AutoMarks: A GNN-based Automated Physical Design Watermarking Framework

RUISI ZHANG, Electrical and Computer Engineering, University of California San Diego, La Jolla, United States
RACHEL SELINA RAJARATHNAM, Electrical and Computer Engineering, The University of Texas at Austin, Austin, United States
DAVID PAN, Electrical and Computer Engineering, The University of Texas at Austin, Austin, United States
FARINAZ KOUSHANFAR, Electrical and Computer Engineering, University of California San Diego, La Jolla, United States

Physical design refers to converting a logical circuit description into a physical chip layout for manufacturing. Although design companies invest huge efforts to fine-tune the cell positions and connections for better power, performance, and area metrics, chip layouts are vulnerable to security risks along the supply chain. Physical design watermarking injects signatures on the chip layout, allowing ownership authentication and avoiding potential security risks. A substantial effort is required to identify the best locations to insert the watermarks on the chip layout without affecting the overall layout quality. This paper presents AutoMarks, an automated and transferable watermarking framework that leverages graph neural networks to reduce watermark search overhead during the integrated circuit design's placement stage. AutoMarks's novel automated watermark search is accomplished by (i) constructing novel graph and node features with physical, semantic, and design constraint-aware representation; (ii) designing a data-efficient sampling strategy for watermarking fidelity label collection; and (iii) leveraging a graph neural network to learn the connectivity between cells and predict the watermarking fidelity on unseen layouts. Extensive evaluations on ISPD'15, ISPD'19, and ICCAD'15 benchmarks demonstrate that our proposed automated methodology: (i) is capable of finding quality-preserving watermarks in a short time; (ii) is transferable across various designs, i.e., AutoMarks trained on one layout is generalizable to other benchmark circuits; (iii) is effective in watermarking designs optimized for wirelength and timing metrics. AutoMarks is also resilient against potential watermark removal and forging attacks. Our code is publicly available on GitHub [1].

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Hardware** → **Very large scale integration design**; • **Security and privacy** → **Security in hardware**.

Additional Key Words and Phrases: Physical Design, Watermarking, Graph Neural Networks

1 Introduction

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 [18, 46] transforms the design's logic netlist to a physical layout for fabrication. The placement [25, 35] and

[1] https://github.com/ruisizhang123/PD_WM_GNN

Authors' Contact Information: Ruisi Zhang, Electrical and Computer Engineering, University of California San Diego, La Jolla, California, United States; e-mail: ruz032@ucsd.edu; Rachel Selina Rajarathnam, Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas, United States; e-mail: rachelselina.r@utexas.edu; David Pan, Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas, United States; e-mail: dpan@ece.utexas.edu; Farinaz Koushanfar, Electrical and Computer Engineering, University of California San Diego, La Jolla, California, United States; e-mail: farinaz@ucsd.edu.

ACM 1557-7309/2025/9-ART
https://doi.org/10.1145/3768343

routing [11, 24] 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 [42], such as intellectual property (IP) theft, counterfeiting, and unauthorized overproduction. Watermarking [17, 48, 51, 52] 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 [52]: (i) invasive watermarking and (ii) constraint-based watermarking. Invasive watermarking [6, 38, 43, 48] 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 [8, 17, 52], 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. More recent work introduced ICMarks [51, 52] to encode a watermark region of cells onto modern IC design without impacting the overall quality metrics. It aims to constrain watermark cells to be in the watermark region and the remaining cells to be out. Then, the selected region of cells is encoded as a co-optimization term during the global placement, in which selected cell locations within the region are considered as the watermark. However, searching for the ideal region constraints that adhere to design constraints and maintain watermarking fidelity consumes significant time.

This paper 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 [9, 15], gate level [21], and macro level [49]. 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 [52]. The placement is then routed to obtain the watermarked layout. Finally, *watermark extraction* employs reverse-engineering approaches [4, 39] 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, our contributions are summarized as follows:

- 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 [7], ISPD'19 [27], and ICCAD'15 [20] 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[52].
- Evaluations of AutoMarks under various watermark removal and forging attacks showcase AutoMarks's resiliency; the adversary cannot remove the signatures without significant quality degradations.

