OptiMarks.

Method	WM	non-WM1	non-WM2
WER (%)	100%	57%	57%

Watermarking Constraints Capacity We show the capacity of OptiMarks when encoding different number of constraints that result in different statistical significance level in Figure 6.4.

As seen, as the number of encoded constraints increases, the solution quality deteriorates significantly when p-value decreases from 0.05 to 0.005 are applied to graphs in the Stanford dataset. OptiMarks's performance degrades when the watermarking strength increases to 0.01, which provides significant watermarking strength to provide sufficient ownership proof for the solution without significantly degrading the quality.

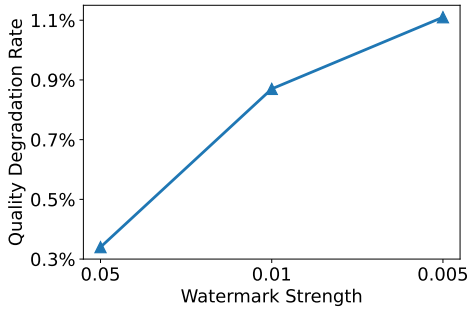


Figure 6.4. Watermarking capacity of OptiMarks.

6.6 Conclusion

This chapter presents OptiMarks, a general-purpose watermarking framework for protecting the intellectual property of combinatorial optimization solutions. We first formulate watermarks as an additional set of constraints in the optimization procedure, which is applicable to any objective function. Then, we leverage adversarial training and error-correction code techniques to robustly decode signatures from the watermarked solution even under malicious modifications. To accommodate the growing problem size, we also present the first distributed watermarking approach that encodes signatures concurrently onto multiple subgraphs to improve watermarking efficiency. Extensive evaluations of MaxCut, MaxInd, and SAT problems on both simulated and real-world datasets have demonstrated the effectiveness and robustness of OptiMarks.

6.7 Acknowledgements

Chapter 6 is a reprint of the material as it appears in: Ruisi Zhang, Nasimeh Heydaribeni, and Farinaz Koushanfar, “OptiMarks: Tracing Constrained Combinatorial Solutions with Watermarking,” in *Arxiv*, 2026. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Automated Integrated Circuit Artifact IP Protection

In the integrated circuit (IC) supply chain, multiple stakeholders collaborate to design, manufacture, and test the circuits for enhanced quality metrics and correct functionalities. During the IC design flow, the physical design stage [92, 207] transforms the design's logic netlist to a physical layout for fabrication. The placement [127, 159] and routing [124, 58] of the circuit components are optimized while ensuring various design constraints and desired functionalities. However, the IC layouts are susceptible to various security threats from the supply chain [198], such as intellectual property (IP) theft, counterfeiting, and unauthorized overproduction. Watermarking [91, 223, 268, 267] protects the physical design IP by encoding invisible, confidential, and robust signatures onto the IC layouts. These signatures help the design company to claim ownership of the physical design layouts and track the distribution of the design outcomes.

Prior physical design watermarking solutions fall into two categories [268]: (i) invasive watermarking and (ii) constraint-based watermarking. Invasive watermarking [171, 223, 16, 203] adds redundant wires or cells into the layouts as watermarks. Nevertheless, the encoded signatures can be easily forged if the adversary has prior knowledge of the watermarking scheme, making it less secure for real-world applications. Constraint-based watermarking [91, 268, 25], on the other hand, adds additional positional constraints to watermark selected cells during the

placement stage. However, selecting the watermark cells without considering the modern IC design constraints, such as macros and fence regions, would significantly deteriorate quality metrics.

This chapter first introduces ICMarks [268, 267] to encode a watermark region of cells onto modern IC design without impacting the overall quality metrics. It consists of two consecutive steps, namely, *Global Watermarking* (GW) and *Detailed Watermarking* (DW), and targets the watermark insertion at the global and detailed placement stages, respectively. *Global Watermarking* (GW) leverages a novel scoring scheme to identify a region to watermark that does not violate the design constraints and ensures minimal performance deterioration. The identified watermarked region, along with the associated cells inside it, encodes the GW signature by co-optimizing the placement such that only the associated cells intersect the watermarked region. Next, *Detailed Watermarking* (DW) selects cells that do not overlap with nearby cells when moving along the x/y axis within GW's watermarked region. The cells are watermarked by shifting in the x/y directions to encode DW signature in the detailed placement. As such, the strategically inserted watermarked region and cells ensure the signature insertion satisfies the modern IC design constraints and incurs minimal quality degradation. The design company proves ownership by reverse engineering [192, 9] the logic netlist and all standard cell locations from an IC layout in GDSII format, and employ ICMarks to decode signature for ownership verification.

Then, this chapter presents AUTOMARKS that leverages graph neural networks (GNNs) to automate the search for the best region constraints to watermark, resulting in an efficient and transferable watermarking framework for physical design. By representing the layout as a graph with node features capturing physical, semantic, and design constraint information, we leverage GNNs to learn the overall quality degradation (i.e., wirelength or timing metrics) incurred by watermarking a region centered on a specific node. The GNN training is performed on one specific layout and can later be generalized toward watermarking other layouts of the same quality metric objectives using the pre-trained weights. As such, AUTOMARKS is efficient in

finding the target position to encode the watermark and quality-preserving by learning the metrics labels when encoding the signature. AUTOMARKS is designed to protect layout-level design IP, which is complementary to existing approaches that protect IP at the RTL level [27, 87], gate level [112], and macro level [240]. Together, these techniques can be integrated to provide broader security coverage across the IC supply chain.

AUTOMARKS encompasses three key stages: (i) *watermark search*, (ii) *watermark insertion*, and (iii) *watermark extraction*. During *watermark search*, AUTOMARKS employs GNN to identify the ideal region on the layout to encode the watermark. The identification consists of a training stage to leverage a GNN to learn which node does not affect the wirelength or timing metrics after watermark insertion; and an inference stage to generalize the GNN prediction results on unseen layouts. We highlight that we designed a multi-branch GNN to optimize the timing objectives in order to preserve the quality of both total negative slack and worst negative slack on the watermarked layout. The *watermark insertion* stage encodes the watermark to the cells within the watermark region during the global placement, similar to [268]. The placement is then routed to obtain the watermarked layout. Finally, *watermark extraction* employs reverse-engineering approaches [192, 9] to acquire the cell connections and the positions from the layout. The design company can prove its ownership by decoding watermark cell positions and determining the percentage of cells within the watermark region. In brief, the contributions are summarized as follows:

- We present ICMARKS, the first watermarking scheme with both region and position constraints for the modern VLSI layouts, which does not degrade the layout quality while being robust against watermark removal and forging attacks. ICMARKS features: (i) a novel search algorithm to identify the watermarked region that adheres to design constraints and incurs minimal performance degradation in *Global Watermarking*; (ii) an innovative encoding mechanism for quality-preserving signature insertion and augmented robustness in *Detailed Watermarking*.

- We conduct experiments on benchmarks of different design objectives and sizes: (i) ICMARKS introduces no degradations on the wirelength-driven ISPD 2015 [24] and ISPD 2019 [134] benchmarks, whereas the best prior method [25] degrades the average routed wirelength by up to 1.4%; (ii) ICMARKS shows no degradations on the timing-driven ICCAD'2015 [100] benchmarks, whereas the best prior method [93, 91] degrades the total and worst negative slack by 1.14% and 1.51% respectively [1].

- We present AUTOMARKS, a transferable watermarking framework for physical design that leverages graph neural networks to automate the *watermark search* for designs that target to optimize wirelength or timing metrics. AUTOMARKS preserves the watermarking fidelity by representing the design layout as a graph and employing GNN to predict the quality degradation due to watermarking at different locations in the layout. AUTOMARKS is agnostic to the placement objectives and successfully encodes watermarks on benchmarks optimized for both wirelength and timing metrics by designing a multi-branch GNN to learn the label distributions.

- Experiments on the ISPD'15 [24], ISPD'19 [134], and ICCAD'15 [100] benchmarks demonstrate AUTOMARKS's transferability toward unseen layouts while maintaining the watermarking fidelity; It reduces the search time by 50% for large designs ($\geq 500k$ cells) compared to ICMarks[268]. Evaluations of AUTOMARKS under various watermark removal and forging attacks showcase AUTOMARKS's resiliency; the adversary cannot remove the signatures without significant quality degradations.

7.1 Background and Related Work

[1]The IC physical design is fragile that 0.5% performance degradation can compromise the additional efforts from the design company for quality metrics optimization [187].

7.1.1 Graph Neural Networks for Physical Design

A graph $G = (V, E)$ consists of a set of nodes $V = \{v_1, v_2, ..., v_{|V|}\}$ and a set of edges $E = \{e_1, e_2, ..., e_{|E|}\}$. Each node $v_i \in V$ has a k-dimensional feature vector $\mathbf{h}_i \in R^k$. The goal of a GNN [276] is to learn a function g that maps the feature embedding $\mathbf{h}_i$ of node v_i to a new embedding vector $\hat{\mathbf{h}_\mathbf{i}} \in R^h$, capturing both local and global information. In a multi-layer GNN, this mapping is performed iteratively, allowing nodes to update their feature embeddings using information from their neighbors via message passing. For each node v_i, message passing involves receiving feature embeddings from its neighbors $j \in N_i$ and aggregating the messages using customizable functions to obtain an updated representation $\hat{\mathbf{h}_\mathbf{i}}$. The computations are formulated in Equation 7.1, where f, g, and $\oplus$ are customizable functions, such as convolution. Here, $\mathbf{h}_v^{(l)}$ and $\mathbf{h}_u^{(l)}$ are the node features at layer l, $\mathscr{N}(v)$ is the set of neighbors for node v, and $\mathbf{h}_v^{(l+1)}$ is the updated feature of node v at layer $(l+1)$.

$$\mathbf{h}_v^{(l+1)} = g(\mathbf{h}_v^{(l)} \underset{u \in \mathscr{N}(v)}{\oplus} f(\mathbf{h}_u^{(l)}, \mathbf{h}_v^{(l)})) \tag{7.1}$$

As the vertices and edges in a graph are analogous to the cells and their connections in an IC netlist, graph neural networks (GNNs) serve as a promising approach during various stages of physical design, such as logic synthesis, floorplanning, partitioning, placement, and routing to improve the overall power-performance-area (PPA) metrics and speed up the chip design process [220, 144, 146, 145, 79, 139, 196]. During the placement stage of physical design, GNNs can optimize the macro locations, overall timing, and power [148, 159, 90, 147]. In addition, GNNs can ensure hardware reliability, security, and reverse engineering of gate-level netlists during the physical design stage [10, 8].

7.1.2 VLSI Placement

The integrated circuit (IC) design flow starts with design specifications that are converted to register transfer level (RTL) using a hardware description language (HDL). As depicted in

Figure 7.1, the physical design process synthesizes the design HDL to generate a logic-level netlist comprising macros and standard cells from the technology library. A floorplanning step determines regions and locations for the design's macros and the IO (input-output) ports. Typically, the standard cells are placed during the placement stage, followed by routing all the design connectivity. The routed design is verified to ensure manufacturability and the physical layout is generated in GDSII format to send to the fabrication unit for manufacturing.

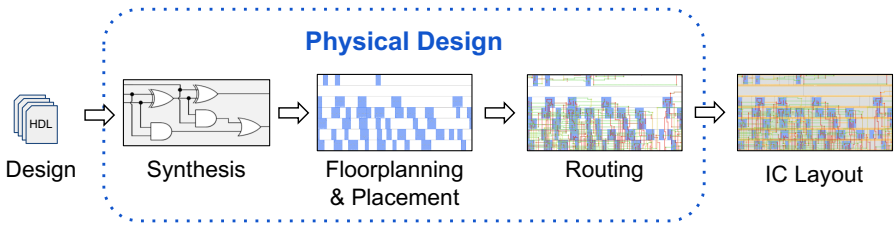


Figure 7.1. The VLSI physical design process. It generates the IC layout from a design RTL through synthesis, placement, and routing stages.

With fixed macro locations, placement is an essential step in the physical design flow to determine the locations of all the standard cells in the design [187, 182, 5, 124]. It significantly impacts the quality and efficiency of the subsequent stages, including routing and manufacturing. The VLSI placement typically consists of three stages: Global Placement, Legalization, and Detailed Placement.

Global Placement In the global placement stage, the placer aims to achieve roughly legal cell locations by distributing the cells across the chip area to minimize overlaps. In addition, the placer also targets other objectives, such as minimizing wirelength and ensuring timing constraints are met.

❶ **Wirelength-driven Placement**: A wirelength-driven placer aims to minimize the overall wirelength W while ensuring minimal cell overlap, as specified in Equation 7.2 [127].

$$min_{x,y} W(x,y) \; s.t. \; D(x,y) < \mathscr{D} \tag{7.2}$$

where (x,y) is the location of cells in the placement, and $W(x,y)$ denotes the total wirelength

connecting cells in the design. $D(x,y)$ is a density measure of the overlaps among the cells, and it should be below the threshold $\mathscr{D}$.

❷ **Timing-driven Placement**: A timing-driven placer prioritizes optimizing timing critical paths in the design while ensuring the overlaps among the cells are minimum [123]. Given a timing endpoint t (a primary output port or an input pin of memory element) with an arrival time of $t_{\text{AT}}(t)$ and a required arrival time $t_{\text{RAT}}(p)$, the *slack* of t is calculated as follows,

$$\begin{aligned} slack(t) &= t_{\text{RAT}}(t) - t_{\text{AT}}(t) \\ TNS &= \sum_{t \in P_{\text{end}}} min(0, slack(t)) \\ WNS &= \min_{t \in P_{\text{end}}} slack(t) \end{aligned} \tag{7.3}$$

With the timing endpoints P_{end} in the design, the total negative slack *TNS* and the worst negative slack *WNS* are computed as shown in Equation 7.3. A timing-driven placer optimizes either the *WNS* or the *TNS* to meet the timing constraint of the design.

Legalization The legalization stage moves cells to eliminate the cell overlaps, ensure the row alignments, and guarantee the the design constraints are met [222, 168, 255]. The multi-row height cells are prioritized for legalization before the single-row height cells.

Detailed Placement The detailed placement stage refines the legalized placement to improve design metrics such as wirelength or timing constraints [178, 37]. The refinement can be achieved by operations like (i) Swapping the positions of two cells to improve the objectives without causing legality or timing violations and (ii) Window-based approaches to find optimal cell locations within the specified region.

7.1.3 Watermarking in IC Design

Watermarking [194, 32, 232] encodes unique and confidential signatures into the IC layouts to assist product owners in proof of ownership. Design houses leverage this technique to enhance IP rights protection in the chip supply chain and detect unauthorized usages or

replications. These watermarks are typically inserted in the logic design or physical design level.

Logic Design Watermarking The watermarking at the logic design level focuses on safeguarding the Register Transfer Level (RTL) or netlist ownership. Designers embed unique signatures into the RTL or netlist that do not impact the core functionality and are invisible upon adversaries' inspection. The signature insertion can be categorized as follows: (i) modifying the finite state machine (FSM) to add additional inputs or unused components as watermarks [41, 54]; (ii) adding additional power components as side-channel signatures [18, 46]; and (iii) introducing unique triggers during netlist design that induces the malfunctions when the triggers are detected [213].

More recent works introduce logic locking to add additional logic gates and circuits that alter the original design's functionality when the incorrect key is applied [211, 95]; obfuscation to adding dummy logic gates or additional circuit connections to make the logic design appear differently from the original one [206, 15]. These techniques are orthogonal to the logic design watermarking, which can be combined to establish more robust intellectual property protection (IPP) frameworks.

Logic design watermarking faces several limitations: (i) Adding additional watermarks into the logic design needs to consider physical design criteria like timing to maintain chip quality. However, watermarking after these constraints have been optimized at the physical design stage reduces such discrepancies. (ii) In logic design, if the logic circuits are not performing correctly after watermark insertion, designers need to revisit the RTL design in the previous stage. Physical watermarking, however, does not modify the circuits' functionalities and only changes how cells are placed/connected on the canvas.

Physical Design Watermarking The watermarking at the physical design level encodes signatures onto the IC layout from two directions: (i) constraint-based watermarking [203, 25, 16, 169, 91, 93] that uses cell topology/position as the additional constraints during physical design. Row Parity [93, 91] constraints 1-bit watermark cells to be on the odd row and the 0-bit watermark cells to be on the even row. Cell Scattering [25] enforces the 1-bit watermark cell to move one unit along the y-axis and 0-bit cell one unit along the x-axis. (ii) invasive

watermarking [171, 223] that adds redundant cells or wires as the watermark signatures.

The watermarking frameworks' quality is evaluated by the following metrics.

Criteria 1 - Fidelity: The watermarked layouts shall incur minimal quality degradation compared with non-watermarked ones on metrics like wirelength and timing. Besides, the watermarked layouts should adhere to the layouts' design constraints, such as fence regions and row alignments.

Criteria 2 - Efficiency: The watermarking framework shall be efficient with minimal overheads for watermark insertion.

Criteria 3 - Robustness: The watermarked layout shall be robust against various attacks targeting to remove or forge the signature.

Table 7.1. Comparison of IC watermarking (WM) frameworks. The empty circle framework does not meet criteria; the half circle framework partially meets criteria; the full circle framework completely meets criteria.

Stage	WM Method	Fidelity	Efficiency	Robustness
Logic Design	Logic Design WM [41, 18, 213]	◐	○	◐
Physical Design	Invasive WM [171, 223]	◐	●	○
	Constraint-based WM [91, 93, 203, 16, 169]	◐	●	●
	ICMARKS	●	●	●

As listed in Table 7.1, during the logic design watermarking [41, 18, 213], the impact of layout quality and robustness depends on the design of the signature insertion algorithms. The watermark encoding also has to consider the physical design criteria in the later stages, and if certain requirements or design specifications are not met, the designers need to revisit the logic design for updates. Such alignments make it less efficient to insert watermarks at the logic design stage. The constraint-based watermarking frameworks [91, 93, 203, 16, 169] encodes signatures during the placement stage of physical design. The watermarks are embedded as part of the layout. By encoding the constraints into the layout, constraint-based watermarking are robust against removal or forging attacks. The overhead of watermark insertion remains

negligible compared to the time required to optimize the IC layout. However, the watermarking frameworks do not consider the new design constraints, such as macros and fence regions that could lead to performance degradation.

By adding redundant cells/wires, the invasive watermark [171, 223] is efficient for watermark insertion, and the quality degradation depends on how the corresponding algorithm designs the insertion mechanism. Nevertheless, the encoded watermarks can be forged if the adversaries have knowledge of the watermarking algorithm.

7.2 Motivation and Problem Formulation

In this section, we introduce ICMARKS and AUTOMARKS's goal and its threat model.

Watermarking Goal In the physical design stage, the IC design company invests huge efforts in identifying and fine-tuning optimization objectives for better cell placement and net routing. The optimization and customization are infused into different levels of the final layout, including (but not limited to) blocks and cells, to boost IC performance and reduce manufacturing expenses. As such, the final IC layouts constitute invaluable intellectual property, emphasizing the need to safeguard it.

ICMARKS and AUTOMARKS establishes ownership proof for the VLSI physical design by inserting watermarks into the placement stage, thereby authenticating the IP for the final layout. Figure 7.2 depicts a typical scenario within the supply chain. The IC design company D uses highly customized and optimized placement and routing algorithms to enhance physical design quality. The layout L_D from the physical design stage is watermarked and sent to the fabrication company for manufacturing. The manufactured ICs are verified for their functionalities in the test company. The design company D can employ reverse engineering approaches [192, 9] to extract the logic netlist and all standard cell locations from an IC layout in GDSII format, and employ reverse engineering tools to decode the signature for ownership verification.

Threat Model The adversary $\mathscr{A}$ in the fabricate or test company aims to steal the layout and

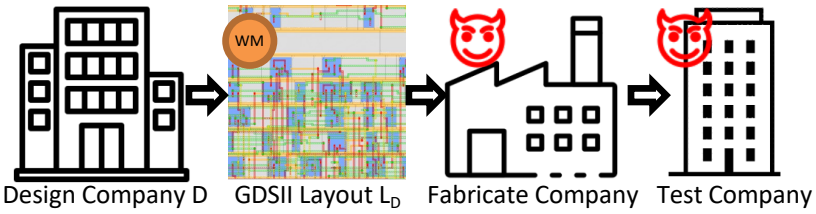


Figure 7.2. IC layout watermark scenario in the supply chain. The design company watermarks the IC layout before manufacturing and testing.

stop D from claiming its ownership by executing different attacks.

❶ **Adversary's Locations**: We consider the adversary $\mathscr{A}$ to be in the fabrication or test company and has access to the layouts produced by the design company.

❷ **Adversary's Objective**: Adversary $\mathscr{A}$'s objective is to (i) prevent design company D from ownership proof by removing or forging the encoded signature; (ii) the attacked layout performance, such as wirelength or timing metrics shall not be significantly compromised.

❸ **Adversary's Capacities**: We consider an adversary $\mathscr{A}$ with the following capacities: (i) The adversary $\mathscr{A}$ has access to the layout L_D. However, he cannot reverse-engineer the optimizations and customizations to reproduce L_D because they are infused into every level of the design. (ii) The adversary $\mathscr{A}$ has access to open-source physical design tools to remove or forge the watermark. He also knows the general algorithms to watermark the layout. However, the random seeds, signatures, and insertion parameters are beyond his reach.

7.3 ICMARKS Design

This section presents the proposed layout watermarking ICMARKS, a two-level scheme that inserts watermarks during multiple stages of VLSI placement. In Section 7.3.1, we introduce the *Global Watermarking* method. Next, we propose an independent embedding of the watermark signatures as *Detailed Watermarking* in Section 7.3.2. While *Global* and *Detailed Watermarking* can be used independently for IP protection, ICMarks combines both techniques to take advantage of their best properties.