Paper Organization: Section 2 includes the background and related work, and Section 3 introduces the problem formulation. Section 4 presents AutoMarks's methodology, and the experiments demonstrating AutoMarks's effectiveness are in Section 5. Section. 6 provides a summary of the work.

2 Background and Related Work

This section provides a background on graph neural networks (GNNs) in Section 2.1 along with the different techniques used during watermarking in physical design in Section 2.2.

2.1 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 [55] is to learn a function g that maps the feature embedding $\mathbf{h}_i$ of node v_i to a new embedding vector $\hat{\mathbf{h}_i} \in R^h$, 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 $\hat{\mathbf{h}_i}$. The computations are formulated in Equation 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, $\mathcal{N}(v)$ is the set of neighbors for node v, and $\mathbf{h}_v^{(l+1)}$ is the updated feature of node v at layer $(l+1)$.

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

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 [14, 28–31, 41, 47]. During the placement stage of physical design, GNNs can optimize the macro locations, overall timing, and power [16, 32, 33, 35]. In addition, GNNs can ensure hardware reliability, security, and reverse engineering of gate-level netlists during the physical design stage [3, 5].

2.2 Physical Design Watermarking

Watermarking is a prevalent technique in various domains, including image [36, 54], text [2, 13], and audio processing [45, 53], allowing creators to assert their intellectual property (IP) rights without compromising product quality. Physical design watermarking encodes imperceptible and unique identifiers onto the physical design layouts to protect the design company's intellectual property (IP). Invasive watermarking schemes add redundant cells or nets onto the layout as watermarks [6, 38, 43, 48], but can be easily counterfeited if the attacker

has prior knowledge of the watermarking scheme. Constraint-based watermarking schemes [8, 17, 19, 37] modify the cell row index and cell spacing as watermarks during the placement stage. For example, row parity [17, 19] enforces 1-bit watermark cells to be on the odd row and the 0-bit watermark cells to be on the even row. Cell scattering [8] constraints the 1-bit watermark cell to move one unit along the y-axis and 0-bit cell one unit along the x-axis. While those approaches enable successful watermark insertion, the watermarked layout quality can be significantly degraded if the watermarks are selected without considering modern design constraints such as macros and fence regions. Invasive watermarking approaches, like buffer insertion [38, 48], add redundant cells or wires as the watermark signatures.

The recent constraint-based watermarking work, ICMarks [52], employs a scoring function to evaluate each subregion to ensure (i) sufficient cells to accommodate the signature bits; (ii) small total cell area within the watermark region for cell maneuverability; and (iii) minimal cell area overlap on the region boundary to reduce the impact of cell displacement on layout performance. The novel scoring scheme identifies a region to watermark that does not violate the design constraints and ensures minimal performance deterioration. The selected best-scored region and cells within the region are used for watermark encoding. Then, the watermark insertion encodes the region of cells during the global placement, which co-optimizes the placement objective to ensure that only the watermark cells are in the watermark region. Finally, the detailed placement selects cells that do not overlap with nearby cells when moving along the x/y axis within global watermarking's watermarked region. The selected cells are watermarked by shifting in the x/y directions to encode detailed watermarking signatures in the detailed placement.

3 Motivation and Problem Formulation

Section 3.1 outlines the use of AutoMarks scheme during physical design as a proof of IC ownership. Section 3.2 describes the characteristics of an effective watermarking method, and Section 3.3 specifies the potential attacks to thwart the effectiveness of the watermark.

3.1 Scenario

The IC design company establishes ownership proof for the final physical design layout by encoding watermarks during the placement stage using AutoMarks. The watermarked layout is sent along the supply chain for manufacturing and testing. To prove ownership, the design company reverse-engineers the cell locations from the layout [4, 39] and compares the encoded watermarks with the decoded ones as depicted in Fig. 1.

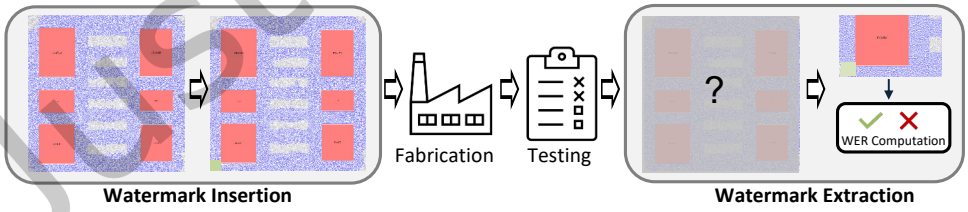


Fig. 1. After watermark insertion, the watermarked placement is routed to obtain the watermarked layout. The layout is sent for fabrication and testing. If the design company suspects a layout is watermarked, they reverse-engineer the layout to acquire the cell locations and compare them to the watermarked region of cells for $\%WER$ to claim ownership.

3.2 Watermarking Criteria

An ideal watermarking framework shall meet the following criteria:

- *Fidelity*: The watermarked layout does not impact the functionality and performance of the IC. As such, the watermarked products can be used for downstream manufacturing.
- *Effectiveness*: The encoded watermarks can be successfully extracted along the supply chain for the design company to claim their ownership.
- *Stealthiness*: the watermarks are undetectable within the layout, preventing them from being identified or targeted for removal attacks.
- *Robustness*: The watermarks can withstand various removal and forging attacks from the adversary and remain detectable after transformations.

3.3 Threats

We consider the adversary to be a member of the supply chain, e.g., a fabrication or testing company, with access to the layout and general knowledge of watermarking algorithms. However, he/she cannot access the specific watermarking signatures or hyperparameters used by the design company. Potential threats to the watermarked layout include:

- *Watermark Removal Attacks*: The adversary removes the watermark by perturbing the placement solutions while ensuring the layout quality is not compromised. Potential transformations include slightly perturbing cell positions, applying another round of detailed placement, etc.
- *Watermark Forging Attacks*: The adversary forges the watermark by counterfeiting a different set of watermarks. As such, they claim the watermarked layout belongs to him/her instead of the design company.

4 AutoMarks Methodology

This section introduces AutoMarks and the stages in its pipeline: *Watermark Search*, *Watermark Insertion*, and

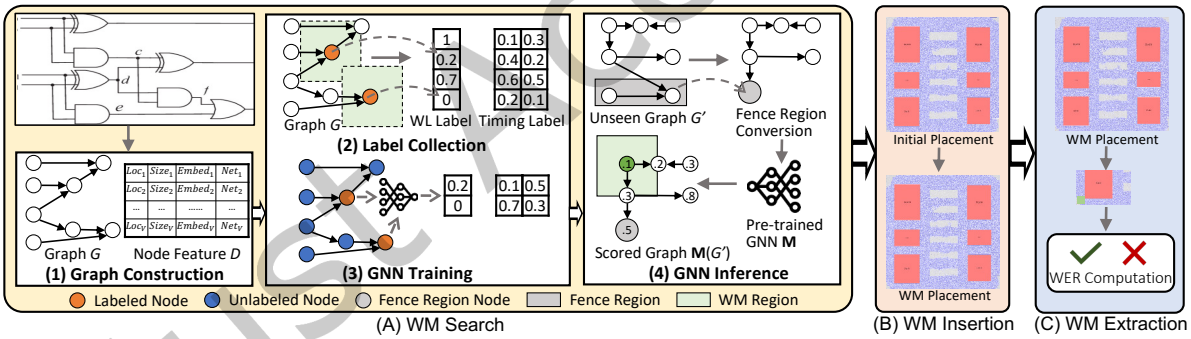


Fig. 2. AutoMarks flow: The *Watermark Search* leverages GNN to identify the region and cells to watermark. For the wirelength-driven benchmarks, the label and GNN prediction result is the wirelength improvement rate over the non-watermarked placement. For the timing-driven benchmarks, the labels and 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.

Watermark Extraction, as depicted in Fig. 2. *Watermark Search* identifies potential regions and cells to encode the signature on layouts that optimize for wirelength in Section 4.1, and timing metrics in Section 4.2. *Watermark Insertion* encodes watermarks onto the IC layout without degrading the quality metrics as detailed in Section 4.3. Finally, *Watermark Extraction* decodes the watermark signature to verify ownership as described in Section 4.4.

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 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.