7.3.1 *Global Watermarking* (GW)

Global Watermarking embeds watermarks during the global placement stage. The watermarks, including a pre-defined watermarked region and associated cells in the region, are embedded as a co-optimization term during global placement. The co-optimization objective is to ensure only those associated cells are placed in the watermarked region.

GW Watermark Selection Given the original, optimized placement denoting the position of all cells as P_{ori}, we employ a sliding window-based algorithm that traverses the placement to search for a region to watermark as in Fig. 7.3. An ideal watermarked region should meet three criteria: (i) as the signatures are encoded by moving cells along x/y-axis in DW, the number of cells N_c within the region should be sufficient to accommodate the required number of signature bits N_w; (ii) the total cell area S_{cell} within the watermark region are S shall be small, thereby providing ample space for watermarked cells to maneuver. The small watermark region utilization ensures the watermarks are encoded in the low-density regions. ; and (iii) the cells area $S_{overlap}$ overlap on the watermarked region boundary shall be minimized, such that their displacement from the watermarked region has minimal impact on the layout performance. These requirements are incorporated into an evaluation function f, as illustrated in Eqn. 7.4, to evaluate each region of the original placement P_{ori}.

$$f = \alpha\frac{N_w}{N_c} + \beta\frac{S_{cell}}{S} + \gamma\frac{S_{overlap}}{S} \tag{7.4}$$

The scoring function is balanced by adjusting the weights α, β, γ, where $\alpha, \beta, \gamma \in [0,1]$. The scores of the traversed regions are then normalized into a range of $[0,1]$. For regions that are nested within macros or other fence regions and any region where $N_c < N_w$, the evaluation function f yields the highest evaluation scores as 1.0, rendering the regions not suitable for inserting GW watermarks. The region with the minimum score is chosen as our target watermarked region R_w, and the set of associated cells within the region is denoted as C_{w1}.

GW Watermark Insertion The watermarked region R_w and its associated cells C_{w1} is formulated

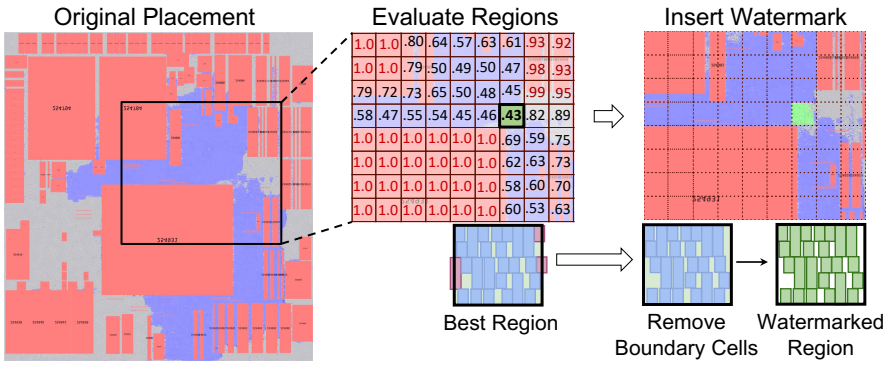


Figure 7.3. Global watermarking pipeline with stride = sliding window size. The blue cells are the standard cells, the red cells are the macros, and the green cells are wm cells in GW.

as the additional watermarked region constraint in the placement objective. It enforces the associated cells C_{w1} to be in the watermarked region R_w and other cells to be out of R_w. For a design with K fence regions, the placement formulation includes watermarked region R_w as the additional optimization constraints, as specified in Eqn. 7.5.

$$\begin{aligned} \min_v \quad & \textstyle\sum_{e\in E} W(e;v) + \lambda D(v), \\ \text{s.t.} \quad & v_k = (x_k, y_k) \in R_k, \quad k = 0, \cdots, R_K, R_w, \end{aligned} \tag{7.5}$$

W is the wirelength term and D is the cell density term with density multiplier λ. v denotes the (x,y) location of cell and $e \in E$ is the design net. The watermarked region R_w is added as an additional region constraint with associated cells C_{w1} to obtain a watermarked placement P_w.

GW Watermark Extraction The design owner asserts ownership by employing the watermarked region R_w to extract cells C'_{w1} within the region and comparing these with the associated cells C_{w1} and non-associated cells C_{w1o}. The extraction rate of the watermark, denoted as WER_{GW}, is calculated in Eqn. 7.6.

$$\%WER_{GW} = 100 \times \frac{|(C'_{w1} \cap C_{w1}) - (C'_{w1} \cap C_{w1o})|}{|C_{w1}|} \tag{7.6}$$

7.3.2 *Detailed Watermarking* (DW)

In *Detailed Watermarking*, the watermarked cells are embedded by moving them slightly along the x- or y-axis after legalization. A follow-up detailed placement stage compensates for the performance degradation from the watermark insertion.

DW Watermark Selection The *Detailed Watermarking* moves cells along the x- or y-axis to insert watermarks into the design. Randomly selecting watermarked cells and moving them without considering the dense interconnection could lead to significant performance degradation. To avoid this, *Detailed Watermarking* only selects cells that will not overlap with their neighbors after cell movements as the watermark as in Fig. 7.4.

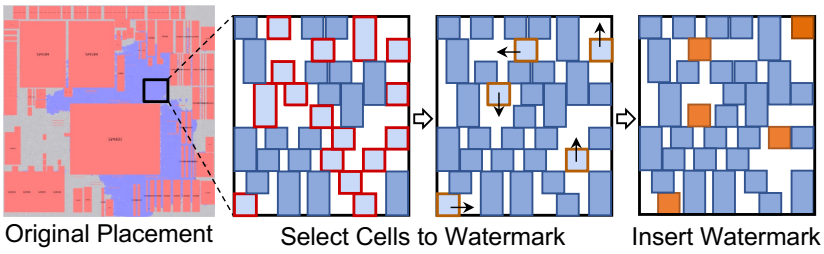


Figure 7.4. Detailed watermarking pipeline with candidate cells from which watermarked cells are selected.

Algorithm 5 outlines how *Detailed Watermarking* identifies cells that will not overlap with neighbors during watermarking movements and subsequently inserts the signature. It starts by comparing the positions of cells within each row and identifying the cell indices that can move $|d_x|$ along the x-axis. Then, the algorithm compares cells across adjacent rows to determine if they can be moved up or down for $|d_y|$. The cells C_x with room along the x-axis and C_y with room along the y-axis are identified as candidate cells for watermarking. D_x and D_y are associated with these candidate cells to retain a record of their movement directions and distances.

DW Watermark Insertion At the end of legalization with placement $P_{lg} = (X_{lg}, Y_{lg})$, cells are moved along the x- or y-axis to insert unique watermark signatures hashed as N bit sequences B_N. If the i-th bit in the signature B_N is 1, the cell c_i randomly selected from the candidate watermarked cell set C_x is moved along the x-axis for $D_x[c_i]$. If the i-th signature bit is 0, the

Algorithm 5: DW Watermark Selection and Insertion

Input: Legalized placement (X_{lg}, Y_{lg}), signature B_N, unit distances $|d_x|$ and $|d_y|$
Output: Watermarked cells C_{w2}, placement (X_{wm}, Y_{wm})

Initialize empty sets C_x, C_y, D_x, D_y
for $r \leftarrow 0$ **to** `total_num_rows` **do**
  foreach $c \in$ CELLS_IN_ROW(r) **do**
    if IS_MOVABLE_X(c) **then** `// left: -1; right: 1`
      Append c to C_x
      $s \leftarrow$ GET_DIRECTION_X(c)
      Add $(s \cdot |d_x|)$ to D_x
    if IS_MOVABLE_Y(c) **then** `// down: -1; up: 1`
      Add c to C_y
      $s \leftarrow$ GET_DIRECTION_Y(c)
      Add $(s \cdot |d_y|)$ to D_y
Randomly shuffle C_x and C_y
$X_{itr}, Y_{itr} \leftarrow X_{lg}, Y_{lg}$
for $i \leftarrow 0$ **to** $|B_N| - 1$ **do**
  if $B_N[i] = 1$ **then**
    $c_i \leftarrow$ RANDOM_CHOOSE(C_x) `// no replacement`
    $X_{itr}[c_i] \leftarrow X_{lg}[c_i] + D_x[c_i]$
  else
    $c_i \leftarrow$ RANDOM_CHOOSE(C_y) `// no replacement`
    $Y_{itr}[c_i] \leftarrow Y_{lg}[c_i] + D_y[c_i]$
  Add c_i to C_{w2}
$(X_{wm}, Y_{wm}) \leftarrow$ DETAILED_PLACEMENT(X_{itr}, Y_{itr})
return C_{w2}, X_{wm}, Y_{wm}

cell c_i is randomly chosen from the candidate watermarked cell set C_y and moved along the y-axis for $D_y[c_i]$. The resulting placement is denoted as intermediate placement $P_{itr} = (X_{itr}, Y_{itr})$, and the indices of the cells moved along the x- or y-axis form the watermarked cells C_{w2}. These movements are performed before detailed placement, allowing possible performance degradation to be compensated during the subsequent detailed placement phase and yield a watermarked solution $P_{wm} = (X_{wm}, Y_{wm})$. The selected watermarked cells C_{w2}, and their corresponding position distance $\text{Dist} = P_{itr}(C_{w2}) - P_{wm}(C_{w2})$ between intermediate placement $P_{itr}(C_{w2})$ and the watermarked placement $P_{wm}(C_{w2})$ constitute the watermark. The random wm cell selection before detailed placement ensures the signatures are encoded randomly.

DW Watermark Extraction The design owner detects watermarks in the placement P' by comparing the watermarked cell C_{w2} position with the intermediate placement P_{itr} and calculates the new distance Dist$'$ as $P_{itr}(C_{w2}) - P'(C_{w2})$. If C'_{w2} cells in the extracted Dist$'$ matches the Dist both along x- and y-axis, the watermark extraction rate is calculated as Eqn. 7.7.

$$\%WER_{DW} = 100 \times \frac{|C'_{w2}|}{|C_{w2}|} \tag{7.7}$$

7.3.3 ICMARKS Watermarking

As a combination of both the GW and the DW, ICMARKS applies global watermarking during its global placement stage and detailed watermarking on top of the watermarked region before its detailed placement stage. By combining these two watermarking schemes, the inserted watermark is further augmented.

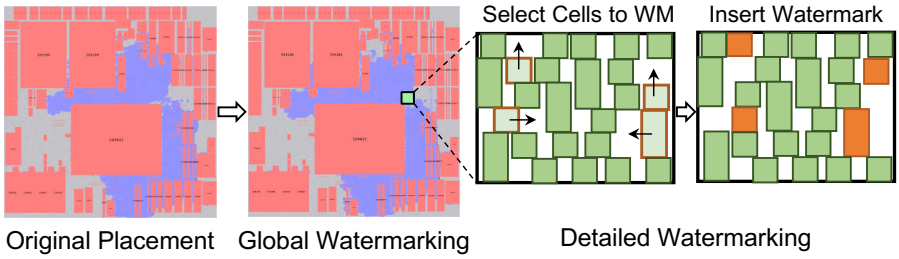


Figure 7.5. ICMARKS framework. ICMARKS first applies global watermarking during its global placement and then applies detailed watermarking on top of the watermarked region before the detailed placement. The green cells are the wm cells in GW stage and orange cells are the wm cells in DW stage.

ICMARKS Watermark Insertion As in Fig. 7.5, ICMARKS employs a hybrid approach that combines *Global* and *Detailed Watermarking* to produce stronger watermarks. Initially, ICMARKS searches for a watermarked region using the strategy akin to *Global Watermarking*, yielding a watermarked region R_w and associated cells C_{w1}. ICMARKS runs global placement and legalization with such watermark constraints. Then, it perturbs cells along the x- and y-axis within the watermarked region R_w, following the approach of *Detailed Watermarking* and obtain

an intermediate placement P_{itr}. Afterward, the detailed placement proceeds to compensate for watermarking degradation and produce the watermarked placement P_{wm}. The perturbed distance Dist and cells C_{w2} are the watermarks in the second step.

ICMARKS Watermark Extraction To claim ownership of the layout, given the cell positions P', the watermark holder can use the following two steps. (i) The extraction of the watermarks C'_{w1} from the watermarked region R_w, with the global extraction rate denoted as in Eqn. 7.6. (ii) Afterwards, the inserted signatures are extracted by comparing $P' - P_{itr}$ with Dist at the watermarked cell indices C_{w2}. The number of cells successfully matched is denoted as C'_{w2}, from which the detailed extraction rate is calculated as in Eqn. 7.7. The final WER is calculated in Eqn. 7.8.

$$\%WER_{\text{ICMARKS}} = 100 \times \frac{WER_{GW}+WER_{DW}}{2} \tag{7.8}$$

7.4 AUTOMARKS Methodology

This section introduces AUTOMARKS and the stages in its pipeline: *Watermark Search*, *Watermark Insertion*, and *Watermark Extraction*, as depicted in Figure 7.6. *Watermark Search* identifies potential regions and cells to encode the signature on layouts that optimize for wirelength in Section 7.4.1, and timing metrics in Section 7.4.2. *Watermark Insertion* encodes watermarks onto the IC layout without degrading the quality metrics as detailed in Section 7.4.3. Finally, *Watermark Extraction* decodes the watermark signature to verify ownership as described in Section 7.4.4.

7.4.1 *Wirelength-driven Watermark Search*

The objective of the *Watermark Search* is to identify a region of cells that incurs minimal quality degradation after the signature is inserted. It starts by constructing a graph representing the netlist while preserving location, semantic, and design constraint features. Then, AUTOMARKS

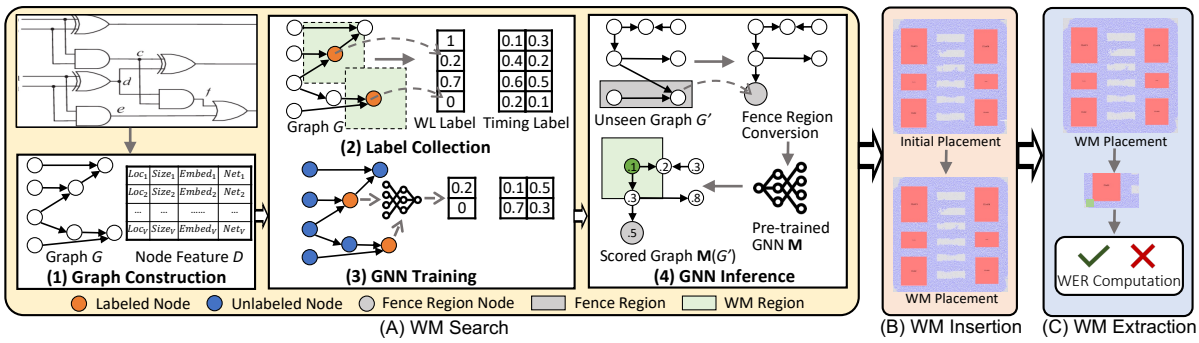


Figure 7.6. AUTOMARKS flow: The Watermark Search leverages GNN to identify the region and cells to watermark. For the wirelength-driven benchmarks, the label and GNN prediction result is the wirelength improvement rate over the non-watermarked placement. For the timing-driven benchmarks, the labels and the GNN prediction results are the total negative slack and worst negative slack improvement rate over the non-watermarked placement. Then, the Watermark Insertion encodes the selected watermark region of cells on the layout during the placement stage. The Watermark Extraction decodes the watermark and compares it with the encoded ones for ownership proof.

collects the quality degradation labels on the training design and trains a graph neural network to learn the watermarking fidelity on the training nodes. The pre-trained GNN inferences on an unseen layout and the minimum-scored region nodes are selected as the target cell to watermark.

Graph Construction The design instances, e.g., standard cells and macros, are connected to others by wired nets through the cell pins. As in Figure 7.6(A-1), the design is converted into a directed homogeneous graph $G = (V, E)$, where the source nodes are the cells sending signals and the destination nodes represent the cells loading signals. The edges E are the nets connecting the cells. For each node in the graph, we construct an 8-dimension feature D as listed in Table 7.2. The first four dimensions describe the physical information, including cell locations from initial placement and the cell size. To capture the semantic information, we extract the cell name embeddings [149] from S-BERT [195] and use Principal Component Analysis (PCA) [2] to reduce them to four dimensions to form the last four features of D.

Modern IC designs often have additional design constraints, such as macros and fence regions. We incorporate these design constraints into the graph modeling as follows:

- *Macros*: The macros are treated the same as other standard cells in the graph. They are

Table 7.2. Node feature construction for wirelength-driven watermark search.

Type	Dim	Description
Cell Location	2	from initial placement (x, y)
Cell Size	2	cell size (width, height)
Cell Name	4	S-BERT semantics [195]

distinguished from the standard cells by different cell name embeddings.

- *Fence Regions*: As depicted in Figure 7.6(A-4), the cells in the fence region are characterized as a single node in the graph, with cell location/size set to the fence region location/size. The name of the fence region cell is assigned after the macros.

Label Collection We collect the watermarking quality degradation labels on `ispd19test6` [134]. We chose this design as the training graph for two reasons: (i) the design is of medium size with $\sim$ 180k nodes. It ensures a complex graph structure for the GNN to learn, and the label collection takes one minute per node; (ii) the design has macros, enabling the GNN to learn the graph structure with design constraints. The watermark region has a fixed size of $N \times row_height$, where $N = 10$.

Watermarking on every node [2] and acquiring its corresponding watermarking quality degradation would take several months for the `ispd19test6` design. Therefore, we grid the layout with the size of the watermark region and randomly sample one node from each grid as the training node. The subsampling results in approximately $3.8k$ training nodes and takes only three days to obtain all labels. For the i-th node, we denote its improvement over the initial placement wirelength WL_{INIT} as $\hat{L}_i = \frac{WL_i}{WL_{INIT}}$. After acquiring the labels of each training node, $\hat{L}_i$ is transformed into the range of $[0, 1]$ for better GNN learning. As specified in Equation 7.9, if $\hat{L}_i < 1$, it indicates watermarking on the node will not introduce any performance degradation. Such nodes are scored as 0, indicating an ideal position for watermark encoding. If $\hat{L}_i > \beta$, it suggests significant performance degradation from watermarking. As such, L_i is set to 1,

[2] For ease of explanation, we refer to watermarking the node as encoding the watermark region centered on the node throughout the paper.

denoting an unsatisfactory position for watermarking. For the $\hat{L}_i$ between 1 and β, we normalize the scores between $[0,1]$. For GNN traning, we set $\beta = 1.01$.

$$L_i = \begin{cases} 0 & \text{if } \hat{L}_i < 1 \\ \frac{\hat{L}_i - 1}{\beta} & \text{if } 1 \leq \hat{L}_i \leq \beta \\ 1 & \text{if } \hat{L}_i > \beta \end{cases} \tag{7.9}$$

Graph Neural Network Training The GNN's objective is to predict the quality metrics degradation from watermarking on the node. The model $\mathbf{M}_{wl}$ consists of seven-layer graph convolutions with ReLU activation [118]. As in Equation 7.10, for the l-th layer, $\mathcal{N}(i)$ is the set of neighboring nodes of node i, $\mathbf{W}^{(l)}$ is the learnable weight matrix at layer l, $\mathbf{h}_j^{(l)}$ represents the feature vector of the neighboring node j at layer l. The $\mathbf{h}_i^{(l+1)}$ is node i's learned feature at layer $(l+1)$.

$$\mathbf{h}_i^{(l+1)} = \text{ReLU}\left(\sum j \in \mathcal{N}(i) \frac{1}{c_{ij}} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right) \tag{7.10}$$

$\mathbf{M}_{wl}$ is trained to minimize the MSE loss between the predicted label $\mathbf{M}_{wl}(G)$ and the ground truth label L as in Equation 7.11.

$$\mathscr{L}_{\text{wl}} = \frac{1}{|L|} \sum_{i=1}^{|L|} (\mathbf{M}_{wl}(G_i) - L_i)^2 \tag{7.11}$$

Inference on Unseen Designs The unseen design is converted to a graph G' following the ***Graph Construction***. The trained GNN $\mathbf{M}_{wl}$ scores each node in G' and obtains score L'. The cell index $c_{wc} = \arg\min(L')$ is selected as the target cell to watermark. As in Figure 7.6(A-4), the watermark region R_w is centered at c_{wc}. The cells $C_w \in R_w$ are designated as the watermark cells. When encoding watermarks on larger designs, where the chip canvas is much larger to accommodate more cells than the training design `ispd19test6`, a larger watermark region R_w is required to ensure the watermark region is hard to remove. To address this, we perform a

post-aggregation over the graph and obtain L'_{agg} using Equation 7.12. The L'_{agg} better estimates the cell neighbors over larger ranges without retraining the GNN on larger designs. The target cell index is obtained as $c_{wc} = argmin(L' + \gamma * L'_{agg})$.

$$L'_{agg,i} = \frac{1}{N} \sum_{k=1}^{N} \left(\frac{1}{|\mathcal{N}_k(i)|} \sum_{j \in \mathcal{N}_k(i)} L'_j \right) \tag{7.12}$$

7.4.2 *Timing-driven Watermark Search*

The timing-driven watermark search is similar to the wirelength-driven search in Section 7.4.1 but has different prediction objectives. The timing-driven watermark search has two prediction targets, namely, the total negative slack (TNS) and the worst negative slack (WNS). We employ a similar pipeline as the wirelength-driven watermark search but (1) incorporate the timing feature in graph construction to enhance the netlist representation and (2) revise the GNN architecture to have one main branch for graph feature extraction and two follow-up subbranches to estimate the TNS and WNS.