4.1.1 ***Graph Construction.*** The design instances, e.g., standard cells and macros, are connected to others by wired nets through the cell pins. As in Fig. 2(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 1. 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 [34] from S-BERT [40] and use Principal Component Analysis (PCA) [1] to reduce them to four dimensions to form the last four features of D.

Type	Dim	Description
Cell Location	2	from initial placement (x, y)
Cell Size	2	cell size (width, height)
Cell Name	4	S-BERT semantics [40]

Table 1. Node Feature Construction for Wirelength-driven Watermark Search.

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 distinguished from the standard cells by different cell name embeddings.
- *Fence Regions*: As depicted in Fig. 2(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.

4.1.2 ***Label Collection.*** We collect the watermarking quality degradation labels on the `ispd19test6` [27] design. We chose this design as the training graph for two reasons: (i) the design is of medium size with ~ 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 2, if $\hat{L}_i < 1$, it indicates watermarking on the node will not introduce any performance degradation. Such nodes are scored as 0,

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

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, 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 $\beta = 1.01$.

$$L_i = \begin{cases} 0 & \text{if } \hat{L}_i < 1 \\ \frac{\hat{L}_i - 1}{\beta} & \text{if } 1 \leq \hat{L}_i \leq \beta \\ 1 & \text{if } \hat{L}_i > \beta \end{cases} \tag{2}$$

4.1.3 ***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 [22]. As in Equation 3, 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 $(l + 1)$.

$$\mathbf{h}_i^{(l+1)} = \text{ReLU}\left(\sum j \in \mathcal{N}(i) \frac{1}{c_{ij}} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right) \tag{3}$$

$\mathbf{M}_{wl}$ is trained to minimize the MSE loss between the predicted label $\mathbf{M}_{wl}(G)$ and the ground truth label L as in Equation 4.

$$\mathcal{L}_{\text{wl}} = \frac{1}{|L|} \sum_{i=1}^{|L|} (\mathbf{M}_{wl}(G_i) - L_i)^2 \tag{4}$$

4.1.4 ***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 Fig. 2(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 5. 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 $c_{wc} = argmin(L' + \gamma * L'_{agg})$.

$$L'_{agg,i} = \frac{1}{N} \sum_{k=1}^{N} \left(\frac{1}{|\mathcal{N}_k(i)|} \sum_{j \in \mathcal{N}_k(i)} L'_j \right) \tag{5}$$

4.2 *Timing-driven Watermark Search*

The timing-driven watermark search is similar to the wirelength-driven search in Section 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.

4.2.1 ***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 2. It models the slack of each cell and helps the graph neural network predict the total negative slack and worst negative slack.

Type	Dim	Description
Cell Location	2	from initial placement (x, y)
Cell Size	2	cell size (width, height)
Cell Name	4	S-BERT semantics [40]
Net weight	1	average of net weights connected to the cell

Table 2. Node Feature Construction for Timing-driven Watermark Search.

4.2.2 ***Label Collection.*** We collect the timing metric labels on `superblue5` [20] 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) [18]. 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.

4.2.3 ***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 2 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 6.

$$\mathcal{L}_{\text{timing}} = \frac{1}{|L|}\sum_{i=1}^{|L|}((\mathbf{M}_m(\mathbf{M}_{tns}(G_i)) - L_{tnsi})^2 + (\mathbf{M}_m(\mathbf{M}_{wns}(G_i)) - L_{wnsi})^2) \tag{6}$$

4.2.4 ***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.

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 [12]. 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.

$$\begin{aligned} \min_v \quad & \textstyle\sum_{e\in E} W(e; v) + \lambda D(v), \\ \text{s.t.} \quad & v_k = (x_k, y_k) \in R_k, \quad k = 1, \cdots, R_K, R_w, \end{aligned} \tag{7}$$

As in Fig 1, the watermarked placement is then routed, and the design company obtains the watermarked layout. The watermarked layout is sent for fabrication and testing with signatures rooted in the integrated circuit.