Graph Construction Similar to wirelength-driven watermarking, we leverage the cell location, cell size, and cell name to model the physical information of the cell in the layout. We also include the additional features as the average of net weights for each node in Table 7.3. It models the slack of each cell and helps the graph neural network predict the total negative slack and worst negative slack.

Table 7.3. Node feature construction for timing-driven watermark search.

Type	Dim	Description
Cell Location	2	from initial placement (x, y)
Cell Size	2	cell size (width, height)
Cell Name	4	S-BERT semantics [195]
Net weight	1	average of net weights connected to the cell

Label Collection We collect the timing metric labels on `superblue5` [100] design, as the design is of medium size and helps the GNN to learn the mapping between watermark encoding and

timing metrics within reasonable time. The watermark region has a fixed size of $N = 100$. The non-watermarked layout's TNS and WNS are denoted as TNS_{INIT} and WNS_{INIT}. The i-th node's timing-metrics improvement over the initial ones is $\hat{TNS}_i = \frac{TNS_i}{TNS_{INIT}}$ and $\hat{WNS}_i = \frac{WNS_i}{WNS_{INIT}}$. Then, the labels are normalized similarly to the wirelength-driven watermarking, denoted as L_{tns} and L_{wns}, and used for GNN training. We note that the timing metric can vary at different design stages (placement, clock-tree synthesis, and routing) [92]. In AUTOMARKS, these metrics are extracted at the placement stage and used primarily as relative guidance signal to identify non-timing-critical paths for signature embedding.

Graph Neural Network Training The GNN's objective is to predict the *total negative slack* $\hat{TNS}_i$ and the *worst negative slack* $\hat{WNS}_i$ when watermarking on the respective i-th node. The model $\mathbf{M}_{timing}$ consists of a main branch $\mathbf{M}_m$ and two sub-branches $\mathbf{M}_{tns}$ and $\mathbf{M}_{wns}$. The main branch is used to extract the node features from Table 7.3 of the watermarked region, whereas sub-branches are used to predict the respective L_{tns} and L_{wns}. The model is trained to minimize the MSE loss between the predicted label $\mathbf{M}_m(\mathbf{M}_{tns}(G_i))$ and $\mathbf{M}_m(\mathbf{M}_{wns}(G_i))$ as in Equation 7.13.

$$\mathscr{L}_{\text{timing}} = \frac{1}{|L|}\sum_{i=1}^{|L|}((\mathbf{M}_m(\mathbf{M}_{tns}(G_i)) - L_{tnsi})^2 + (\mathbf{M}_m(\mathbf{M}_{wns}(G_i)) - L_{wnsi})^2) \tag{7.13}$$

Inference on Unseen Designs With the trained model $\mathbf{M}_{timing}$, AUTOMARKS is used to infer potential region to watermark on the unseen layout, and scores each node in G' to obtain L'_{tns} and L'_{wns}. To select the cell yielding good quality of both TNS and WNS, we first narrow down a set of candidate cells by excluding ones' predicted L'_{tns} and L'_{wns} larger than β. The cell index $c_{wc} = \arg\min(L'_{tns} + L'_{wns})$ is selected as the target cell to watermark. The watermark region R_w is centered at c_{wc}. The cells $C_w \in R_w$ are designated as the watermark cells.

7.4.3 *Watermark Insertion*

After obtaining the watermark region R_w and the watermarked cells C_w, we encode them as a placement co-optimization objective to ensure only C_w are placed in the region R_w. For a design with K fence regions, AUTOMARKS encodes an additional fence region for R_w with C_w as the watermark following Equation 7.14 [70]. Here, W is the wirelength term, and D is the cell density term with density multiplier λ. v denotes the (x,y) location of cell and $e \in E$ is the design net.

$$\begin{aligned} \min_v \quad & \textstyle\sum_{e\in E} W(e;v) + \lambda D(v), \\ \text{s.t.} \quad & v_k = (x_k, y_k) \in R_k, \quad k = 1, \cdots, R_K, R_w, \end{aligned} \tag{7.14}$$

7.4.4 *Watermark Extraction*

To prove ownership, the design company reverse-engineers the suspect layout [192, 9] and obtains netlist connectivities and cell locations. Such methods can recover large layouts containing over $7M$ cells with over 98% accuracy and efficiency. Then, AUTOMARKS queries the locations of C_w and computes the cells C'_w within the watermark region R_w. The watermark extraction rate $(\%WER)$ is computed as in Equation 7.15. The owners can use the percentage of the matched watermark as $\%WER$ to claim their ownership of the integrated circuit.

$$\%WER = 100 \times \frac{|C'_w|}{|C_w|} \tag{7.15}$$

7.5 ICMARKS Experiments

In this section, we first introduce the experiment setups in Section 7.5.1. Then, we evaluate the performance of wirelength-driven and timing-driven design watermarking in Section 7.5.2. Next, we show the robustness of ICMARKS under attacks in Section 7.5.3.

7.5.1 Experimental Setup

Datasets: We evaluate the watermarking impact on wirelength-driven benchmark: ISPD'2015 [24] and ISPD'2019 [134]; and the timing-driven ICCAD'2015 [100] benchmark suite. All designs in the ICCAD'2015 benchmark suite are large and contain more than 700k cells. The statistics and node technology of these designs are summarized in Table 7.4, with designs having fence regions marked in blue. The macros, fence regions, and IO pin locations in the benchmarks are fixed.

Table 7.4. Benchmark statistics and node technology. The designs with fence regions are in blue. The sliding window's size and stride are multiples of the adjacent row-height.

ISPD 2015 (WL)

Design	Cells	Nets	Size	Stride	Node
perf_a	108K	115K	10	5	65nm
perf_b	113K	113K	10	5	65nm
dist_a	127K	134K	20	5	65nm
mult_b	146K	152K	10	5	65nm
mult_c	146K	152K	20	5	65nm
pci_a	30K	34K	10	5	65nm
pci_b	29K	33K	20	5	65nm
superblue11	926K	936K	100	40	28nm
superblue16	680K	697K	50	10	28nm
perf_1	113K	113K	50	10	65nm
fft_1	35K	33K	10	5	65nm
fft_2	35K	33K	10	5	65nm
fft_a	34K	32K	10	5	65nm
fft_b	34K	32K	20	5	65nm
mult_1	160K	159K	50	10	65nm
mult_2	160K	159K	50	10	65nm
mult_a	154K	154K	50	10	65nm
superblue12	1293K	1293K	50	10	28nm
superblue14	634K	620K	50	10	28nm
superblue19	522K	512K	100	10	28nm

ISPD 2019 (WL) and ICCAD 2015 (Timing)

Design	Cells	Nets	Size	Stride	Node
ISPD 2019 (WL)					
ispd19test1	9K	3K	10	5	32nm
ispd19test2	73K	72K	10	5	32nm
ispd19test3	8K	9K	10	1	32nm
ispd19test4	151K	146K	10	5	65nm
ispd19test5	29K	29K	10	10	65nm
ispd19test6	181K	180K	50	10	32nm
ispd19test7	362K	359K	50	25	32nm
ispd19test8	543K	538K	50	10	32nm
ispd19test9	903K	895K	200	20	32nm
ispd19test10	903K	895K	10	5	32nm
ICCAD 2015 (Timing)					
superblue1	1209K	1215K	100	50	45nm
superblue3	1213K	1224K	100	30	45nm
superblue4	795K	802K	100	50	45nm
superblue5	1086K	1100K	100	50	45nm
superblue7	1931K	1933K	100	30	45nm
superblue10	1876K	1898K	100	30	45nm
superblue16	981K	999K	100	30	45nm
superblue18	768K	771K	100	50	45nm

Experiment Setup: ICMARKS is implemented with Python 3.9 and benchmarked on a Linux Ubuntu system equipped with NVIDIA TITAN XP GPUs, each with 12 GB RAM, and 48 Intel(R) Xeon(R) CPUs, accompanied by 128 GB RAM. The watermarking methodology of ICMARKS can be integrated into any VLSI physical design framework, and we chose to build upon the open-source placement framework, DREAMPlace [70] to demonstrate its viability. The DREAMPlace [70] accelerates the state-of-the-art placement algorithm ePlace [141]/RePlAce [36] on a GPU and maintains the same level of performance. For routing, we employ the open-source state-of-the-art framework CUGR [133]. For timing-driven benchmark suite, we use DREAMPlace 4.0 [123] where the timing metrics are measured by OpenTimer [83].

Hyperparameters: For the hyperparameters, we perform a grid search for the GW phase. In the search of Eqn. 7.4, the $\alpha = \{0.1, 0.5, 0\}$, β=$\{0.1, 0.5, 0\}$, and $\gamma = 0.1$. The search terminates when the watermarked layout quality is not compromised. For all the considered benchmark suites, we report the GW hyperparameters in Table 7.4. In the DW phase, $d_x = 1$, and d_y is set to one adjacent row-height.

Baseline: We choose the following state-of-the-art watermarking frameworks as our baselines: (i) topology constraint-based **Row Parity** [93, 91] that inserts unique bit sequences as watermarks by shifting cells to different rows in the placement stage. Cells with a 1-bit are moved to an odd row, while cells with a 0-bit are moved to an even row; (ii) position constraint-based **Cell Scattering** [25] that employs pseudorandom coordinate transformation (PRCT) algorithms to scatter the watermarked cells on the chip canvas as watermarks. Cells with 1-bit are moved along the y-axis, and cells with 0-bit are moved along the x-axis if they do not overlap with their neighbors; (iii) invasive **Buffer Insertion** [223] that adds additional buffers as watermarks without affecting timing critical paths during the placement stage. Two buffers are inserted to represent 0-bit, and one buffer is inserted to represent 1-bit. For fair comparisons, we re-implemented the baselines [93, 91, 25, 223] and integrated them into the DREAMPlace codebase for benchmarking.

We skip the baselines that: (i) have different watermarking targets. [169, 16] are designed for smaller full-custom IC designs, [204, 121] are designed for FPGAs, whereas our watermarking targets are modern digital VLSI designs; (ii) have similar watermarking approaches as our baselines, and we use the baselines as a proof-of-concept. [203] inserts flip-flops instead of buffers into the layouts as watermarks. The signature length is set to 50-bit for all frameworks.

Evaluation Metrics: For the wirelength-driven benchmark suites, whose quality is measured by the layout wirelength, we use three metrics to evaluate the watermark performance:(i) **Placement WireLength Rate (PWLR)**: The rate of estimated half-perimeter wirelength (HPWL) of watermarked layout compared to the original one; (ii) **Routing WireLength Rate (RWLR)**: The rate of routed wirelength of watermarked layout compared to the original one; (iii) **Watermark**

Extraction Rate (WER): The percentage of signatures extracted from the watermarked layout. For the timing-driven benchmark suite, whose quality is measured by the timing slack, we use three metrics to evaluate the performance under static timing analysis [83]: (i) **Total Negative Slack Rate (TNSR)**: The rate of total negative slack (TNS) of watermarked layout compared to original one; (ii) **Worst Negative Slack Rate (WNSR)**:The rate of worst negative slack (WNS) of watermarked layout compared to original one; (iii) **Watermark Extraction Rate (WER)**: The percentage of signatures extracted from the layout. The rate average is calculated by the geometric mean of designs' metrics.

To preserve the optimized outcomes from the physical design algorithms, we set a threshold of layout quality degradation to not exceed 0.5%. Because surpassing this threshold would counteract the benefits derived from the performance enhancement efforts in physical design [187].

7.5.2 Experimental Results

Wirelength-driven Watermarking Fidelity The performance of different watermark schemes is tabulated in Table 7.5 for the ISPD'2015 benchmarks [24], and in Table 7.6 for the ISPD'2019 benchmarks [134]. For all the layouts in Table 7.5-Table 7.6, their inserted watermarks can be successfully extracted. Therefore, we evaluate how much layout performance is compromised to accommodate such watermark insertion among these watermarking frameworks.

❶ **Comparison with Prior Constraint-based Watermarking** [93, 91, 25]: Among the two benchmark suites, ICMARKS results in no PWLR and RWLR degradation for accommodating the 50-bit signature. In contrast, the topology constraint-based Row Parity [93, 91] changes the row index of the cells to encode watermarks but does not consider potential design constraints (e.g., fence regions and macros) in VLSI design. As a result, it results in 2.31% and 1.94% routed wirelength (RWLR) degradation on ISPD'2015 [24] and ISPD'2019 [134], respectively. The position constraint-based Cell Scattering [25] perturbs cell locations in the optimized layout

Table 7.5. Performance on the ISPD'2015 benchmarks. All the design watermarks are successfully extracted, i.e., WER = 100%. The PWLR and RWLR are the placement and routed wirelength rates over the original designs. FR is fence region. The results in gray fail buffer insertion WM with significant degradation on the high-utilized designs (denoted with star).

Design	Utl.	Row Parity [93, 91]		Cell Scattering [25]		Buffer Insertion [223]		ICMARKS: GW		ICMARKS: DW		ICMARKS: GW+DW	
		PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓
perf_a ⋆	71.69%	1.0045	1.0155	0.9978	0.9967	1.5289	2.3270	0.9976	0.9873	0.9994	1.0003	**0.9972**	**0.9873**
perf_b	49.71%	1.0058	1.0267	1.0020	1.0001	1.0176	1.0745	0.9930	0.9901	0.9990	1.0002	**0.9890**	**0.9901**
dist_a	61.62%	1.0015	0.9999	0.9984	0.9965	1.0995	1.1072	1.0004	1.0052	0.9998	1.0005	**1.0004**	**1.0052**
mult_b ⋆	77.31%	1.0047	1.0199	0.9991	0.9923	1.8966	2.9374	1.0010	1.0028	1.0002	1.0044	**0.9994**	**1.0028**
mult_c ⋆	77.31%	1.0031	1.0252	0.9980	1.0023	1.6003	2.7459	1.0017	0.9979	0.9990	1.0020	**0.9963**	**0.9979**
pci_a	50.84%	1.0080	1.0340	0.9931	0.9846	1.6269	1.6849	1.0361	1.0048	1.0011	1.0021	**0.9997**	**0.9950**
pci_b	54.77%	1.0069	1.1173	0.9912	0.9977	1.2239	1.8207	0.9951	1.0026	0.9997	1.0008	**0.9951**	**1.0026**
superblue11 ⋆	73.64%	1.0052	1.0344	1.0278	1.0358	1.6930	3.7086	0.9992	0.9992	0.9995	1.0005	**0.9986**	**0.9992**
superblue16 ⋆	74.49%	1.0030	1.0210	0.9988	0.9989	1.1023	1.1377	1.0018	1.0204	**1.0000**	**1.0000**	1.0017	1.0204
perf_1 ⋆	90.57%	1.0035	1.0199	0.9985	0.9983	1.0150	1.0642	1.0031	0.9973	1.0042	1.0019	**0.9964**	**0.9973**
fft_1 ⋆	83.54%	1.0017	1.0260	1.0167	1.0156	1.0680	1.1457	0.9671	0.9673	1.0026	0.9975	**0.9671**	**0.9673**
fft_2	49.97%	1.0050	1.0260	1.0148	1.0114	1.4603	1.4476	0.9767	0.9770	1.0027	0.9996	**0.9767**	**0.9770**
fft_a ⋆	74.02%	1.0121	1.0200	1.0024	0.9933	1.8203	2.1923	0.9939	0.9895	1.0075	1.0094	**0.9939**	**0.9895**
fft_b ⋆	78.01%	1.0018	1.0075	0.9961	1.0030	1.9859	2.5615	0.9909	0.9890	0.9991	1.0036	**0.9909**	**0.9890**
mult_1 ⋆	80.24%	1.0050	1.0238	1.0077	1.0065	1.0464	1.0631	0.9753	0.9744	1.0004	1.0008	**0.9753**	**0.9744**
mult_2 ⋆	79.03%	1.0033	1.0199	0.9984	0.9967	1.0736	1.1147	0.9867	0.9900	0.9990	0.9992	**0.9852**	**0.9900**
mult_a ⋆	86.10%	1.0037	1.0105	0.9995	0.9968	1.3862	1.6738	0.9973	0.9916	1.0005	0.9958	**0.9973**	**0.9916**
superblue12	57.62%	1.0031	1.0067	0.9979	0.9956	1.0683	1.1044	1.0036	0.9732	0.9994	0.9942	**0.9854**	**0.9732**
superblue14 ⋆	77.63%	1.0020	1.0057	0.9991	0.9981	1.0212	1.0286	**0.9851**	0.9867	1.0001	1.0000	0.9887	**0.9867**
superblue19 ⋆	81.51%	1.0025	1.0077	1.0001	1.0005	1.0295	1.0672	0.9881	0.9809	1.0003	1.0001	**0.9814**	**0.9809**
Average: FR	-	1.0047	1.0322	1.0006	1.0005	1.0724	1.1061	1.0028	1.0011	0.9997	1.0012	**0.9975**	**1.0000**
Average: All	-	1.0043	1.0231	1.0018	1.0012	1.0536	1.0901	0.9946	0.9913	1.0007	1.0006	**0.9908**	**0.9908**

Table 7.6. Performance on ISPD'2019 benchmarks. All the design watermarks are successfully extracted, i.e., WER = 100%. The PWLR and RWLR are the placement and routed wirelength rates over the original designs.

Design	Utl.	Row Parity [93, 91]		Cell Scattering [25]		Buffer Insertion [223]		ICMARKS: GW		ICMARKS: DW		ICMARKS: GW+DW	
		PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓
ispd19test1 ⋆	83.14%	1.0060	1.0129	0.9978	0.9998	1.0394	1.0619	0.9995	1.0015	1.0023	1.0043	**0.9955**	**1.0015**
ispd19test2 ⋆	84.11%	1.0184	1.0201	1.0096	1.0016	1.0127	1.0408	**0.9981**	0.9999	1.0021	1.0003	0.9988	**0.9999**
ispd19test3 ⋆	88.77%	1.0130	1.0475	1.0180	1.0193	1.0582	1.0801	1.0059	1.0045	**0.9961**	**0.9976**	1.0059	1.0045
ispd19test4	65.90%	1.0017	1.0050	0.9999	0.9998	1.1452	1.2494	0.9957	0.9907	0.9998	1.0028	**0.9957**	**0.9907**
ispd19test5	43.06%	1.0102	1.0556	1.0998	1.0975	0.9859	1.0121	1.0013	0.9998	**0.9996**	**0.9968**	1.0013	0.9998
ispd19test6 ⋆	74.60%	1.0032	1.0106	0.9994	0.9993	1.0044	1.1108	1.0023	1.0026	**1.0001**	**0.9997**	1.0023	1.0026
ispd19test7 ⋆	95.56%	1.0028	1.0160	1.0136	1.0109	1.0003	1.0717	1.0050	1.0054	**1.0000**	**0.9997**	1.0050	1.0054
ispd19test8 ⋆	79.90%	1.0001	1.0082	0.9966	0.9958	1.0118	1.0806	0.9961	0.9929	0.9999	0.9996	**0.9961**	**0.9929**
ispd19test9 ⋆	84.02%	1.0072	1.0108	1.0108	1.0095	1.0164	1.0814	1.0023	1.0023	**1.0009**	**0.9999**	1.0023	1.0023
ispd19test10 ⋆	88.48%	1.0025	1.0090	1.0043	1.0108	1.0378	1.0982	0.9972	0.9967	0.9999	1.0002	**0.9972**	**0.9966**
Average	-	1.0065	1.0194	1.0060	1.0140	1.0304	1.0871	1.0003	0.9996	1.0001	1.0001	**1.0000**	**0.9996**

for signature insertion. However, randomly selecting and moving watermarked cells after the optimization is done results in quality degradation, as reflected by the 0.12% and 1.4% RWLR degradation over the original design on ISPD'2015 [24] and ISPD'2019 [134].

❷ **Comparison with Invasive Watermarking** [223]: The invasive watermarking Buffer Insertion [223] overlooks the additional design constraints, like fence regions and macros, in the modern VLSI design. The additional 50 buffers might be added close to such design constraints and result in significant cell displacement, as the design has to accommodate the buffers while sat-

isfying the design constraints. As in Table 7.5-Table 7.6, Buffer Insertion [223] introduced 9.01% and 8.71% routed wirelength (RWLR) degradation on ISPD'2015 [24] and ISPD'2019 [134] benchmarks. ICMARKS, however, has no wirelength degradation on both benchmarks. Besides, for the highly-utilized designs, where most of the layout space is filled with standard cell and fixed marcos/fence regions/IO pins, to accommodate the additional buffers, cells have to be displaced significantly from their non-watermarked position for buffer insertion. As such, it introduced significant PWLR and RWLR degradations as the grey numbers in Table 7.5.

❸ **DW Insertion Before/After Detailed Placement:** After the placement stage is finished, the baseline Cell Scattering [25] moves cells in the layout along the x/y axis if there is space. The key difference is Cell Scattering [25] performs the movement after detailed placement, whereas the ICMARKS: DW is performed before detailed placement. As seen in Table 7.5- 7.6, ICMARKS: DW improves the RWLR quality from Cell Scattering [25] by 0.06% and 1.39%, respectively.

❹ **Comparison with the GW and DW submodules**: As a combination of GW and DW, ICMARKS takes advantage of their strength and further reduces the performance degradation from accommodating watermarks. In contrast to GW, ICMARKS slightly improves the placement quality. This improvement stems from modifying cell positions within a designated region, either across different rows or along the x-axis. Such modifications inherently disturb solutions formulated during global placement. Given that these disturbances occur in less compact regions, they provide different inputs to the detailed placement process, thereby offering potential optimization trajectories to minimize overall wirelength. The optimization to GW depends on the quality of the perturbations introduced, where the refinement on ISPD'2019 [134] is marginal, and the improvement on ISPD'2015 [24] is 0.05% over GW.