4.4 *Watermark Extraction*

To prove ownership, the design company reverse-engineers the suspect layout [4, 39] 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 8. The owners can use the percentage of the matched watermark as $\%WER$ to claim their ownership of the integrated circuit.

$$\%WER = 100 \times \frac{|C'_w|}{|C_w|} \tag{8}$$

5 Experiments

This section introduces the experiment setups in Section 5.1. The watermarking results are presented in Section 5.2 and the robustness evaluations are presented in Section 5.3. Finally, we conduct an ablation study in Section 5.4 and provide AutoMarks's visualizations in Section 5.5.

5.1 Experiment Setups

5.1.1 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 [12] as the placement algorithm and CUGR [26] 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 3, 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 4. 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 5.

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 3. GNN training hyperparameters.

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 4. GNN architecture $\mathbf{M}_{wl}$ for Wirelength-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

Table 5. GNN architecture $\mathbf{M}_{timing}$ for Timing-driven Watermark Search.

5.1.2 Benchmarks. We evaluate AutoMarks's watermarking performance on the wirelength-driven ISPD'2015 [7], and ISPD'2019 [27] benchmark suites; and the timing-driven ICCAD'2015 [20] benchmark suite. Designs with fence region constraints are highlighted in blue in Tables 6 and 7. ICCAD'2015's [20] layouts in Table 8 do not have fence regions.

5.1.3 Baseline. We compare AutoMarks with state-of-the-art frameworks employing both invasive and constraint-based watermarking:

- **Row Parity** [17, 19] 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.
- **Cell Scattering** [8] scatters watermark cells on the chip canvas as watermarks using pseudorandom coordinate transformation (PRCT) algorithms. 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.
- **ICMarks** [52] employs a scoring function to evaluate each layout subregion and select the best region of cells that adheres to design constraints and has minimal impact on the layout as the watermark region. The watermark region enforces only the watermark cells to be in the watermark region
- **Buffer Insertion** [48] adds additional buffers as watermarks during the placement stage. Two buffers are inserted to represent 0-bit, and one buffer is inserted to represent 1-bit.

We skip the baselines that: (i) have different watermarking targets, e.g., smaller full-custom IC designs [6, 37] and FPGAs [23, 44]. In FPGAs, signatures are typically encoded by configuring lookup tables (LUTs) as specific functions; and in small full-custom IC designs, they are embedded by adjusting transistors to match watermark values. These watermarking approaches are fundamentally different and orthogonal to AutoMarks's watermark insertion for IC layouts.; (ii) have similar watermarking approaches as our baselines, and we use the baselines as a proof-of-concept to demonstrate the performance. For example, flip-flops are encoded as watermarks in [43] instead of buffers, as in [48]. For this reason, we do not include it in this paper.

5.1.4 Evaluation Metrics. For wirelength-driven benchmark suites, we employ the following metrics to measure the watermarked layout quality:

- **Placement WireLength Rate (PWLR)**: The rate of estimated half-perimeter wirelength (HPWL) of watermarked layout compared to the original one.
- **Routing WireLength Rate (RWLR)**: The rate of routed wirelength of watermarked layout compared to the original one.

For timing-driven benchmark suite, we employ the following metrics measure the watermarked layout quality:

- **Total Negative Slack Rate (TNSR)**: The rate of total negative slack (TNS) of watermarked layout compared to original one.
- **Worst Negative Slack Rate (WNSR)**: The rate of worst negative slack (WNS) of watermarked layout compared to original one.

Besides, we measure the percentage of watermark cells successfully extracted as Watermark Extraction Rate (WER).

5.2 AutoMarks Results

5.2.1 ***Wirelength-driven Watermarking Fidelity and Transferability.*** We compare the watermarking performance of AutoMarks and the baselines on the ISPD'2015 [7] benchmarks in Table 6, and the ISPD'2019 [27] benchmarks in Table 7. 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.

Comparison with Constraint-based WM: Compared to AutoMarks that maintains watermarking fidelity, the baseline Row Parity [17, 19] and Cell Scattering [8] 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 [7] and ISPD'2019 [27] benchmarks respectively. The Row Parity [17, 19] and Cell Scattering [8] approaches do not consider the design constraints, like fence

Design	Cells	Nets	Row Parity [17, 19]		Cell Scattering [8]		Buffer Insertion [48]		ICMarks [52]		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 6. Performance on the ISPD'2015 benchmarks [7]. 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 ⋆).