Compared with DW, ICMARKS watermarks upon the placement solution from GW, whereas DW watermarks the original solution. The additional co-optimization of the GW watermarked region makes the placer further refine the cell positions in the watermarked region and, thus, improves the watermarked layout quality. By encoding DW signature on the GW layout, ICMARKS introduced no quality degradations. In contrast, DW watermarks upon the

original placement solution, which could introduce more performance degradation when inserting DW signatures. It is reflected by the 0.06% and 0.01% RWLR improvement over the original design on ISPD'2015 [24] and ISPD'2019 [134] benchmarks, respectively.

❺ **Layout Utilization and Design Constraints Impact on WM Performance:** As in Table 7.5-7.7, invasive watermarking yields worse performance on high layout utilization designs ($\geq$ 70%), where the layout utilization is larger than 70%. For invasive watermarking, the performance degradation becomes worse on designs with additional design constraints, e.g., fence regions and macros. Because the buffers can be encoded close to such constraints and result in significant cell displacements. In contrast, constraint-based watermarking approaches co-optimize the watermark and placement objectives to preserve the layout quality.

Timing-driven Watermarking Fidelity As depicted in Table 7.7, ICMARKS continuously preserves the layout quality in timing-driven watermarking benchmark ICCAD'2015 [100]. By strategically searching for the watermarked region and cells with minimal impact on the performance, ICMARKS results in no WNS and TNS degradation. On the opposite, the constraint-based Row Parity [93, 91] and Cell Scattering [25] overlook design constraints in modern VLSI design and did not design the watermarking algorithms with the timing optimization objectives. It results in 1.51% and 4.24% WNSR degradation over the original designs, respectively. Furthermore, while Buffer Insertion [223] encodes additional buffers on the non-timing critical path, the inserted buffers still result in 5.66% TNSR and 10.63% WNSR timing metrics degradation.

Table 7.7. Performance on ICCAD'2015 benchmarks. All the design watermarks are successfully extracted, i.e., WER = 100%. The TNSR and WNSR are the total and worst negative slack rates over the original designs.

Design	Utl.	Row Parity [93, 91]		Cell Scattering [25]		Buffer Insertion [223]		ICMARKS: GW		ICMARKS: DW		ICMARKS: GW+DW	
		TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓
superblue1	78.07%	1.0053	1.0094	1.0119	1.0026	1.0083	1.0020	0.9001	**0.8514**	1.0256	1.0249	**0.8283**	1.0021
superblue3	76.78%	1.0053	1.0176	1.0066	0.9891	1.1035	1.0988	**0.9302**	**0.9029**	1.0035	0.9884	0.9363	0.9047
superblue4	80.12%	1.1299	0.9914	0.9698	1.0191	1.1954	1.1995	**0.9037**	**0.9370**	0.9554	0.9930	0.9221	1.0094
superblue5	73.30%	0.9801	**0.9887**	1.0126	0.9977	1.0932	1.1174	0.9159	1.0001	0.9922	0.9959	**0.9159**	1.0508
superblue7	76.48%	0.9801	0.9932	0.9979	1.0212	0.9933	1.0199	**0.9584**	0.9758	0.9934	1.0199	0.9636	**0.9727**
superblue10	75.80%	1.0099	**1.0041**	1.0218	1.0141	1.0194	1.2918	1.0069	1.0072	**1.0058**	1.0068	1.0069	1.0072
superblue16	79.23%	0.9814	1.1271	0.9067	1.3274	1.0109	1.0877	1.0166	1.0473	**0.9060**	1.3485	1.0069	**1.0072**
superblue18	67.44%	1.0068	0.9965	0.9981	1.0064	1.0438	1.0613	0.9337	**0.9403**	1.0014	1.0002	**0.9254**	0.9449
Average	-	1.0114	1.0151	0.9900	1.0424	1.0566	1.1063	0.9448	**0.9559**	0.9848	1.0418	**0.9366**	0.9867

Watermarking Capacity The watermarking capacity is measured by the maximum length of watermark bits that can be inserted into the layout without significantly degrading the layout quality. For Row Parity [93, 91], Cell Scattering [25], and ICMARKS: DW, the signature length corresponds to the moved/inserted cell number. For Buffer Insertion [223], the length is the number of buffer-inserted nets. For ICMARKS: GW, the signature length corresponds to the N_w in Eqn. 7.4. For ICMARKS, the signature length corresponds to the N_w cells in global watermarking and moves N_w cell over the watermarked region in the detailed watermarking.

We use ISPD'2019 [134] as the benchmarking target and display the results in Fig. 7.7. As seen, the maximum bits that can be inserted by Row Parity and Cell Scattering are both less than 30 bits. Both methods move the cells across rows after the detailed placement as watermarks, which leads to worse RWLR than PWLR, as the router has to cross rows to connect the cells. Buffer Insertion [223] inserts signatures without considering the design constraints, leading to significant cell displacement after signature insertion. Therefore, it exhibits low watermarking capacity on ISPD'2019 [134]. The ICMARKS: DW's capacity is larger and reaches $\sim$ 100 bits. Because it moves the cells before detailed placement and the subsequent detailed placement compensates for the watermark insertion. ICMARKS: GW and ICMARKS further outperform the scheme. They both demonstrate a capacity of more than 200 bits by exploring the less compact region for watermark encoding. Since ICMARKS builds the watermarking scheme on top of ICMARKS: GW, the two PWLR and RWLR lines are close in Fig. 7.7.

Besides, we analyze the maximum number of watermark regions that can be encoded onto the layout without compromising the quality. We choose the best-scored k regions in the GW stage and encode the k regions during GW. As in Fig. 7.8, encoding 3 watermark regions results in over 20% wirelength degradation. When encoding two or more region constraints into global placement, more cells are on the selected watermark region boundary. Expelling the boundary cells to ensure the design adheres to watermarking constraints results in significant cell displacement and degrades the watermarked layout quality.

Watermarking Efficiency The watermarking efficiency is benchmarked by the fraction of the

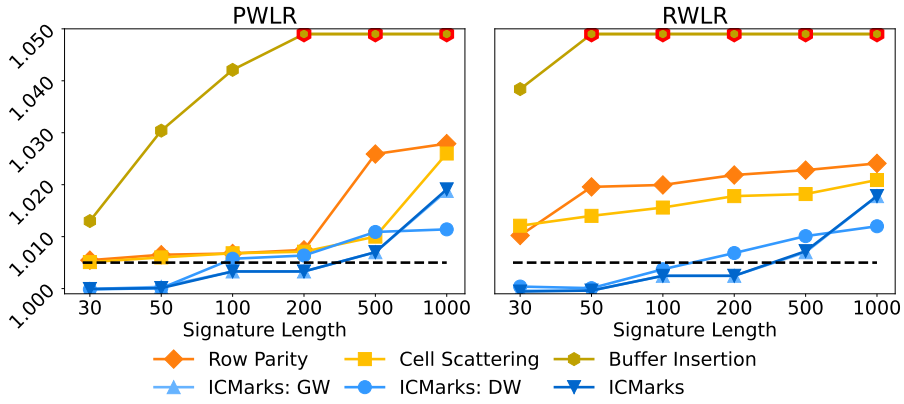


Figure 7.7. Watermarking capacity for different frameworks on ISPD'2019 benchmarks. We consider the threshold for acceptable degradation layout quality as 0.5%. The red outline indicates PWLR and RWLR are higher than the 5% limit.

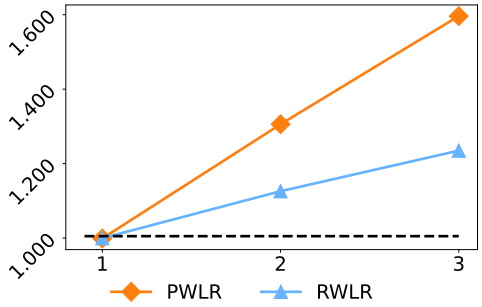


Figure 7.8. Watermarking capacity for watermark regions on ISPD'2019 benchmarks.

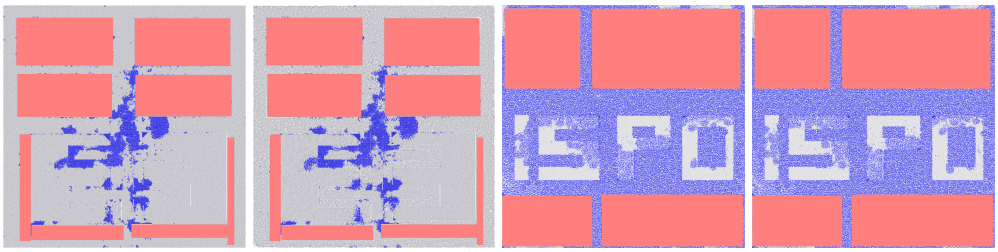
additional time takes for ICMARKS's watermark insertion compared to the overall physical design stage, primarily placement and routing. We include the average time takes to encode 50-bit signatures onto the ISPD'2015 benchmark [24] and ISPD'2019 benchmark [134] in Table 7.8. The non-WM time and WM time is the average (geometric-mean) time it takes for placement and routing on the non-watermarked and watermarked layout. The Slow Down is the percentage overhead ICMARKS introduced to the overall placement and routing phase. As seen, the time taken for watermark insertion is $\sim$10% compared with the overall placement and routing stage, making ICMARKS efficient for watermarking. Besides, no additional computation resources or external tools are required for the signature insertion.

Watermarking Stealthiness We display the layout watermarked by ICMARKS with various sizes and different design constraints in Fig. 7.9. As seen, the watermarks are embedded as part

Table 7.8. ICMARKS's efficiency on different benchmarks.

Design	non-WM Time (s)	WM Time (s)	Slow Down (%)
ISPD'2015 [24]	141.37	157.96	11.73%
ISPD'2019 [134]	946.23	996.18	9.21%

of the layout, and invisible upon inspection while maintaining 100% WERs.



(a) test5(WM) **(b)** test5(no-WM) **(c)** perf_a(WM) **(d)** perf_a(no-WM)

Figure 7.9. Watermarked Design Examples. The blue cells are the standard cells, and the red cells are the macros.

Watermark Strength Watermark strength measures the probability a non-watermarked layout carries the signature by coincidence. For each design with a total of $|C|$ cells, we watermark $|C_w|$ cells and produce the watermarked layout. In the $|C_w|$ cells, x of the cells do not meet the watermarking constraints. $|C_n|$ from the non-watermarked layout matches the watermark requirement by co-incidence. Following Row Parity [91], the watermark strength is calculated by Eqn. 7.16 as follows, where $p = \frac{|C_n|}{|C_w|}$ is the constraints met by coincidence.

$$P_c = \sum_{i=0}^{x} \cdot \mathbf{C}(|C_w|, i) \cdot (p)^i \cdot (1-p)^{x-i} \tag{7.16}$$

For each design, we calculate the P_c by running the placement with and without ICMARKS to obtain $|C_w|$ and $|C_n|$. In most of the designs, $|C_n| = 0$, meaning the non-watermarked layouts do not match the watermarking constraints. Therefore, we report the maximum P_c on ISPD'2019 benchmark [134] in Table 7.9 for ICMARKS and its submodules. The watermarks are successfully encoded during the global/detailed placement stage and have different positions from their non-watermarked ones. As such, the signatures' randomness will not be undermined

by the global/detailed placement optimizations.

Table 7.9. Watermarking strength for different signature lengths by the proposed ICMARKS watermarking framework.

	Row Parity	ICMARKS: GW	ICMARKS: DW	ICMARKS: GW + DW
P_c	8.8×10^{-16}	9.09×10^{-53}	8.08×10^{-62}	7.35×10^{-114}

Ownership Proof in Real-World Settings To prove ownership, the design company obtains the suspicious layout and employs reverse-engineering approaches [192, 9] to acquire the logic netlist and all standard cell locations. Such methodology [192, 9] recovers large layouts netlists (over 7000k cells) from the GDSII layout with over 98% accuracy and efficiency. Then, the design company uses the netlists to recover the standard cells' and macros' locations. The watermark extraction algorithms are subsequently used for ownership proof. While the reverse-engineering misalignment might degrade the watermark extraction rates slightly, ICMARKS still provides sufficient ownership proof for the design companies benefiting from two aspects: (i) high watermarking strength of 7.35×10^{-114} for 50-bit signature; (ii) high watermarking capacity that can accommodate more than 200-bit signatures without significant quality degradation, as in Fig. 7.7. As a result, even if only 88% signatures are extracted (10% lost from attacks in Section 7.5.3 and 2% lost from reverse engineering [192, 9]), ICMARKS still provides strong watermarks for 50-bit signatures. In Null Hypothesis [11], a watermarking strength (p-value) of smaller than 0.05 indicates the statistically significant presence of a watermark. Given the much lower watermarking strength than 0.05 after reverse engineering, the design company can thereby confidently claim ownership of the design layout.

Impact on Routability We analyze the impact of ICMARKS on the design routability using the ISPD'2019 [134] benchmarks employing OpenROAD [94]'s global and detailed router. The final wirelength after detailed routing (Rout. WL), average routing congestion (Rout. Congestion), an initial count of DRC violations (#DRV), the number of detailed routing optimization iterations to fix DRV (#Opt.), and the total valid via access points required for all standard cells and macros in the design (#ValidViaAp) are plotted in Fig. 7.10 for the watermarked layouts over the

non-watermarked ones.

Across all the considered benchmarks, the watermarked ones have 6% lower Rout. Congestion and Rout. WL compared to the non-watermarked designs. For large layouts with $> 300k$ cells, ICMARKS introduces 2% more routing congestion than non-watermarked ones, showing it has minimal impact on layout routability. Nevertheless, ICMARKS does incur an increase in #DRV and #Opt., by 19% and 20%, respectively, across all the designs for watermarked ones, and 10% more #DRV requiring $\sim 5\%$ more #Opt. for the large designs ispdtest7 - ispdtest10 consisting of $> 300k$ cells. #ValidViaAp, representing the layout's pin accessibility, remains similar for watermarked and non-watermarked designs. Thus, ICMARKS does not degrade the pin access of the design.

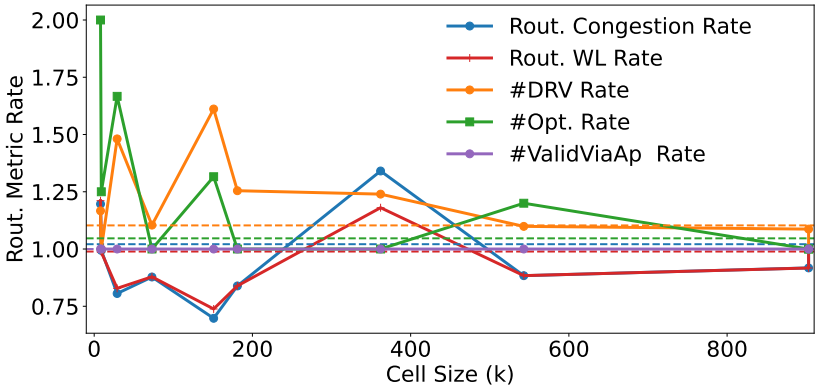


Figure 7.10. Impact of watermarking on routability. The dotted lines represent the geo-mean of large designs ispd19test7 - ispd19test10 with $> 300k$ cells.

7.5.3 Attack Evaluation

In this section, we evaluate ICMARKS's robustness under removal and forging attacks.

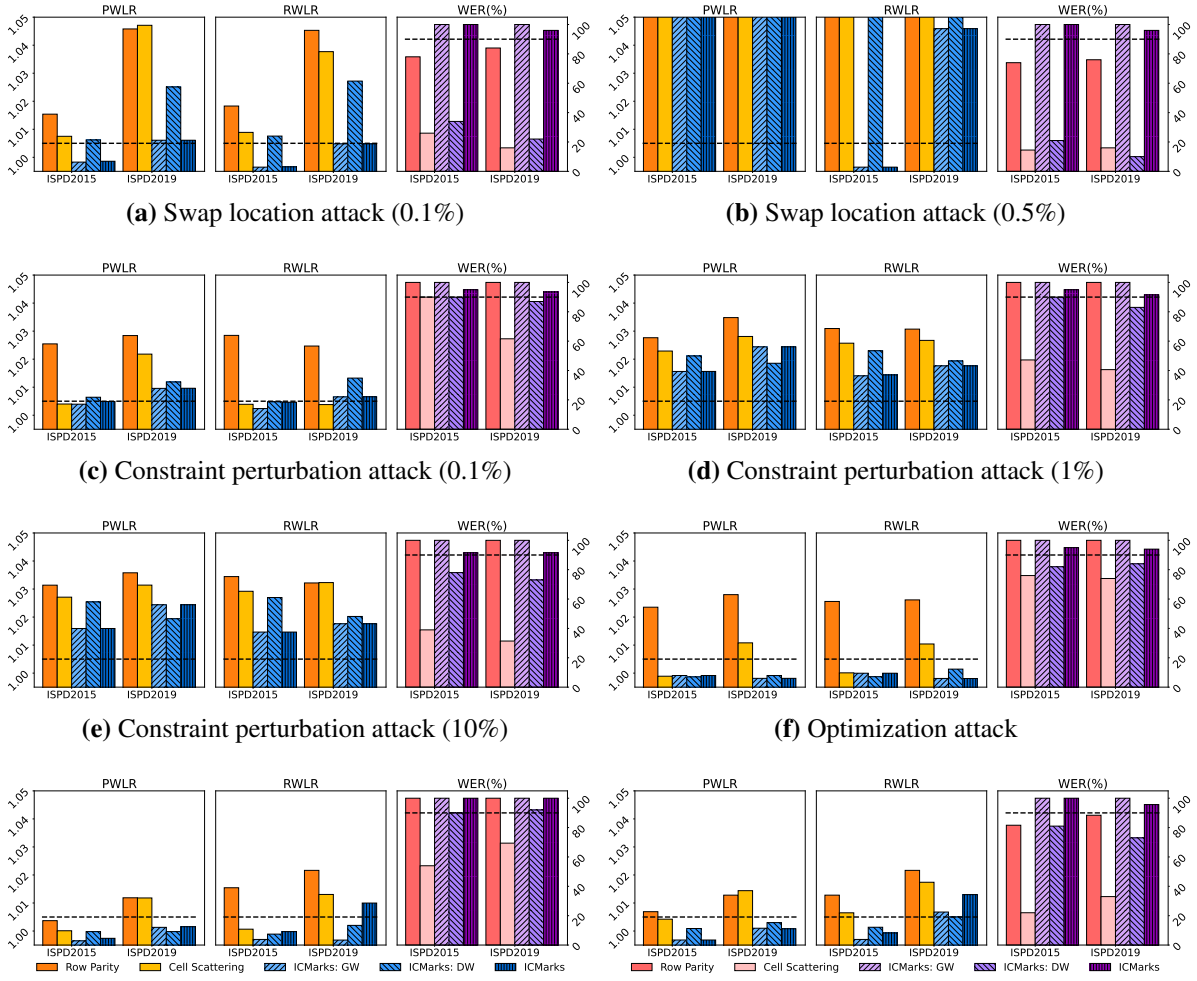
Watermark Removal Attack A successful watermark removal attack shall meet two criteria: (i) the encoded signatures are removed as reflected by the watermark extraction rate (WER) below 90% [93, 91]; (ii) the layout quality is not compromised, where the Placement Half-Perimeter WireLength Rate (PWLR), Routed Wirelength Rate (RWLR), Total Negative Slack Rate (TNSR), and Worst Negative Slack Rate (WNSR) do not exceed 1.005 [187]. The thresholds are reflected

by the black dotted lines in Fig. 7.11 and Fig. 7.12.

For an IC layout watermarked by ICMARKS, the watermarks are inserted by constraining the cell positions and cells' region. To remove the signature, the adversary perturbs the cell positions/regions for watermark removal attacks. We compare four types of attacks aiming to erase the signatures at different levels of watermarking and prevent the owner from claiming ownership: (i) swap location attacks (SLA) [91], which target to attack Row Parity framework by random swapping cell locations, (ii) constraint perturbation attacks (CPA), which target to attack Cell Scattering and ICMARKS: DW by moving cells along x/y axis if there is space; (iii) Optimization attacks (OA), which target to attack all watermarking frameworks by running another round of detailed placement; and (iv) adaptive region attacks (ARA), which target to attack ICMARKS: GW and ICMARKS framework by searching for less compacted regions and perturb cells around the region. We show ICMARKS is robust against all removal attacks, and the results are summarized in Fig. 7.11 and Fig. 7.12 for wirelength-driven and timing-driven placements respectively.

We skip Buffer Insertion [223] and the corresponding watermark removal attacks because (i) Buffer Insertion's [223] watermarked layout quality degradation is significantly higher than the considered threshold of 0.5% as in Table 7.5-7.7; (ii) Other watermarking frameworks insert signatures without modifying the netlist, whereas Buffer Insertion [223] encodes adding buffers into the netlist as watermarks. While the buffer removal attacks, which remove multiple cascaded buffers, can potentially erase Buffer Insertion's signature, our primary goal is evaluating the robustness of ICMARKS under attacks. Therefore, we did not include the attacks at the netlist level in this section.