Design	Cells	Nets	Row Parity [17, 19]		Cell Scattering [8]		Buffer Insertion [48]		ICMarks [52]		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**

Table 7. Performance on the ISPD'2019 benchmarks [27]. 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.

regions or macros when selecting the watermark cells, resulting in worse performance. Compared to AutoMarks, ICMarks [52] introduced slightly more degradations on ISPD'2019 [27] 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 [52].

Comparison with Invasive WM: Compared to AutoMarks, the invasive watermarking Buffer Insertion [48] significantly degrades the PWLR and RWLR by 5.36% and 9.01% on the ISPD'2015 [7] designs; and by 3.04% and 8.71% on the ISPD'2019 [27] 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.

5.2.2 ***Timing-driven Watermarking Fidelity and Transferability.*** The timing-driven watermarking performance on ICCAD'15 [20] benchmarks is in Table 8. The encoded watermarks for all methodologies are extracted successfully, $\%WER = 100$ for all designs. Similar to Section 5.2.1, 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 [17, 19] and invasive watermarking Buffer Insertion [48] 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 3.

Design	Cells	Nets	Row Parity [17, 19]		Cell Scattering [8]		Buffer Insertion [48]		ICMarks [52]		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

Table 8. Performance on ICCAD'2015 benchmarks [20]. 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.

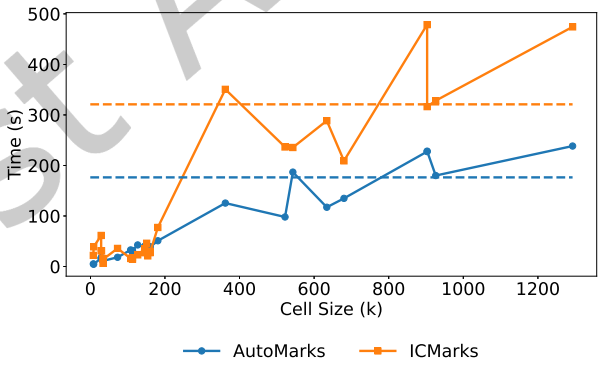


Fig. 3. The wirelength-driven watermark search time for different designs in ISPD'2015 [7] and ISPD'2019 [27]. The dotted line is the average search time of the large designs ($\geq 500k$ cells).

5.2.3 ***Efficiency.*** Fig. 3 and Table 9 compare the watermark search time for ICMarks [52] and AutoMarks. For smaller designs in wirelength-driven watermark search, the search time of AutoMarks is very close to ICMarks [52]. However, for the large designs ($\geq 500k$ cells) in Fig. 3, the average search time of AutoMarks is

176.38*s*, whereas ICMarks requires 320.93*s*. 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 9, ICMarks [52] takes an average of 72.37*s* to find an ideal position to encode watermarks, whereas the average time of AutoMarks is 53.99*s*. 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%.