❶ **Swap Location Attacks (SLA)**: In SLA [91], cells randomly exchange their locations with another set of cells. Then, a follow-up legalization and detailed placement compensate for the performance degradation and ensure the cells follow the design rules. In Fig. 7.11a and Fig. 7.11b, we randomly choose 0.1% and 0.5% cells from the layout and pair them to exchange their locations. As seen, even small location swaps result in huge performance



(a) Swap location attack (0.1%)

(b) Swap location attack (0.5%)

(c) Constraint perturbation attack (0.1%)

(d) Constraint perturbation attack (1%)

(e) Constraint perturbation attack (10%)

(f) Optimization attack

(g) Adaptive region attack (top-1)

(h) Adaptive region attack (top-5)

Figure 7.11. Watermarking performance under different attacks for wirelength-driven placement on the ISPD'2015 and ISPD'2019 benchmarks. The black dotted line in the two left subfigures denotes the quality degradation threshold of 1.005, and the black dotted line in the rightmost subfigure denotes the watermark extraction threshold of 90%.

degradation on modern IC layouts. In terms of the watermark extraction, Row Parity [93, 91], Cell Scattering [25], and ICMARKS: DW get WERs below 90% and failed to verify their ownership. Those watermarking frameworks spread the watermarks across the layout, where minor changes will subsequently modify most cell locations within the compact region. By embedding cells in the less compact region, ICMARKS: GW and ICMARKS are less sensitive to such changes. As a result, ICMARKS: GW and ICMARKS still maintain high WERs.

❷ **Constraint Perturbation Attacks (CPA)**: In CPA, cells shift their location along the x-axis for $\delta_x = 1$ or y-axis for δ_y set to one adjacent row-height if such movements do not result in overlapping with their neighbors. In Fig. 7.11c, Fig. 7.11d, and Fig. 7.11e, we move the positions of 0.1%, 1%, 10% of the cells have space to move in the layout. From here, we find that under CPA 0.1% attack, layout performance hit the boundary of the wirelength degradation (PWLR and RWLR) threshold. For the watermark extraction, ICMARKS, ICMARKS: GW, and Row Parity [93, 91] achieve over 90% WER. However, the WER of Cell Scattering [25] and ICMARKS: DW are subsequently lower than 90% because the constraint perturbation targets to remove the potential watermarked cell positions.

For the CPA 1% and 10% cell performance, however, the PWLR and RWLR degradations are greater than 1.005, meaning the attack failed to maintain the layout quality improvement from the design company's physical design optimizations. But ICMARKS: GW and ICMARKS still maintain higher than 90% WER and successfully help the design company to claim ownership of the layout.

❸ **Optimization Attacks (OA)**: This attack employs an additional optimization stage to remove the watermarks. The optimization is implemented through another round of detailed placement. It aims to change cell locations slightly for signature removal while maintaining the layout quality. As shown in Fig. 7.11f, the layout wirelength degradation in Cell Scattering [25], ICMARKS: GW, ICMARKS: DW, and ICMARKS are all below 1.005, indicating the attack preserves the layout quality. The WER of Cell Scattering [25] and ICMARKS: DW are below 90%, meaning OA successfully removed their signatures. In contrast, ICMARKS: GW and ICMARKS have over 90% WER, demonstrating their resiliency.

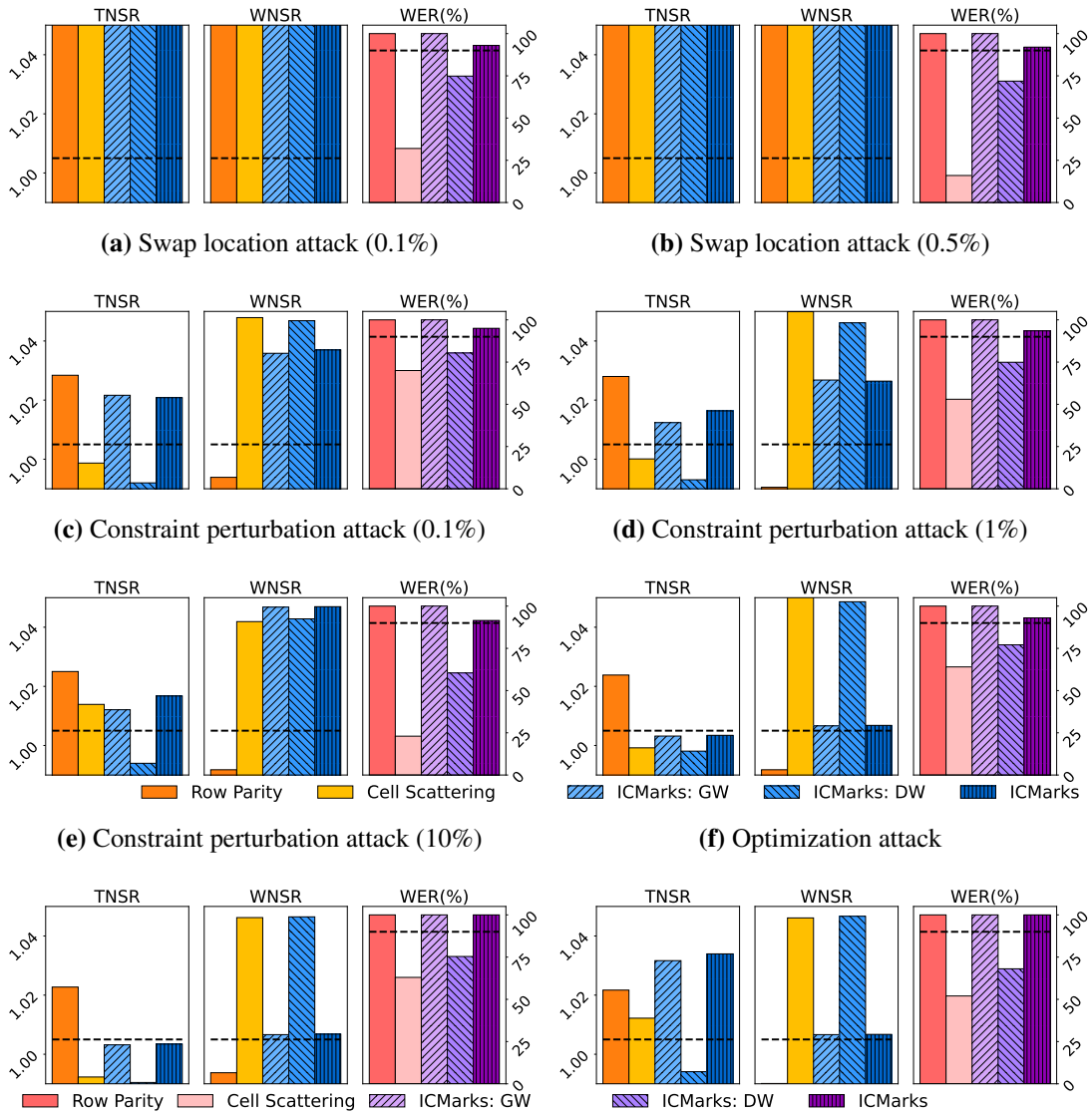
❹ **Adaptive Region Attacks (ARA)**: This attack aims to remove watermarks in ICMARKS:GW. The adversary has prior knowledge of how ICMARKS:GW performs the watermarking and has access to the hyperparameters used to search the watermarked region. The adversary operates on top of the watermarked layout. He tries to remove the inserted watermarks by moving cells within the searched top-1 or top-5 regions if there is room. In Fig. 7.11g and Fig. 7.11h, the

PWLR and RWLR degradation for ICMARKS: GW, ICMARKS: DW, and ICMARKS are around the threshold of 0.5% degradation, and the attacks do not significantly degrade the layout quality. The WER of ICMARKS: GW and ICMARKS remain over 90%. As the watermark insertion is performed on a non-watermarked layout, and the attack regions searched by ARA are on a watermarked layout, the watermark signatures are not the same. Therefore, ARA fails to remove ICMARKS: GW and ICMARKS's signature, even if the same region watermarking algorithm is employed. In contrast, the watermarks of Row Parity [93, 91], Cell Scattering [25], and ICMARKS:DW are spread across the layout, where minor changes in the compact area will subsequently modify the inserted watermarks. Therefore, these frameworks have compromised WER.

Attacks on Timing-driven Placement: In Fig. 7.12, the attack performance on timing-driven placement follows a similar trend as the wirelength-driven placement results in Fig. 7.11. ICMARKS and ICMARKS: GW remain resilient under all the removal attacks with a WER over 90%. Row Parity [93, 91] is also resilient to the attacks and has WER of over 90%. However, the watermarked layouts degrade the timing metrics (TNSR and WNSR) by $\geq 1\%$ in Table 7.7 for timing-driven placement. The signatures in the baseline Cell Scattering [25] and ICMARKS:DW are removed with WER $< 90\%$.

Watermark Forging Attack Instead of removing the watermarked signature, the adversary in a watermark forging attack counterfeits another set of watermarks on the watermarked layout and falsely claims his ownership. Row Parity [93, 91] and Buffer Insertion [223] techniques are not resilient to forging attacks. If the adversary has prior knowledge of the watermarking algorithm, they can easily counterfeit a different set of forged signatures from the row index ID of cells or add additional buffers into the layout for false ownership proof. For Cell Scattering [25], signatures are harder to counterfeit because forging the signature requires both random seeds and a non-watermarked layout. However, the inserted signatures can be easily erased by different watermark removal techniques in Section 7.5.3.

ICMARKS: GW is resilient to forging attacks because the region with the minimal



(a) Swap location attack (0.1%)

(b) Swap location attack (0.5%)

(c) Constraint perturbation attack (0.1%)

(d) Constraint perturbation attack (1%)

(e) Constraint perturbation attack (10%)

(f) Optimization attack

(g) Adaptive region attack (top-1)

(h) Adaptive region attack (top-5)

Figure 7.12. Watermarking performance under different attacks for timing-driven placement on the ICCAD'2015 benchmarks. The black dotted line in the two left subfigures denotes the quality degradation threshold of 1.005, and the black dotted line in the rightmost subfigure denotes the watermark extraction threshold of 90%.

evaluation score is unique to the original non-watermarked placement. The non-watermarked layout and the scoring parameters are kept confidential. The ICMARKS: GW signature verification requires the owner to provide that information to reproduce the watermark region R_w. Therefore, the adversary cannot counterfeit the watermarks without access to the non-watermark

layout. ICMARKS: DW also exhibits similar properties, where the signatures are encoded before detailed placement on the intermediate placement P_{itr}. As such, the adversary with only access to the watermarked layout cannot reproduce the watermarked cells or distance to forge the signature. ICMARKS, as a combination of ICMARKS: GW and ICMARKS: DW is also resilient to watermark forging attacks.

We evaluate the probability that an adversary forges the ICMARKS: GW signature, and we assume the adversary has prior knowledge of the scoring mechanism and watermarked region size. However, he does not have access to the scoring hyperparameters. Different from the Adaptive Region Attacks (ARA) in Watermark Removal Attacks, the adversary forges the signature by moving the cells in the searched region. We re-score the region using hyperparameters $\alpha = 25$, $\beta = 15$, and $\gamma = 10$. We rank the scores the adversary gets from low to high, where lower scores indicate the region is more ideal for him/her to forge the signature. We show the rank of the watermarked region appears in the re-scored watermarked layout in Fig. 7.13 on both ISPD'2019 [134] and ISPD'2015 [24] benchmarks. As seen, most of the layouts are ranked away from the top-5 regions, meaning the adversary cannot easily forge the watermarked region of cells.

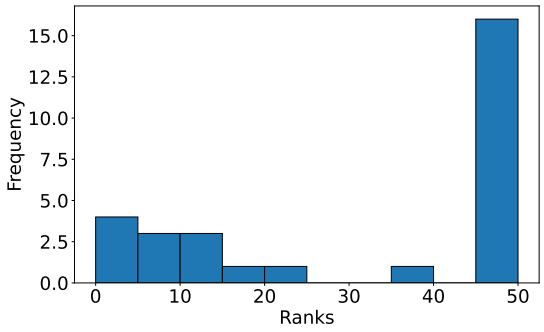


Figure 7.13. Rank distribution of the watermarked region appears in the re-scored watermarked region.

7.6 AUTOMARKS Experiments

This section introduces the experiment setups in Section 7.6.1. The watermarking results are presented in Section 7.6.2 and the robustness evaluations are presented in Section 7.6.3. We conduct an ablation study in Section 7.6.4 and provide visualizations in Section 7.6.5.

7.6.1 Experiment Setups

Hardware Infrastructure and Implementation Details AUTOMARKS is agnostic to the physical design algorithms. As a proof-of-concept, we employ the widely recognized open-source and state-of-the-art physical design framework with DREAMPlace [70] as the placement algorithm and CUGR [133] as the routing algorithm. We note that AUTOMARKS is agnostic to placement and routing algorithms, as it leverages a graph neural network to identify optimal layout positions that embed the signature with minimal performance degradation. This watermarking process is independent of the underlying placement and routing methods.

The training and inference of AUTOMARKS are performed on NVIDIA RTX A6000 GPUs with Ubuntu 22.04.3 LTS and Intel(R) Xeon(R) CPUs. The graph neural network model is a seven-layer graph convolution network with ReLU activation. As in Table 7.10, it is trained with the SGD optimizer, with the learning rate set to 0.01, weight decay set to 0.001, and the momentum value set to 0.9. It trains 30 epochs with a batch size set to 1280. The GNN is trained to sample 15, 20, 35, 50, 100, 200, and 500 nodes at each hop with detailed architecture in Table 7.11. For benchmarks optimizing for timing metrics, the GNN's main branch has five-layer graph convolutions (GConv) with ReLU activation. The follow-up sub-branches have two-layer graph convolutions with ReLU activation as in Table 7.12.

Benchmarks We evaluate AUTOMARKS's watermarking performance on the wirelength-driven ISPD 2015 [24], and ISPD'2019 [134] benchmark suites; and the timing-driven ICCAD'2015 [100] benchmark suite. Designs with fence region constraints are highlighted in blue in

Table 7.10. GNN training settings.

Variables	Settings
Epoch	30
Batch size	1280
Optimizer	SGD
Learning rate	0.001
Weight decay	0.001
Momentum	0.9
Loss function	MSE Loss

Table 7.11. GNN architecture $\mathbf{M}_{wl}$ for WL-driven search.

Layers	Input Dim	Output Dim	Activation
GConv1	8	64	ReLU
GConv2	64	64	ReLU
GConv3	64	64	ReLU
GConv4	64	64	ReLU
GConv5	64	64	ReLU
GConv6	64	64	ReLU
GConv7	64	1	ReLU

Table 7.12. GNN architecture $\mathbf{M}_{timing}$ for timing-driven watermark search.

Branch	Layers	Input Dim	Output Dim	Activation
$\mathbf{M}_m$	GConv1	9	64	ReLU
$\mathbf{M}_m$	GConv2	64	64	ReLU
$\mathbf{M}_m$	GConv3	64	64	ReLU
$\mathbf{M}_m$	GConv4	64	64	ReLU
$\mathbf{M}_m$	GConv5	64	64	ReLU
$\mathbf{M}_{tns}$, $\mathbf{M}_{wns}$	GConv6	64	64	ReLU
$\mathbf{M}_{tns}$, $\mathbf{M}_{wns}$	GConv7	64	1	ReLU

Tables 7.13 and 7.14. ICCAD'2015's [100] layouts in Table 7.15 do not have fence regions.

7.6.2 AUTOMARKS Results

Wirelength-driven Watermarking Fidelity and Transferability We compare the watermarking performance of AUTOMARKS and the baselines on the ISPD'2015 [24] benchmarks in Table 7.13, and the ISPD'2019 [134] benchmarks in Table 7.14. The encoded watermarks are extracted successfully, i.e., $\%WER = 100$ for all frameworks. We highlight that the AUTOMARKS is only trained on the `ispd19test6` design and inference on the rest of the designs, significantly reducing the watermark search time and improving transferability.

❶ **Comparison with Constraint-based WM:** Compared to AUTOMARKS that maintains watermarking fidelity, the baseline Row Parity [93, 91] and Cell Scattering [25] degrades the PWLR by 0.18% and 0.60% and the RWLR by 0.12% and 1.40% over the non-wm designs on ISPD'2015 [24] and ISPD'2019 [134] benchmarks respectively. The Row Parity [93, 91] and Cell Scattering [25] approaches do not consider the design constraints, like fence regions or macros when selecting the watermark cells, resulting in worse performance. Compared to AUTOMARKS, ICMarks [268] introduced slightly more degradations on ISPD'2019 [134] benchmarks. It is because AUTOMARKS learns the mapping from layout subgraphs to the actual PWLR improvement, which serves a better quality degradation estimation than the scoring function used in ICMarks [268].

❷ **Comparison with Invasive WM:** Compared to AUTOMARKS, the invasive watermarking

Table 7.13. Performance on the ISPD'2015 benchmarks. All the design watermarks are successfully extracted, i.e., WER = 100%. The PWLR and RWLR are the placement and routed wirelength rates over the original designs. The results in gray fail buffer insertion WM with significant degradation on the high-utilized designs (denoted with star).

Design	Cells	Nets	Row Parity [93, 91]		Cell Scattering [25]		Buffer Insertion [223]		ICMarks [268]		AUTOMARKS	
			PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓
perf_a ⋆	108K	115K	1.0045	1.0155	0.9978	0.9967	1.5289	2.3270	0.9972	**0.9873**	**0.9956**	0.9929
perf_b	113K	113K	1.0058	1.0267	1.0020	1.0001	1.0176	1.0745	**0.9890**	0.9901	0.9898	**0.9824**
dist_a	127K	134K	1.0015	0.9999	**0.9984**	**0.9965**	1.0995	1.1072	1.0004	1.0052	1.0001	1.0040
mult_b ⋆	146K	152K	1.0047	1.0199	**0.9991**	0.9923	1.8966	2.9374	0.9994	1.0028	1.0021	**0.9922**
mult_c ⋆	146K	152K	1.0031	1.0252	0.9980	1.0023	1.6003	2.7459	**0.9963**	**0.9979**	1.0010	1.0022
pci_a ⋆	30K	34K	1.0080	1.0340	**0.9931**	**0.9846**	1.6269	1.6849	0.9997	0.9950	0.9996	0.9974
pci_b ⋆	29K	33K	1.0069	1.1173	**0.9912**	0.9977	1.2239	1.8207	0.9951	1.0026	0.9973	**0.9885**
superblue11 ⋆	926K	936K	1.0052	1.0344	1.0278	1.0358	1.6930	3.7086	**0.9986**	**0.9992**	1.0073	1.0136
superblue16	680K	697K	1.0030	1.0210	**0.9988**	**0.9989**	1.1023	1.1377	1.0017	1.0204	1.0025	1.0129
perf_1	113K	113K	1.0035	1.0199	0.9985	0.9983	1.0150	1.0642	0.9964	0.9973	**0.9909**	**0.9956**
fft_1	35K	33K	1.0017	1.0260	1.0167	1.0156	1.0680	1.1457	**0.9671**	0.9673	0.9711	**0.9660**
fft_2 ⋆	35K	33K	1.0050	1.0260	1.0148	1.0114	1.4603	1.4476	**0.9767**	**0.9770**	0.9854	0.9823
fft_a ⋆	34K	32K	1.0121	1.0200	1.0024	0.9933	1.8203	2.1923	0.9939	0.9895	**0.9839**	**0.9773**
fft_b ⋆	34K	32K	1.0018	1.0075	0.9961	1.0030	1.9859	2.5615	0.9909	**0.9890**	**0.9869**	1.0039
mult_1	160K	159K	1.0050	1.0238	1.0077	1.0065	1.0464	1.0631	**0.9753**	**0.9744**	0.9773	0.9769
mult_2	160K	159K	1.0033	1.0199	0.9984	0.9967	1.0736	1.1147	0.9852	0.9900	**0.9831**	**0.9872**
mult_a ⋆	154K	154K	1.0037	1.0105	0.9995	0.9968	1.3862	1.6738	0.9973	**0.9916**	**0.9965**	0.9934
superblue12	1293K	1293K	1.0031	1.0067	0.9979	0.9956	1.0683	1.1044	**0.9854**	**0.9732**	0.9869	0.9783
superblue14	634K	620K	1.0020	1.0057	0.9991	0.9981	1.0212	1.0286	**0.9887**	**0.9867**	1.0083	1.0037
superblue19	522K	512K	1.0025	1.0077	1.0001	1.0005	1.0295	1.0672	**0.9814**	**0.9809**	1.0100	1.0563
Average	-	-	1.0043	1.0231	1.0018	1.0012	1.0536	1.0901	**0.9908**	**0.9908**	0.9937	0.9966

Table 7.14. Performance on the ISPD'2019 benchmarks. All the design watermarks are successfully extracted, i.e., WER = 100%. The PWLR and RWLR are the placement and routed wirelength rates over the original designs.

Design	Cells	Nets	Row Parity [93, 91]		Cell Scattering [25]		Buffer Insertion [223]		ICMarks [268]		AUTOMARKS	
			PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓	PWLR ↓	RWLR ↓
ispd19test1	9K	3K	1.0060	1.0129	0.9978	0.9998	1.0394	1.0619	**0.9955**	1.0015	1.0026	**0.9989**
ispd19test2	73K	72K	1.0184	1.0201	1.0096	1.0016	1.0127	1.0408	0.9988	0.9999	**0.9921**	**0.9927**
ispd19test3	8K	9K	1.0130	1.0475	1.0180	1.0193	1.0582	1.0801	1.0059	1.0045	**0.9875**	**0.9865**
ispd19test4	151K	146K	1.0017	1.0050	0.9999	0.9998	1.1452	1.2494	**0.9957**	**0.9907**	0.9970	1.0001
ispd19test5	29K	29K	1.0102	1.0556	1.0998	1.0975	0.9859	1.0121	1.0013	0.9998	**0.9982**	**0.9953**
ispd19test6	181K	180K	1.0032	1.0106	**0.9994**	0.9993	1.0044	1.1108	1.0023	1.0026	1.0000	**0.9971**
ispd19test7	362K	359K	1.0028	1.0160	1.0136	1.0109	**1.0003**	1.0717	1.0050	1.0054	1.0150	1.0139
ispd19test8	543K	538K	1.0001	1.0082	0.9966	0.9958	1.0118	1.0806	0.9961	**0.9929**	**0.9945**	0.9996
ispd19test9	903K	895K	1.0072	1.0108	1.0108	1.0095	1.0164	1.0814	1.0023	1.0023	**0.9973**	**1.0000**
ispd19test10	903K	895K	1.0025	1.0090	1.0043	1.0108	1.0378	1.0982	0.9972	**0.9966**	**0.9972**	1.0002
Average	-	-	1.0065	1.0194	1.0060	1.0140	1.0304	1.0871	1.0000	0.9996	**0.9981**	**0.9982**

Buffer Insertion [223] significantly degrades the PWLR and RWLR by 5.36% and 9.01% on the ISPD'2015 [24] designs; and by 3.04% and 8.71% on the ISPD'2019 [134] benchmarks over the non-watermarked designs. As the layout is highly utilized, adding redundant buffers results in significant cell placement reordering to accommodate the watermark cells, requiring more routing resources to connect the additional components than constraint-based AUTOMARKS.

Timing-driven Watermarking Fidelity and Transferability The timing-driven watermarking

performance on ICCAD'15 [100] benchmarks is in Table 7.15. The encoded watermarks for all methodologies are extracted successfully, $\%WER = 100$ for all designs. Similar to Section 7.6.2, AUTOMARKS's GNN is trained only on the `superblue5` design and inferences on the rest of the designs, with β set to 1.010. Similar to wirelength-driven watermarking, AUTOMARKS achieves 1.24% lower WNSR compared to the best-performing constraint-based Row Parity [93, 91] and invasive watermarking Buffer Insertion [223] frameworks. Such improvements come from leveraging GNN to learn the subgraph structure yielding better performance after watermark insertion. Compared to ICMarks, AUTOMARKS demonstrates comparable watermarking performance but less searching time by leveraging the GNN to predict the watermarking performance in parallel as in Figure 7.14.

Table 7.15. Performance on ICCAD'2015 benchmarks. All the design watermarks are successfully extracted, i.e., WER = 100%. The TNSR and WNSR are the total and worst negative slack rates over the original designs.

Design	Cells	Nets	Row Parity [93, 91]		Cell Scattering [25]		Buffer Insertion [223]		ICMarks [268]		AUTOMARKS	
			TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓	TNSR ↓	WNSR ↓
superblue1	1209K	1215K	1.0053	1.0094	1.0119	1.0026	1.0083	1.0020	**0.8283**	1.0021	1.0023	**1.0011**
superblue3	1213k	1224k	1.0053	1.0176	1.0066	0.9891	1.1035	1.0988	0.9363	**0.9047**	**0.9283**	1.0015
superblue4	795k	802k	1.1299	0.9914	0.9698	1.0191	1.1954	1.1995	**0.9221**	1.0094	1.0038	**1.0012**
superblue5	1086k	1100k	0.9801	**0.9887**	1.0126	0.9977	1.0932	1.1174	**0.9159**	1.0508	1.0025	1.0068
superblue7	1931k	1933k	0.9801	0.9932	0.9979	1.0212	0.9933	1.0199	**0.9636**	**0.9727**	1.0043	1.0025
superblue10	1876k	1898k	1.0099	**1.0041**	1.0218	1.0141	1.0194	1.2918	1.0069	1.0072	**1.0034**	1.0052
superblue16	981k	999k	0.9814	1.1271	**0.9067**	1.3274	1.0109	1.0877	1.0069	1.0072	1.0032	**1.0043**
superblue18	768k	771k	1.0068	0.9965	0.9981	1.0064	1.0438	1.0613	**0.9254**	**0.9449**	0.9841	0.9980
Average	-	-	1.0114	1.0151	0.9900	1.0424	1.0566	1.1063	**0.9366**	**0.9867**	0.9911	1.0025

Efficiency Figure 7.14 and Table 7.16 compare the watermark search time for ICMarks [268] and AUTOMARKS. For smaller designs in wirelength-driven watermark search, the search time of AUTOMARKS is very close to ICMarks [268]. However, for the large designs ($\geq 500k$ cells) in Figure 7.14, the average search time of AUTOMARKS is 176.38s, whereas ICMarks requires 320.93s. As ICMarks uses a fixed window size to traverse and score the layout, while AUTOMARKS employs GNN to batch-predict the region node scores, AUTOMARKS reduces the watermark search time for large designs by 45.04%. In timing-driven watermark search, all of the layouts are large designs with more than 500k cells. In Table 7.16, ICMarks [268] takes an average of 72.37s to find an ideal position to encode watermarks, whereas the average

time of AUTOMARKS is 53.99s. As such, AUTOMARKS reduces the search time by 25.39% on timing-driven layouts. However, we note there are a few designs (superblue1, superblue5, and superblue7) where ICMarks demonstrates close runtime with AUTOMARKS. It is primarily because the macros have dense distributions in which the ICMarks avoids watermarking on those regions and skips the scoring evaluation. By excluding those designs in which ICMarks have a higher overlap rate ($> 70\%$), evaluated by skipped evaluations due to macro overlapping over the total number of scoring, AUTOMARKS reduces the search time of ICMarks by 41.52%.

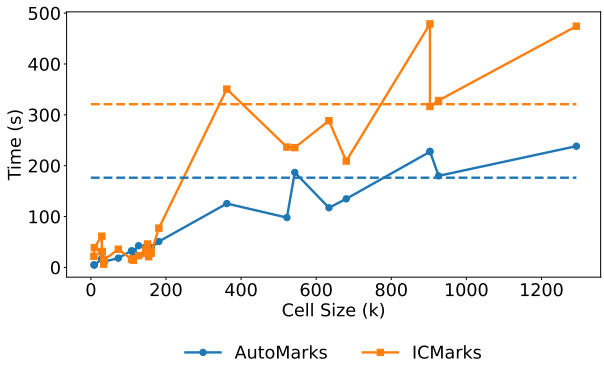


Figure 7.14. The wirelength-driven watermark search time for different designs in ISPD'2015 and ISPD'2019. The dotted line is the average search time of the large designs ($\geq 500k$ cells).

Table 7.16. Watermark search runtime in ICCAD'2015.

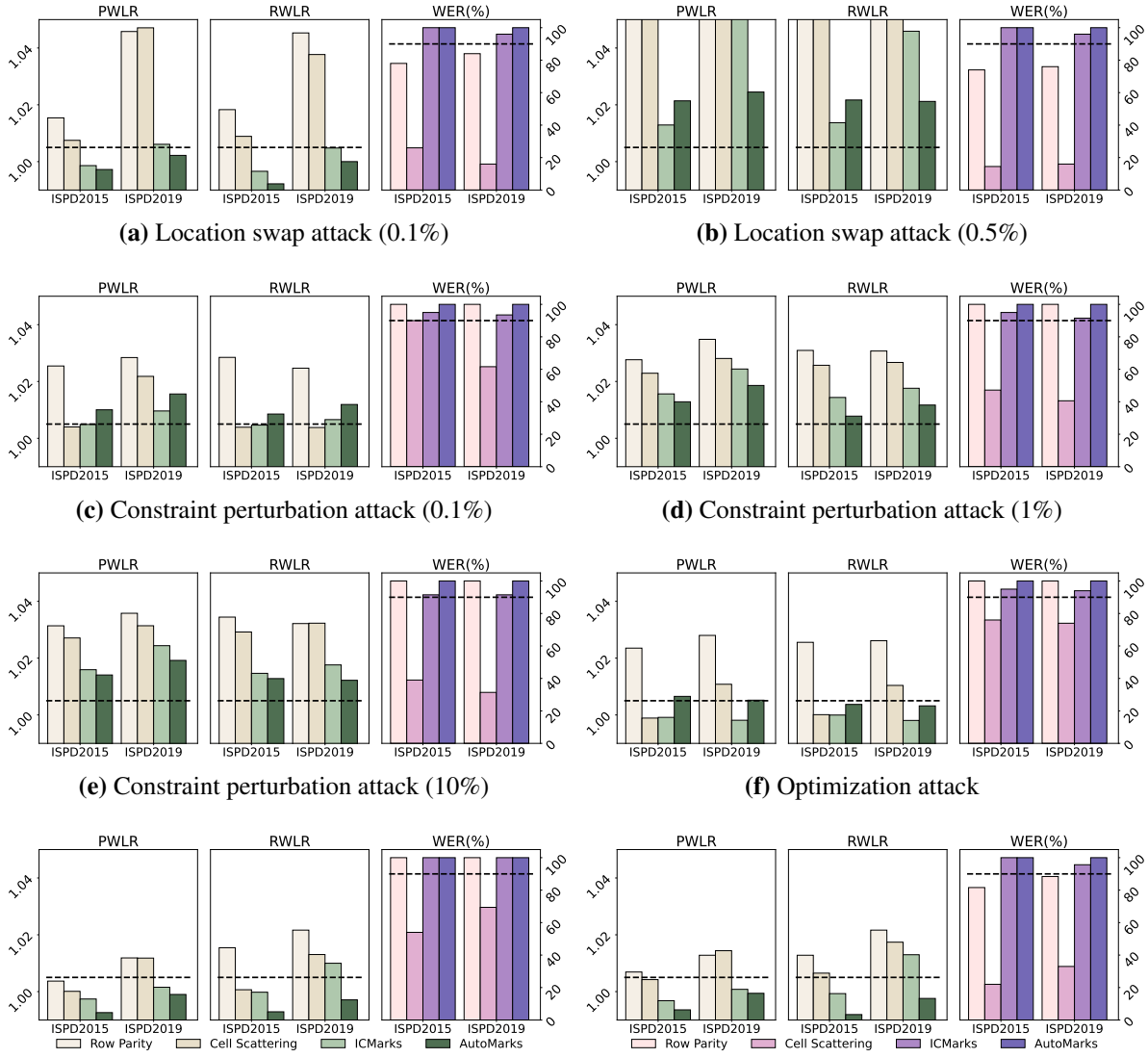
Design	ICMarks(s)	AUTOMARKS(s)	Overlap Rate
superblue1	47.68	50.24	78.30%
superblue3	144.25	70.24	41.66%
superblue4	32.68	45.29	60.04%
superblue5	54.34	40.24	82.40%
superblue7	53.29	51.25	73.53%
superblue10	282.40	100.24	60.23%
superblue16	116.18	80.29	61.89%
superblue18	35.24	27.24	61.48%
Average (Overall)	72.37	53.99	-
Average (Overlap Rate $\leq 70\%$)	98.05	57.34	-

7.6.3 AUTOMARKS's Robustness

Watermark Removal Attacks on Benchmarks Optimizing Wirelength Figure 7.15 evaluates the impact of watermark removal attacks on AUTOMARKS and the baseline approaches. The **Location swap attack** targeting Row Parity [93, 91] selects 0.1% of the total cells randomly and swaps their locations. The **Constraint perturbation attack** targeting Cell Scattering [25] moves 10% of the cells randomly along the x-axis for one unit or the y-axis for one-row height if the cells do not overlap with the neighboring cells. The **Optimization attack** targeting all frameworks applies another round of detailed placement on the watermarked layout to remove signatures. The **Adaptive region attack** targeting ICMarks [268] and AUTOMARKS, assumes the adversary knows the size of the watermark region and uses the evaluation function of ICMarks [268] to identify the watermark regions. Then, the adversary perturbs cells in the top-5 regions to remove watermarks. We do not consider attacks on Buffer Insertion as it incurs significantly more performance degradation for watermark insertion than the constraint-based watermarking approaches.

AUTOMARKS is resilient to all attacks and maintains the watermark extraction rates of 100% even when the quality metrics PWLR and RWLR are greatly compromised; because as long as the watermark cells are within the watermark region, it is more robust toward such slight perturbation of cell locations. In contrast, Row Parity is vulnerable to Location swap attacks; Cell Scattering is vulnerable to Location swap attacks and Constraint perturbation attacks. While ICMarks employs a two-level watermarking framework to strengthen the watermarking strength, the detailed watermarking in ICMarks is vulnerable to watermark removal attacks. As a result, the overall WER for ICMarks slightly degrades.

Watermark Removal Attacks on Benchmarks Optimizing Timing Metrics The watermarked layout performance after attacks for different methodologies are in Figure 7.16. Similar to benchmarks optimizing for wirelength, we evaluated the impact of four attack types: swap location, constraint perturbation, optimization, and adaptive region attacks. The attack setups are



(a) Location swap attack (0.1%)

(b) Location swap attack (0.5%)

(c) Constraint perturbation attack (0.1%)

(d) Constraint perturbation attack (1%)

(e) Constraint perturbation attack (10%)

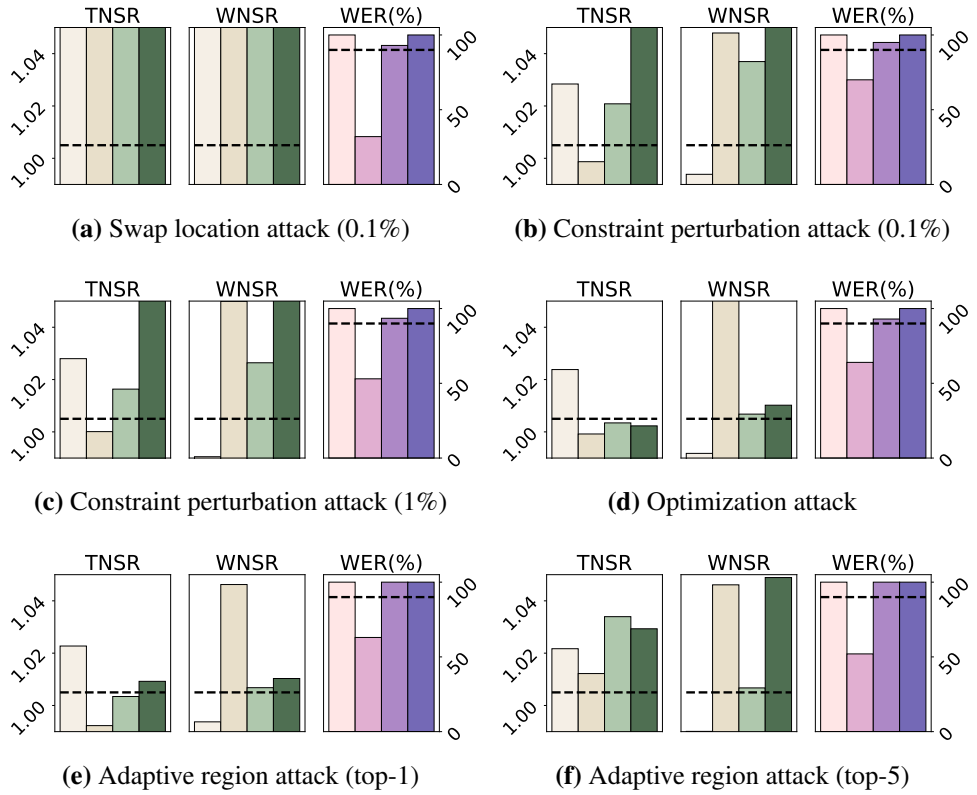
(f) Optimization attack

(g) Adaptive region attack (top-1)

(h) Adaptive region attack (top-5)

Figure 7.15. Watermarking performance under different attacks for wirelength-driven placement on the ISPD'2015 and ISPD'2019 benchmarks. The black dotted line in the two left subfigures denotes the quality degradation threshold of 1.005, and the black dotted line in the rightmost subfigure denotes the watermark extraction threshold of 90%.

detailed in Section 7.6.3. As for attack hyperparameters, we swap the location of 0.1% cells for swap location attack, perturb the location of 0.1% and 1% cells for constraint perturbation attack, and choose top-1 and top-5 regions re-evaluated by ICMarks [268] to perturb cells within the regions. As seen, similar to the wirelength-driven benchmark, AUTOMARKS is resilient against those attacks, maintaining a high extractability of the encoded signatures unless the layout quality



(a) Swap location attack (0.1%)

(b) Constraint perturbation attack (0.1%)

(c) Constraint perturbation attack (1%)

(d) Optimization attack

(e) Adaptive region attack (top-1)

(f) Adaptive region attack (top-5)

Figure 7.16. Watermarking performance under different attacks for timing-driven placement on the ICCAD'2015 benchmarks. The black dotted line in the two left subfigures denotes the quality degradation threshold of 1.005, and the black dotted line in the rightmost subfigure denotes the watermark extraction threshold of 90%.

is greatly compromised. Post-routing ECO may be applied to improve layout performance but can potentially disrupt watermark insertion. To mimic such scenarios, we conducted the optimization attack shown in Figure 7.16d, where an additional round of optimization was performed to remove the signature. Our results show that AUTOMARKS remains resilient in these cases and continues to provide sufficient ownership proof.

Watermark Forging Attacks To forge the signature, the adversary needs to provide the watermark region by reproducing the GNN inference results. However, the adversary does not have access to the design used for GNN training, making it difficult to counterfeit the signature. Therefore, AUTOMARKS is resilient to the forging attacks.

7.6.4 Ablation Study

This subsection provides ablation studies over the different hyperparameter choices in AUTOMARKS's performance on the ISPD'2019 [134] benchmarks as a proof-of-concept.

The Effectiveness of Node Feature: We analyze how nonnative node features impact the GNN performance. In Table 7.17, we skip the cell location, size, and name as the node feature and report their PWLR and RWLR performance respectively. Other settings follow the default ones in AUTOMARKS. Cell location and name have a more significant impact on the performance of the AUTOMARKS compared to cell size. Excluding the cell location and name into the node feature construction results in 2.9% and 3.2% PWLR and 3.4% and 3.3% RWLR degradation, respectively. In contrast, excluding the cell size only results in 0.2% degradation. This is mainly because the cell name indicates the type of cells, e.g., standard cells and macros, and cell location indicates the position of the cells. Both attributes are essential in helping AUTOMARKS learn which node is the ideal candidate for watermark insertion.

The Effectiveness of learning rate: We analyze how different GNN learning rates (LR) would affect AUTOMARKS's performance in Table 7.17. Following Sec. 7.6.2, the AUTOMARKS is trained on `ispd19test6` design and inference on the rest of the benchmarks. Here, we change the learning rate from 0.01 to 0.001 and 0.1. When the learning rate is increased from 0.001 to 0.01, the GNN learns to capture the node connections and predict the watermarking fidelity better. However, increasing the learning rate from 0.01 to 0.1 introduced marginal quality changes.

The Effectiveness of β Threshold: We analyze how different label threshold β choice would impact the watermarking fidelity performance. In Table 7.17, we change the β from 0.01 to 0.005 and 0.2, and show the PWLR and RWLR performance. Other settings follow the default ones in AUTOMARKS. As seen, increasing the β from 0.005 to 0.01 introduced significant PWLR and RWLR performance improvement. It is because more training nodes are normalized into the range of [0,1] that help AUTOMARKS to learn more diverse score predictions. However, increasing the β from 0.01 to 0.02 does not introduce significant watermarking fidelity changes.

The effectiveness of γ: We analyze how different γ choices would affect the larger designs' (i.e. cell size larger than 900k) performance in Table 7.17. Other settings follow the default ones in AUTOMARKS. As seen, adjusting the γ from 0.2 to 0 and 0.5 will change the AUTOMARKS performance on the large layout slightly.

The effectiveness of GNN layers: We analyze how different numbers of the GNN layers would affect the watermarking fidelity performance in Table 7.17. Here, we fix the watermark region size $N = 10$ to be the same as Section 7.6.2 and change the number of GNN layers used for training and inference from $5 \rightarrow 10$. Increasing the layer from 5 to 7 significantly improved the watermarking performance. When the layer number is set to 5, most of the cells are within the watermark region, and the GNN fails to learn the node features on the region boundary, which will be expelled outside the region and play an essential role in quality degradation. Increasing the number of layers results in better watermarking performance. However, increasing the number from 7 to 10 introduced marginal quality improvement and more computational overheads. Therefore, AUTOMARKS employs a graph neural network with 7 layers.

Table 7.17. The effectiveness of different node feature constructions, learning rate choices, β and γ choices, and layer numbers on AUTOMARKS's performance.

GNN Feature		PWLR	RWLR
Node Feature	All	0.9981	0.9982
	w/o cell location	1.0280	1.0312
	w/o cell size	1.0002	0.9977
	w/o cell name	1.0303	1.0304
Learning Rate	0.001	1.0004	1.0009
	0.01	0.9981	0.9982
	0.1	0.9990	0.9998
β Choices	0.005	1.0078	1.0089
	0.01	0.9981	0.9982
	0.02	1.0013	0.9990
γ Choices	0.0	0.9974	1.0002
	0.2	0.9972	1.0001
	0.5	0.9974	1.0002
#Layers	5	1.1033	1.1131
	7	0.9981	0.9982
	10	0.9997	0.9999

7.6.5 Visualizations

In this section, we provide additional visualizations of AUTOMARKS's label and the training procedure.

Visualization of the labels: We visualize the training dataset `ispd19test6`'s labels L distribution in Fig 7.17. As seen, watermarking on more than half of the nodes yields good watermarking performance. The critical step is to ensure the graph neural network learns the subgraph structure of the good nodes and can thus ensure the watermarking fidelity.

Loss curve during GNN training: We show the loss curve in Figure 7.18, in which the model is trained on `ispd19test6` [134] design and the loss in evaluated using Equation 7.11. It demonstrates the graph neural network gradually learns to predict the quality degradations of a given subgraph. The loss converges during the GNN training.