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 ≤ 70%)	98.05	57.34	-

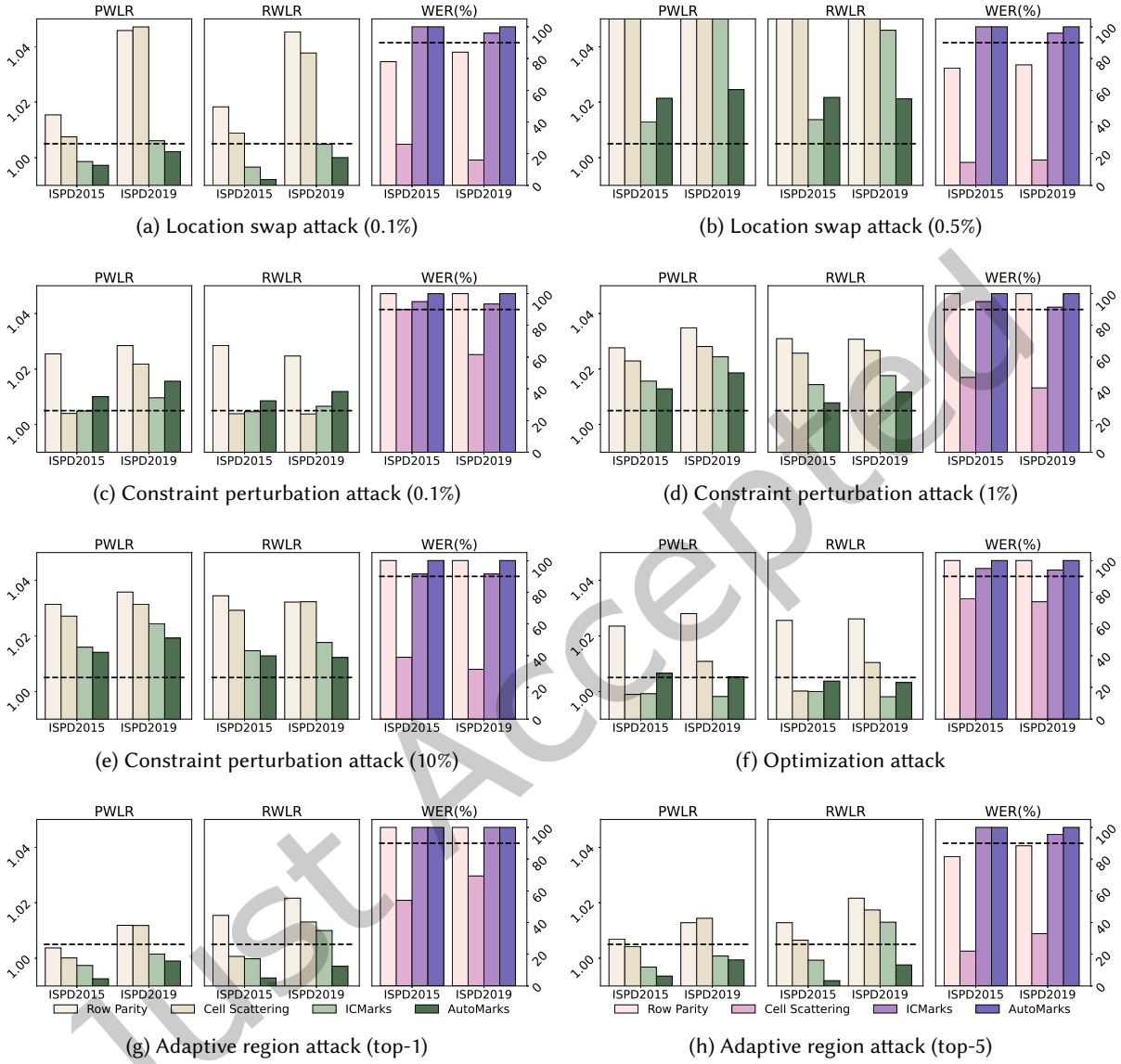
Table 9. Watermark search runtime in ICCAD'2015 [20].

5.3 AutoMarks's Robustness

5.3.1 ***Watermark Removal Attacks on Benchmarks Optimizing Wirelength.***

Fig. 4 evaluates the impact of watermark removal attacks on AutoMarks and the baseline approaches. The **Location swap attack** targeting Row Parity [17, 19] selects 0.1% of the total cells randomly and swaps their locations. The **Constraint perturbation attack** targeting Cell Scattering [8] 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 [52] and AutoMarks, assumes the adversary knows the size of the watermark region and uses the evaluation function of ICMarks [52] 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.