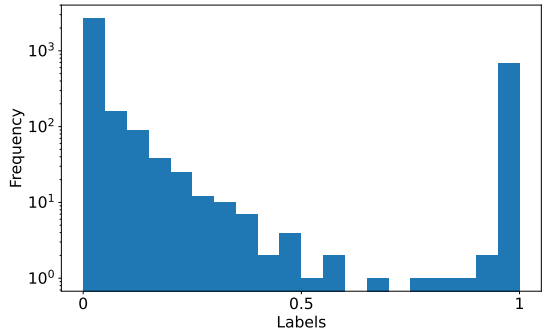


Figure 7.17. Histogram distribution of AUTOMARKS's labels.

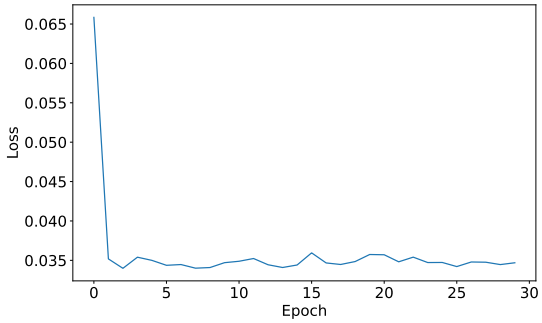
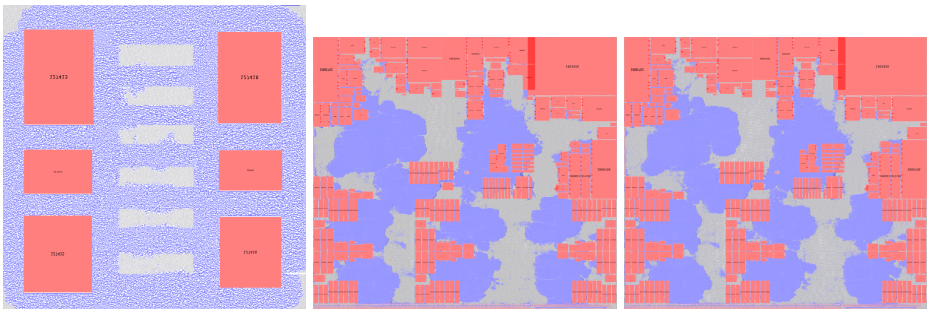


Figure 7.18. Loss curve during the AUTOMARKS's GNN training.

Stealthiness: To verify the stealthiness of AUTOMARKS, we include the watermarked layout and the non-watermarked layout from both ISPD'2015 [24], ISPD'2019 [134], and ICCAD'2015 [100] benchmarks in Figure 7.19. As seen, the watermarks are invisible upon inspection, and the adversary cannot differentiate between a non-watermarked layout and a layout watermarked by AUTOMARKS.



(a) test4(WM) **(b)** test4(non-WM) **(c)** dist_a(WM)



(d) dist_a(non-WM) **(e)** superblue7(WM) **(f)** superblue7(non-WM)

Figure 7.19. Watermarked and Non-watermarked examples.

7.7 Conclusion

In this chapter, we first present ICMARKS, a robust watermarking framework for integrated circuits physical design IP protection in the supply chain. We first introduce a *Global Watermarking* method that identifies the watermarked region with insignificant performance degradation; then, we propose an independent *Detailed Watermarking* technique to select cells that do not overlap with neighbors after perturbation to encode watermarks. Based on these methods, we develop ICMARKS, which combines the best attributes of both *Global* and *Detailed Watermarking*, thereby achieving minimal quality degradation with augmented robustness. Extensive experiments on ISPD'2015 [24], ISPD'2019 [134], and ICCAD'2015 [100] benchmarks demonstrate that ICMARKS successfully inserts watermarks without compromising layout quality. Furthermore, we showcased ICMARKS's resiliency against watermark removal and forging attacks through comprehensive attack evaluations.

Then, this chapter presents AUTOMARKS, an automated, highly transferable, and metric-agnostic watermarking framework for physical design. Leveraging graph neural networks for the watermark region search, AUTOMARKS significantly reduces the search time while preserving watermarking fidelity for both wirelength-driven and timing-driven layouts. Extensive evaluations on the ISPD'15 and ISPD'19, ICCAD'15 benchmarks demonstrate the effectiveness and transferability of our proposed framework compared to existing physical design watermarking approaches. AUTOMARKS is resilient against watermarking removal and forging attacks with 100% watermark extraction rate for proof of ownership.

7.8 Acknowledgements

Chapter 7 is a reprint of the material as it appears in: 1) Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar, "ICMarks: A Robust Watermarking Framework for Integrated Circuit Physical Design IP Protection," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025; and 2) Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar, "AutoMarks: A GNN-based Automated Physical Design Watermarking Framework," in *ACM Transactions on Design Automation of Electronic Systems*, 2025. The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] Sahar Abdelnabi and Mario Fritz. Adversarial watermarking transformer: Towards tracing text provenance with data hiding. In *42nd IEEE Symposium on Security and Privacy*, 2021.

[2] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[3] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

[4] Ako Muhamad Abdullah et al. Advanced encryption standard (aes) algorithm to encrypt and decrypt data. *Cryptography and Network Security*, 16(1):11, 2017.

[5] Anthony Agnesina, Kyungwook Chang, and Sung Kyu Lim. VLSI placement parameter optimization using deep reinforcement learning. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[6] Ramy E Ali, Jinhyun So, and A Salman Avestimehr. On polynomial approximations for privacy-preserving and verifiable relu networks. *arXiv preprint arXiv:2011.05530*, 2020.

[7] David M Allen. Mean square error of prediction as a criterion for selecting variables. *Technometrics*, 13(3):469–475, 1971.

[8] Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu. Graph neural networks: A powerful and versatile tool for advancing design, reliability, and security of ics. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ASPDAC '23, page 83–90, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. Gnn-re: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2435–2448, 2021.

[10] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. Gnn-re: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2435–2448, 2022.

[11] David R Anderson, Kenneth P Burnham, and William L Thompson. Null hypothesis testing: problems, prevalence, and an alternative. *The journal of wildlife management*, pages 912–923, 2000.

[12] Apple Machine Learning Research. Introducing Apple's On-Device and Server Foundation Models. [Online]. Available: https://machinelearning.apple.com/research/introducing-apple-foundation-models, 2024.

[13] Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, et al. Multi-lingual evaluation of code generation models. *arXiv preprint arXiv:2210.14868*, 2022.

[14] Erman Ayday, Emre Yilmaz, and Arif Yilmaz. Robust {Optimization-Based} watermarking scheme for sequential data. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, pages 323–336, 2019.

[15] Priyanka Bagul, Vandana Inamdar, et al. Hardware obfuscation based watermarking technique for ipr ownership identification. *International Journal of Reconfigurable Computing*, 2023, 2023.

[16] Fujun Bai, Zhiqiang Gao, Yi Xu, and Xueyu Cai. A watermarking technique for hard ip protection in full-custom ic design. In *2007 International Conference on Communications, Circuits and Systems*, pages 1177–1180. IEEE, 2007.

[17] Severine Baudry, Jean-Francois Delaigle, Bulent Sankur, Benoit Macq, and Henri Maitre. Analyses of error correction strategies for typical communication channels in watermarking. *Signal Processing*, 81(6):1239–1250, 2001.

[18] Georg T Becker, Markus Kasper, Amir Moradi, and Christof Paar. Side-channel based watermarks for integrated circuits. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 30–35. IEEE, 2010.

[19] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[20] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48):E11221–E11230, 2018.

[21] Franziska Boenisch. A systematic review on model watermarking for neural networks. *Frontiers in big Data*, 4:729663, 2021.

[22] Rishi Bommasani et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[23] Neil Burgess et al. Bfloat16 processing for neural networks. In *ARITH*. IEEE, 2019.

[24] Ismail S Bustany, David Chinnery, Joseph R Shinnerl, and Vladimir Yutsis. ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, pages 157–164, 2015.

[25] Xueyu Cai, Zhiqiang Gao, Fujun Bai, and Yi Xu. A watermarking technique for hard ip protection in post-layout design level. In *2007 7th International Conference on ASIC*, pages 1317–1320, 2007.

[26] Zeju Cai, Jianguo Chen, Wenqing Chen, Weicheng Wang, Xiangyuan Zhu, and Aijia Ouyang. F-codellm: A federated learning framework for adapting large language models to practical software development. In *ICSE*, pages 416–417, 2024.

[27] Rajat Subhra Chakraborty and Swarup Bhunia. Rtl hardware ip protection using key-based control and data flow obfuscation. In *2010 23rd International Conference on VLSI Design*, pages 405–410. IEEE, 2010.

[28] David J Chalmers. Syntactic transformations on distributed representations. *Connectionist Natural Language Processing: Readings from Connection Science*, pages 46–55, 1992.

[29] Pik-Wah Chan and Michael R Lyu. A dwt-based digital video watermarking scheme with error correcting code. In *International Conference on Information and Communications Security*, pages 202–213. Springer, 2003.

[30] Huili Chen et al. Specmark: A spectral watermarking framework for ip protection of speech recognition systems. In *INTERSPEECH*, pages 2312–2316, 2020.

[31] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. Deepattest: An end-to-end attestation framework for deep neural networks. In *ISCA*, pages 487–498, 2019.

[32] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. Intellectual property protection of deep learning systems via hardware/software co-design. *IEEE Design & Test*, 2023.

[33] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 105–113, 2019.

[34] Mark Chen et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[35] Yanni Chen and Keshab K Parhi. Small area parallel chien search architectures for long bch codes. *Ieee Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):545–549, 2004.

[36] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2018.

[37] Wing-Kai Chow, Jian Kuang, Xu He, Wenzan Cai, and Evangeline FY Young. Cell density-driven detailed placement with displacement constraint. In *Proceedings of the 2014 on International symposium on physical design*, pages 3–10, 2014.

[38] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. *arXiv preprint arXiv:2306.09194*, 2023.

[39] Alberto Colorni, Marco Dorigo, Francesco Maffioli, Vittorio Maniezzo, GIOVANNI Righini, and Marco Trubian. Heuristics from nature for hard combinatorial optimization problems. *International Transactions in Operational Research*, 3(1):1–21, 1996.

[40] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[41] Aijiao Cui, Chip-Hong Chang, and Li Zhang. A hybrid watermarking scheme for sequential functions. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 2333–2336, 2011.

[42] Amy B Cyphert. Generative ai, plagiarism, and copyright infringement in legal documents. *Minn. JL Sci. & Tech.*, 25:49, 2023.

[43] Earl Danielson, Josh H Golden, Eric W McFarland, Casper M Reaves, W Henry Weinberg, and Xin Di Wu. A combinatorial approach to the discovery and optimization of luminescent materials. *Nature*, 389(6654):944–948, 1997.

[44] Jerome Darbon, Bulent Sankur, and Henri Maitre. Error correcting code performance for watermark protection. In *Security and watermarking of multimedia contents III*, volume 4314, pages 663–672. SPIE, 2001.

[45] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *ASPLOS*, pages 485–497, 2019.

[46] Upoma Das, M Sazadur Rahman, N Nalla Anandakumar, Kimia Zamiri Azar, Fahim Rahman, Mark Tehranipoor, and Farimah Farahmandi. Psc-watermark: Power side channel based ip watermarking using clock gates. In *2023 IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2023.

[47] Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, et al. Scalable watermarking for identifying large language model outputs. *Nature*, 634(8035):818–823, 2024.

[48] W. Deacon. Virtualization for the masses: Exposing kvm on android. In *KVM Forum*, 2020.

[49] Jeffrey Dean et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[50] Tim Dettmers et al. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[51] Tim Dettmers et al. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[52] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[53] Abdulrahman Diaa, Toluwani Aremu, and Nils Lukas. Optimizing adaptive attacks against content watermarks for language models. *arXiv preprint arXiv:2410.02440*, 2024.

[54] Divyanshu Divyanshu, Rajat Kumar, Danial Khan, Selma Amara, and Yehia Massoud. Fsm inspired unconventional hardware watermark using field-assisted sot-mtj. *IEEE Access*, 11:8150–8158, 2023.

[55] Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. Exploiting llm quantization. *Advances in Neural Information Processing Systems*, 37:41709–41732, 2024.

[56] Jonathan Ellman, Barry Stoddard, and Jim Wells. Combinatorial thinking in chemistry and biology. *Proceedings of the National Academy of Sciences*, 94(7):2779–2782, 1997.

[57] Elias Frantar et al. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[58] Upma Gandhi, Ismail Bustany, William Swartz, and Laleh Behjat. A reinforcement learning-based framework for solving physical design routing problem in the absence of large test sets. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 2019.

[59] Akshay Gangal, Mengmei Ye, and Sheng Wei. Hybridtee: Secure mobile dnn execution using hybrid trusted execution environment. In *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2020.

[60] Leo Gao et al. A framework for few-shot language model evaluation, September 2021.

[61] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *EMNLP*, 2021.

[62] Aayush Garg, Renzo Degiovanni, Mike Papadakis, and Yves Le Traon. On the coupling between vulnerabilities and llm-generated mutants: A study on vul4j dataset. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 305–316. IEEE, 2024.

[63] GitHub. GitHub Copilot. https://docs.github.com/en/copilot, 2023.

[64] Thibaud Gloaguen, Nikola Jovanović, Robin Staab, and Martin Vechev. Discovering clues of spoofed lm watermarks. *arXiv preprint arXiv:2410.02693*, 2024.

[65] Google. Android virtualization framework (avf) overview, 2024.

[66] Dibakar Gope, David Mansell, Danny Loh, and Ian Bratt. Highly optimized kernels and fine-grained codebooks for llm inference on arm cpus. *arXiv preprint arXiv:2501.00032*, 2024.

[67] Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[68] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326. Springer, 2016.

[69] Chenchen Gu, Xiang Lisa Li, Rohith Kuditipudi, Percy Liang, and Tatsunori Hashimoto. Auditing prompt caching in language model apis. *arXiv preprint arXiv:2502.07776*, 2025.

[70] Jiaqi Gu, Zixuan Jiang, Yibo Lin, and David Z Pan. DREAMPlace 3.0: Multi-electrostatics based robust VLSI placement with region constraints. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[71] Biyang Guo, Xin Zhang, Ziyuan Wang, Minqi Jiang, Jinran Nie, Yuxuan Ding, Jianwei Yue, and Yupeng Wu. How close is chatgpt to human experts? comparison corpus, evaluation, and detection. *arXiv preprint arxiv:2301.07597*, 2023.

[72] Jahid Hasan. Overview and applications of zero knowledge proof (zkp). *International Journal of Computer Science and Network*, 8(5):2277–5420, 2019.

[73] Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. What matters in transformers? not all attention is needed. *arXiv preprint arXiv:2406.15786*, 2024.

[74] Xuanli He et al. Cater: Intellectual property protection on text generation apis via conditional watermarks. *Advances in Neural Information Processing Systems*, 35:5431–5445, 2022.

[75] Xuanli He et al. Protecting intellectual property of language generation apis with lexical watermark. In *AAAI*, 2022.

[76] Xuanli He, Qiongkai Xu, Lingjuan Lyu, Fangzhao Wu, and Chenguang Wang. Protecting intellectual property of language generation apis with lexical watermark. *arXiv preprint arXiv:2112.02701*, 2021.

[77] Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen. Combinatorial optimization in vlsi design. *Combinatorial Optimization*, pages 33–96, 2011.

[78] Nasimeh Heydaribeni, Xinrui Zhan, Ruisi Zhang, Tina Eliassi-Rad, and Farinaz Koushanfar. Distributed constrained combinatorial optimization leveraging hypergraph neural networks. *Nature Machine Intelligence*, pages 1–9, 2024.

[79] Hao-Hsiang Hsiao, Yi-Chen Lu, Pruek Vanna-Iampikul, and Sung Kyu Lim. Fasttuner: Transferable physical design parameter optimization using fast reinforcement learning. In *Proceedings of the 2024 International Symposium on Physical Design*, ISPD '24, page 93–101, New York, NY, USA, 2024. Association for Computing Machinery.

[80] Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training. *arXiv preprint arXiv:2410.10989*, 2024.

[81] Edward J Hu et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[82] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[83] Tsung-Wei Huang, Guannan Guo, Chun-Xun Lin, and Martin DF Wong. Opentimer v2: A new parallel incremental timing analysis engine. *IEEE transactions on computer-aided design of integrated circuits and systems*, 40(4):776–789, 2020.

[84] Binyuan Hui et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[85] Mingjia Huo et al. Token-specific watermarking with enhanced detectability and semantic coherence for large language models. *ICML*, 2024.

[86] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[87] Sheikh Ariful Islam, Love Kumar Sah, and Srinivas Katkoori. High-level synthesis of key-obfuscated rtl ip with design lockout and camouflaging. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(1):1–35, 2020.

[88] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[89] Mojan Javaheripi et al. Litetransformersearch: Training-free neural architecture search for efficient language models. *NeurIPS*, 35:24254–24267, 2022.

[90] Xun Jiang, Zizheng Guo, Zhuomin Chai, Yuxiang Zhao, Yibo Lin, Runsheng Wang, and Ru Huang. Invited paper: Accelerating routability and timing optimization with open-source ai4eda dataset circuitnet and heterogeneous platforms. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[91] Andrew B Kahng et al. Constraint-based watermarking techniques for design ip protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(10):1236–1252, 2001.

[92] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. *VLSI physical design: from graph partitioning to timing closure*, volume 312. Springer, 2011.

[93] Andrew B Kahng, Stefanus Mantik, Igor L Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang, and Gregory Wolfe. Robust ip watermarking methodologies for physical design. In *Proceedings of the 35th annual Design Automation Conference*, pages 782–787, 1998.

[94] Andrew B Kahng and Tom Spyrou. The openroad project: Unleashing hardware innovation. In *Proc. GOMAC*, 2021.

[95] Hadi Mardani Kamali, Kimia Zamiri Azar, Farimah Farahmandi, and Mark Tehranipoor. Advances in logic locking: Past, present, and prospects. *Cryptology ePrint Archive*, 2022.

[96] Katarzyna Kapusta, Vincent Thouvenot, Olivier Bettan, Hugo Beguinet, and Hugo Senet. A protocol for secure verification of watermarks embedded into machine learning models. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, pages 171–176, 2021.

[97] J Phillip Kennedy, Lyndsey Williams, Thomas M Bridges, R Nathan Daniels, David Weaver, and Craig W Lindsley. Application of combinatorial chemistry science on modern drug discovery. *Journal of combinatorial chemistry*, 10(3):345–354, 2008.

[98] Robin Keskisärkkä. Automatic text simplification via synonym replacement, 2012.

[99] Eunji Kim, Bradley S Moore, and Yeo Joon Yoon. Reinvigorating natural product combinatorial biosynthesis with synthetic biology. *Nature chemical biology*, 11(9):649–659, 2015.

[100] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 921–926, 2015.

[101] Young-Won Kim, Kyung-Ae Moon, and Il-Seok Oh. A text watermarking algorithm based on word classification and inter-word space statistics. In *ICDAR*, pages 775–779. Citeseer, 2003.

[102] John Kirchenbauer et al. On the reliability of watermarks for large language models. *arXiv preprint arXiv:2306.04634*, 2023.

[103] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *International Conference on Machine Learning*, pages 17061–17084. PMLR, 2023.

[104] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[105] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[106] Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. Robust distortion-free watermarks for language models. *arXiv preprint arXiv:2307.15593*, 2023.

[107] Thomas Lancaster. Artificial intelligence, text generation tools and chatgpt–does digital watermarking offer a solution? *International Journal for Educational Integrity*, 19(1):10, 2023.

[108] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In *USENIX Security 17*, pages 557–574, 2017.

[109] Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. Who wrote this code? watermarking for code generation. *ACL*, 2024.

[110] Boquan Li, Mengdi Zhang, Peixin Zhang, Jun Sun, and Xingmei Wang. Resilient watermarking for llm-generated codes. *arXiv preprint arXiv:2402.07518*, 2024.

[111] Dingji Li, Zeyu Mi, Yubin Xia, Binyu Zang, Haibo Chen, and Haibing Guan. Twinvisor: Hardware-isolated confidential virtual machines for arm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 638–654, 2021.

[112] Dongfang Li, Wenchao Liu, Xuecheng Zou, and Zhenglin Liu. Hardware ip protection through gate-level obfuscation. In *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, pages 186–193. IEEE, 2015.

[113] Lei Li, Yongfeng Zhang, and Li Chen. Prompt distillation for efficient llm-based recommendation. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pages 1348–1357, 2023.

[114] Linyang Li et al. Watermarking llms with weight quantization. *arXiv preprint arXiv:2310.11237*, 2023.

[115] Shen Li, Liuyi Yao, Jinyang Gao, Lan Zhang, and Yaliang Li. Double-i watermark: Protecting model copyright for llm fine-tuning. *arXiv preprint arXiv:2402.14883*, 2024.

[116] Shuai Li et al. Turning your strength into watermark: Watermarking large language model via knowledge injection. *arXiv preprint arXiv:2311.09535*, 2023.

[117] Yiming Li et al. Untargeted backdoor watermark: Towards harmless and stealthy dataset copyright protection. *NeurIPS*, 35:13238–13250, 2022.

[118] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. *Advances in neural information processing systems*, 30, 2017.

[119] Yujia Li et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[120] Zongjie Li, Chaozheng Wang, Shuai Wang, and Cuiyun Gao. Protecting intellectual property of large language model-based code generation apis via watermarks. In *CCS*, pages 2336–2350, 2023.

[121] Wei Liang, Xingming Sun, Zhihua Xia, Decai Sun, and Jing Long. A chaotic ip watermarking in physical layout level based on fpga. *Radioengineering*, 20(1):118–125, 2011.

[122] Yuqing Liang et al. Watermarking techniques for large language models: A survey. *arXiv preprint arXiv:2409.00089*, 2024.

[123] Peiyu Liao, Siting Liu, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. Dreamplace 4.0: Timing-driven global placement with momentum-based net weighting. *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 939–944, 2022.

[124] Jai-Ming Lin, Chung-Wei Huang, Liang-Chi Zane, Min-Chia Tsai, Che-Li Lin, and Chen-Fa Tsai. Routability-driven global placer target on removing global and local congestion for VLSI designs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2021.

[125] Ji Lin et al. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

[126] Ji Lin et al. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

[127] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[128] Zi Lin et al. Toxicchat: Unveiling hidden challenges of toxicity detection in real-world user-ai conversation, 2023.

[129] Aiwei Liu, Leyi Pan, Xuming Hu, Shu'ang Li, Lijie Wen, Irwin King, and Philip S Yu. A private watermark for large language models. *arXiv preprint arXiv:2307.16230*, 2023.

[130] Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Computing Surveys*, 57(2):1–36, 2024.

[131] Chang Liu and Jun Zhao. Resource allocation for stable llm training in mobile edge computing. In *MobiHoc*, pages 81–90, 2024.

[132] Jiawei Liu et al. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[133] Jinwei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline F. Y. Young. Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[134] Wen-Hao Liu, Stefanus Mantik, Wing-Kai Chow, Yixiao Ding, Amin Farshidi, and Gracieli Posser. ISPD 2019 initial detailed routing contest and benchmark with advanced routing rules. In *Proceedings of the 2019 International Symposium on Physical Design*, pages 147–151, 2019.

[135] Yepeng Liu and Yuheng Bu. Adaptive text watermark for large language models. *ICML*, 2024.

[136] Zechun Liu et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*, 2024.

[137] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*, 2024.

[138] Zhuang Liu et al. Trusted-dnn: A trustzone-based adaptive isolation strategy for deep neural networks. In *Proceedings of the ACM Turing Award Celebration Conference-China*, pages 67–71, 2021.

[139] Daniela Sánchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Souvik Hazra, Robert Wille, and Wolfgang Ecker. A survey of graph neural networks for electronic design automation. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2021.

[140] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[141] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2015.

[142] Kangjie Lu, Wenke Lee, Stefan Nürnberger, and Michael Backes. How to make aslr win the clone wars: Runtime re-randomization. In *NDSS*, 2016.

[143] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

[144] Yi-Chen Lu, Wei-Ting Chan, Vishal Khandelwal, and Sung Kyu Lim. Driving early physical synthesis exploration through end-of-flow total power prediction. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, MLCAD '22, page 97–102, New York, NY, USA, 2022. Association for Computing Machinery.

[145] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[146] Yi-Chen Lu and Sung Kyu Lim. On advancing physical design using graph neural networks. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery.

[147] Yi-Chen Lu, Sai Pentapati, and Sung Kyu Lim. The law of attraction: Affinity-aware placement optimization using graph neural networks. In *Proceedings of the 2021 International Symposium on Physical Design*, ISPD '21, page 7–14, New York, NY, USA, 2021. Association for Computing Machinery.

[148] Yi-Chen Lu, Tian Yang, Sung Kyu Lim, and Haoxing Ren. Placement optimization via ppa-directed graph clustering. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, MLCAD '22, page 1–6, New York, NY, USA, 2022. Association for Computing Machinery.

[149] Yi-Chen Lu, Tian Yang, Sung Kyu Lim, and Haoxing Ren. Placement optimization via ppa-directed graph clustering. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, pages 1–6, 2022.

[150] Nils Lukas, Abdulrahman Diaa, Lucas Fenaux, and Florian Kerschbaum. Leveraging optimization for adaptive attacks on image watermarks. *arXiv preprint arXiv:2309.16952*, 2023.

[151] Xinyin Ma et al. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

[152] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[153] Seapahn Megerian, Milenko Drinic, and Miodrag Potkonjak. Watermarking integer linear programming solutions. In *Proceedings of the 39th annual design automation conference*, pages 8–13, 2002.

[154] David Megıas et al. Architecture of a fake news detection system combining digital watermarking, signal processing, and machine learning. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 13(1):33–55, 2022.

[155] Stephen Merity et al. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[156] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[157] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[158] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[159] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[160] Myungsuk Moon, Minhee Kim, Joonkyo Jung, and Dokyung Song. Asgard: Protecting on-device deep neural networks with virtualization-based trusted execution environments. In *Proceedings 2025 Network and Distributed System Security Symposium*, 2025.

[161] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[162] Dimitris Mouris and Nektarios Georgios Tsoutsos. Zilch: A framework for deploying transparent zero-knowledge proofs. *IEEE Transactions on Information Forensics and Security*, 16:3269–3284, 2021.

[163] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023.

[164] Travis Munyer and Xin Zhong. Deeptextmark: Deep learning based text watermarking for detection of large language model generated text. *arXiv preprint arXiv:2305.05773*, 2023.

[165] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911*, 2022.

[166] Gita Naseri and Mattheos AG Koffas. Application of combinatorial optimization strategies in synthetic biology. *Nature communications*, 11(1):2446, 2020.

[167] Paarth Neekhara, Shehzeen Hussain, Xinqiao Zhang, Ke Huang, Julian McAuley, and Farinaz Koushanfar. Facesigns: semi-fragile neural watermarks for media authentication and countering deepfakes. *arXiv preprint arXiv:2204.01960*, 2022.

[168] Renan Netto, Sheiny Fabre, Tiago Augusto Fontana, Vinicius Livramento, Laércio L Pilla, Laleh Behjat, and José Luís Güntzel. Algorithm selection framework for legalization using deep convolutional neural networks and transfer learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(5):1481–1494, 2021.

[169] Min Ni and Zhiqiang Gao. Watermarking system for ic design ip protection. In *2004 International Conference on Communications, Circuits and Systems (IEEE Cat. No. 04EX914)*, volume 2, pages 1186–1190. IEEE, 2004.

[170] Nicolai Thorer Sivesind. Chatgpt-generated-abstracts, 2023.

[171] Tingyuan Nie, Tomoo Kisaka, and Masahiko Toyonaga. A watermarking system for ip protection by a post layout incremental router. In *Proceedings of the 42nd annual Design Automation Conference*, pages 218–221, 2005.

[172] Kaiwen Ning, Jiachi Chen, Qingyuan Zhong, Tao Zhang, Yanlin Wang, Wei Li, Yu Zhang, Weizhe Zhang, and Zibin Zheng. Mcgmark: An encodable and robust online watermark for llm-generated malicious code. *arXiv preprint arXiv:2408.01354*, 2024.

[173] Institute of Electrical, Electronics Engineers. Computer Society. Standards Committee, and David Stevenson. *IEEE standard for binary floating-point arithmetic*. IEEE, 1985.

[174] R OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023.

[175] OpenAI Team. GPT-4. https://openai.com/research/gpt-4.

[176] Laurel J Orr, Karan Goel, and Christopher Ré. Data management opportunities for foundation models. In *CIDR*, 2022.

[177] Khouloud Oueslati, Maxime Lamothe, and Foutse Khomh. Refagent: A multi-agent llm-based framework for automatic software refactoring. *arXiv preprint arXiv:2511.03153*, 2025.

[178] Min Pan, Natarajan Viswanathan, and Chris Chu. An efficient and effective detailed placement algorithm. In *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, pages 48–55. IEEE, 2005.

[179] Qi Pang, Shengyuan Hu, Wenting Zheng, and Virginia Smith. Attacking llm watermarks by exploiting their strengths. In *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024.

[180] Sebastiano Panichella. Vulnerabilities introduced by llms through code suggestions. In *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*, pages 87–97. Springer Nature Switzerland Cham, 2024.

[181] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[182] Sameer Pawanekar, Gaurav Trivedi, and Kalpesh Kapoor. A nonlinear analytical optimization method for standard cell placement of vlsi circuits. In *2015 28th International Conference on VLSI Design*, pages 423–428. IEEE, 2015.

[183] Oksana Pichugina. Placement problems in chip design: Modeling and optimization. In *2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, pages 465–473. IEEE, 2017.

[184] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)*, 51(6):1–36, 2019.

[185] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

[186] Tong Qiao, Yuyan Ma, Ning Zheng, Hanzhou Wu, Yanli Chen, Ming Xu, and Xiangyang Luo. A novel model watermarking for protecting generative adversarial network. *Computers & Security*, 127:103102, 2023.

[187] Yihang Qiu, Yan Xing, Xin Zheng, Peng Gao, Shuting Cai, and Xiaoming Xiong. Progress of placement optimization for accelerating vlsi physical design. *Electronics*, 12(2):337, 2023.

[188] Gang Qu, Jennifer L Wong, and Miodrag Potkonjak. Optimization-intensive watermarking techniques for decision problems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 33–36, 1999.

[189] Qualcomm. Whitepaper: The future of ai is hybrid. *Qualcomm blog*, 2022.

[190] Colin Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[191] Tahmid Noor Rahman, Nusaiba Khan, and Zarif Ishmam Zaman. Redefining computing: Rise of arm from consumer to cloud for energy efficiency. *arXiv preprint arXiv:2402.02527*, 2024.

[192] Rachel Selina Rajarathnam, Yibo Lin, Yier Jin, and David Z. Pan. Regds: A reverse engineering framework from gdsii to gate-level netlist. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 154–163, 2020.

[193] Marco Rando, Cesare Molinari, Lorenzo Rosasco, and Silvia Villa. An optimal structured zeroth-order algorithm for non-smooth optimization. *Advances in Neural Information Processing Systems*, 36:36738–36767, 2023.

[194] Mahendra Rathor and Girraj Prasad Rathor. Hard-sign: A hardware watermarking scheme using dated handwritten signature. *IEEE Design & Test*, 2023.

[195] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[196] Haoxing Ren and Jiang Hu. *Machine Learning Applications in Electronic Design Automation*. Springer, 2022.

[197] Matthew JB Robshaw. Md2, md4, md5, sha and other hash functions. Technical report, Technical Report TR-101, RSA Laboratories, 1995. version 4.0, 1995.

[198] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[199] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[200] Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10), 2020.

[201] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, volume 1, pages 57–64. IEEE, 2015.

[202] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

[203] Debasri Saha, Parthasarathi Dasgupta, Susmita Sur-Kolay, and Samar Sen-Sarma. A novel scheme for encoding and watermark embedding in vlsi physical design for ip protection. In *2007 International Conference on Computing: Theory and Applications (ICCTA'07)*, pages 111–116. IEEE, 2007.

[204] Debasri Saha and Susmita Sur-Kolay. Fast robust intellectual property protection for vlsi physical design. In *10th International Conference on Information Technology (ICIT 2007)*, pages 1–6. IEEE, 2007.

[205] Shojiro Sakata. Extension of the berlekamp-massey algorithm to n dimensions. *Information and Computation*, 84(2):207–239, 1990.

[206] Pranesh Santikellur, Rajat Subhra Chakraborty, and Swarup Bhunia. Hardware ip protection using register transfer level locking and obfuscation of control and data flow. In *Behavioral Synthesis for Hardware Security*, pages 57–69. Springer, 2021.

[207] Majid Sarrafzadeh and CK Wong. *An introduction to VLSI physical design*. McGraw-Hill Higher Education, 1996.

[208] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[209] Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

[210] John Schulman et al. Chatgpt: Optimizing language models for dialogue. *OpenAI blog*, 2022.

[211] Abhrajit Sengupta, Mohammed Nabeel, Nimisha Limaye, Mohammed Ashraf, and Ozgur Sinanoglu. Truly stripping functionality for logic locking: A fault-based perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4439–4452, 2020.

[212] Masoumeh Shafieinejad et al. On the robustness of backdoor-based watermarking in deep neural networks. In *ACM IH&MMSec workshop*, pages 177–188, 2021.

[213] Mohammed Shayan, Kanad Basu, and Ramesh Karri. Hardware trojans inspired ip watermarks. *IEEE Design & Test*, 36(6):72–79, 2019.

[214] Mohamed Shehab, Elisa Bertino, and Arif Ghafoor. Watermarking relational databases using optimization-based techniques. *IEEE transactions on Knowledge and Data Engineering*, 20(1):116–129, 2007.

[215] Zhuojia Shen, Komail Dharsee, and John Criswell. Randezvous: Making randomization effective on mcus. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 28–41, 2022.

[216] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[217] Nojan Sheybani et al. Zkrownn: Zero knowledge right of ownership for neural networks. In *DAC*, pages 1–6. IEEE, 2023.

[218] Marius Sinclair. Comparison of the performance of modern heuristics for combinatorial optimization on real data. *Computers & Operations Research*, 20(7):687–695, 1993.

[219] David So et al. Searching for efficient transformers for language modeling. *NeurIPS*, 34:6010–6022, 2021.

[220] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. Robust gnn-based representation learning for hls. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[221] James C Spall. Adaptive stochastic approximation by the simultaneous perturbation method. *IEEE transactions on automatic control*, 2000.

[222] Peter Spindler, Ulf Schlichtmann, and Frank M Johannes. Abacus: Fast legalization of standard cell circuits with minimal movement. In *Proceedings of the 2008 international symposium on Physical design*, pages 47–53, 2008.

[223] Guangyu Sun, Zhiqiang Gao, and Yi Xu. A watermarking system for ip protection by buffer insertion technique. In *7th International Symposium on Quality Electronic Design (ISQED'06)*, pages 5–pp. IEEE, 2006.

[224] Mingjie Sun et al. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[225] Zhensu Sun, Xiaoning Du, Fu Song, and Li Li. Codemark: Imperceptible watermarking for code datasets against neural code completion models. In *Proceedings of the 31st ACM joint European software engineering conference and symposium on the foundations of software engineering*, pages 1561–1572, 2023.

[226] Aleksandra Swiercz. Hyper-heuristics and metaheuristics for selected bio-inspired combinatorial optimization problems. *Heuristics and Hyper-Heuristics-Principles and Applications*, 1:3–20, 2017.

[227] Chee Wei Tan, Shangxin Guo, Man Fai Wong, and Ching Nam Hang. Copilot for xcode: Exploring ai-assisted programming by prompting cloud-based large language models. *arXiv preprint arXiv:2307.14349*, 2023.

[228] Huachen Tan, Moming Duan, Duo Liu, Longyi Zhou, et al. Rethinking literary plagiarism in llms through the lens of copyright laws. In *The 16th Asian Conference on Machine Learning (Conference Track)*, 2024.

[229] Ruixiang Tang, Yu-Neng Chuang, and Xia Hu. The science of detecting llm-generated texts. *Communications of the ACM*, 2024.

[230] Xueyan Tang, Lingzhi Shi, Xun Wang, Kyle Charbonnet, Shixiang Tang, and Shixiao Sun. Zero-knowledge proof vulnerability analysis and security auditing. *Cryptology ePrint Archive*, 2024.

[231] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[232] Ashraful Tauhid, Lei Xu, Mostafizur Rahman, and Emmett Tomai. A survey on security analysis of machine learning-oriented hardware and software intellectual property. *High-Confidence Computing*, page 100114, 2023.

[233] Hugging Face Team. Transformers: State-of-the-art natural language processing. In *EMNLP*, 2020.

[234] The QEMU Project Developers. virtio-pmem. https://www.qemu.org/docs/master/system/devices/virtio-pmem.html, 2024.

[235] Torch Contributors. PyTorch. https://pytorch.org/, 2023. Last Access on December 26, 2022.

[236] torchao. Torchao: Pytorch-native training-to-serving model optimization, oct 2024.

[237] Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[238] Kun Wang, Kaiyan Chang, Mengdi Wang, Xingqi Zou, Haobo Xu, Yinhe Han, and Ying Wang. Rtlmarker: Protecting llm-generated rtl copyright via a hardware watermarking framework. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pages 808–813, 2025.

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

[240] Jiang Wei and Xiaolei Kong. A framework to analyze different intellectual property systems. In *2010 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 178–182. IEEE, 2010.

[241] Xiuying Wei et al. Outlier suppression: Pushing the limit of low-bit transformer language models. *NeurIPS*, 35:17402–17414, 2022.

[242] Thomas Wolf et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[243] Greg Wolfe, Jennifer L Wong, and Miodrag Potkonjak. Watermarking graph partitioning solutions. In *Proceedings of the 38th annual Design Automation Conference*, pages 486–489, 2001.

[244] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 2014.

[245] Lijun Wu, Jinhua Zhu, et al. Machine translation with weakly paired documents. In *EMNLP*, pages 4375–4384, 2019.

[246] Zhuofeng Wu, He Bai, Aonan Zhang, Jiatao Gu, VG Vydiswaran, Navdeep Jaitly, and Yizhe Zhang. Divide-or-conquer? which part should you distill your llm? *arXiv preprint arXiv:2402.15000*, 2024.

[247] Guangxuan Xiao et al. Smoothquant: Accurate and efficient post-training quantization for large language models. In *ICML*, pages 38087–38099, 2023.

[248] Zhibo Xing, Zijian Zhang, Jiamou Liu, Ziang Zhang, Meng Li, Liehuang Zhu, and Giovanni Russello. Zero-knowledge proof meets machine learning in verifiability: A survey. *arXiv preprint arXiv:2310.14848*, 2023.

[249] Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*, 2024.

[250] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[251] Runxin Xu et al. From dense to sparse: Contrastive pruning for better pre-trained language model compression. In *AAAI*, volume 36, pages 11547–11555, 2022.

[252] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[253] Borui Yang, Wei Li, Liyao Xiang, and Bo Li. Towards code watermarking with dual-channel transformations. *arXiv preprint arXiv:2309.00860*, 2023.

[254] Borui Yang, Wei Li, Liyao Xiang, and Bo Li. Srcmarker: Dual-channel source code watermarking via scalable code transformations. In *IEEE Symposium on Security and Privacy*, 2024.

[255] Haoyu Yang, Kit Fung, Yuxuan Zhao, Yibo Lin, and Bei Yu. Mixed-cell-height legalization on cpu-gpu heterogeneous systems. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 784–789. IEEE, 2022.

[256] Xi Yang, Kejiang Chen, Weiming Zhang, Chang Liu, Yuang Qi, Jie Zhang, Han Fang, and Nenghai Yu. Watermarking text generated by black-box language models. *arXiv preprint arXiv:2305.08883*, 2023.

[257] Zhewei Yao et al. A comprehensive study on post-training quantization for large language models. *arXiv preprint arXiv:2303.08302*, 2023.

[258] Y. Ye. The gset dataset. https://web.stanford.edu/ yyye/ yyye/Gset/ (Stanford, 2003).

[259] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. Edgemoe: Fast on-device inference of moe-based large language models. *arXiv preprint arXiv:2308.14352*, 2023.

[260] KiYoon Yoo, Wonhyuk Ahn, Jiho Jang, and Nojun Kwak. Robust multi-bit natural language watermarking through invariant features. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2092–2115, 2023.

[261] Shuguang Yuan, Xingyu Su, Peizhuo Lv, Weiji Xue, Jing Yu, Xiaojie Zhu, and Chi Chen. An efficient white-box llm watermarking for ip protection on online market platforms. In *SIGKDD*, pages 3680–3691, 2025.

[262] Zcash. Halo2 proof system. https://github.com/zcash/halo2, 2024.

[263] Chen Zhang, Kuntai Du, Shu Liu, Woosuk Kwon, Xiangxi Mo, Yufeng Wang, Xiaoxuan Liu, Kaichao You, Zhuohan Li, Mingsheng Long, et al. Jenga: Effective memory management for serving llm with heterogeneity. *arXiv preprint arXiv:2503.18292*, 2025.

[264] Jie Zhang et al. Deep model intellectual property protection via deep watermarking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4005–4020, 2021.

[265] Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar. {REMARK-LLM}: A robust and efficient watermarking framework for generative large language models. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1813–1830, 2024.

[266] Ruisi Zhang and Farinaz Koushanfar. Emmark: Robust watermarks for ip protection of embedded quantized large language models. *arXiv preprint arXiv:2402.17938*, 2024.

[267] Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar. Automated physical design watermarking leveraging graph neural networks. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, pages 1–10, 2024.

[268] Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar. Icmarks: A robust watermarking framework for integrated circuit physical design ip protection. *arXiv preprint arXiv:2404.18407*, 2024.

[269] Susan Zhang et al. Opt: Open pre-trained transformer language models, 2022.

[270] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

[271] Zaixi Zhang, Ruofan Jin, Kaidi Fu, Le Cong, Marinka Zitnik, and Mengdi Wang. Foldmark: Protecting protein generative models with watermarking. *bioRxiv*, pages 2024–10, 2024.

[272] Ziqi Zhang et al. No privacy left outside: On the (in-) security of tee-shielded dnn partition for on-device ml. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3327–3345. IEEE, 2024.

[273] Xuandong Zhao, Prabhanjan Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for ai-generated text. *arXiv preprint arXiv:2306.17439*, 2023.

[274] Xuandong Zhao, Yu-Xiang Wang, and Lei Li. Protecting language generation models via invisible watermarking. *arXiv preprint arXiv:2302.03162*, 2023.

[275] Xinyao Zheng, Husheng Han, Shangyi Shi, Qiyan Fang, Zidong Du, Xing Hu, and Qi Guo. Inputsnatch: Stealing input in llm services via timing side-channel attacks. *arXiv preprint arXiv:2411.18191*, 2024.

[276] Jie Zhou et al. Graph neural networks: A review of methods and applications. *AI open*, 2020.

[277] Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

[278] Zkonduit. Ezkl. https://github.com/zkonduit/ezkl, 2024.