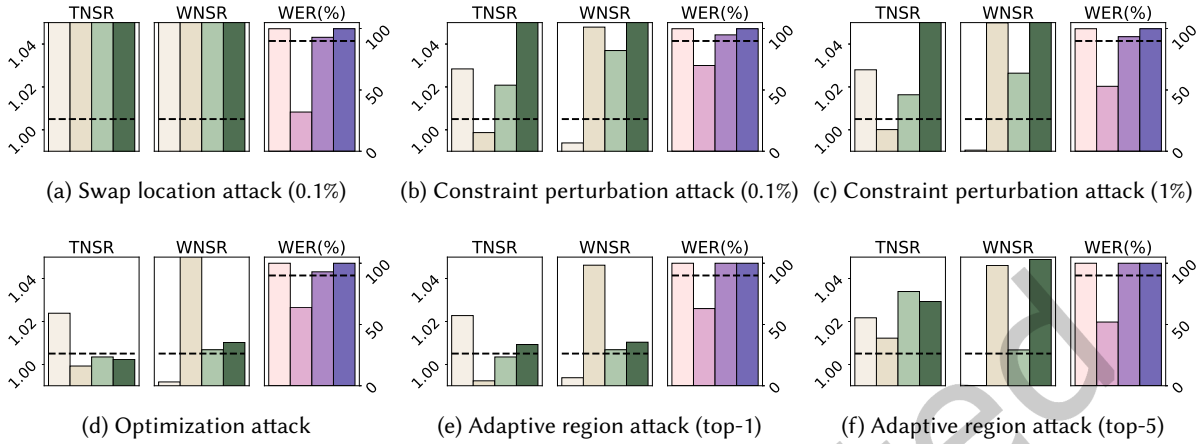
AutoMarks is resilient to all attacks and maintains the watermark extraction rates of 100% even when the quality metrics PWLR and RWLR are greatly compromised; because as long as the watermark cells are within the watermark region, it is more robust toward such slight perturbation of cell locations. 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



(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)

Fig. 4. Watermarking performance under different attacks for wirelength-driven placement on the ISPD'2015 [7] and ISPD'2019 [27] 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%.

watermarking strength, the detailed watermarking in ICMarks is vulnerable to watermark removal attacks. As a result, the overall WER for ICMarks slightly degrades.



(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)

Fig. 5. Watermarking performance under different attacks for timing-driven placement on the ICCAD'2015 benchmarks [20]. 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%.

5.3.2 ***Watermark Removal Attacks on Benchmarks Optimizing Timing Metrics.*** The watermarked layout performance after attacks for different methodologies are in Figure 5. 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 detailed in Section 5.3.1. 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 [52] 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 is greatly compromised. Post-routing ECO [10, 50] 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 5d, 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.

5.3.3 ***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.

5.4 Ablation Study

This subsection provides ablation studies over the different hyperparameter choices in AutoMarks's performance on the ISPD'2019 [27] benchmarks as a proof-of-concept.

The Effectiveness of Node Feature: We analyze how nonnative node features impact the GNN performance. In Table 10, 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 10. Following Sec. 5.2, 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 10, 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 10. 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 10. Here, we fix the watermark region size $N = 10$ to be the same as Section 5.2 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.

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

Table 10. The effectiveness of different node feature constructions, learning rate choices, β and γ choices, and layer numbers on AutoMarks's performance.

5.5 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 6. 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 Fig. 7, in which the model is trained on `ispd19test6` [27] design and the loss in evaluated using Equation 4. It demonstrates the graph neural network gradually learns to predict the quality degradations of a given subgraph. The loss converges during the GNN training.

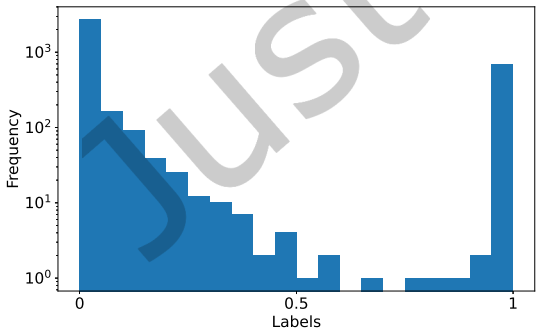


Fig. 6. Histogram distribution of AutoMarks's labels.

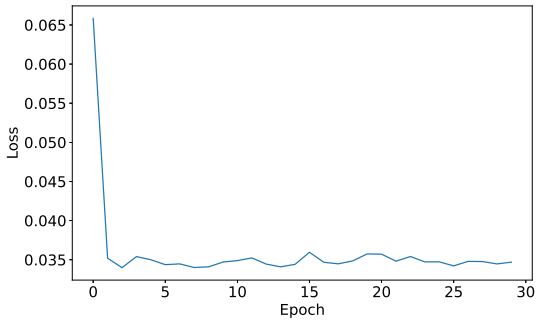
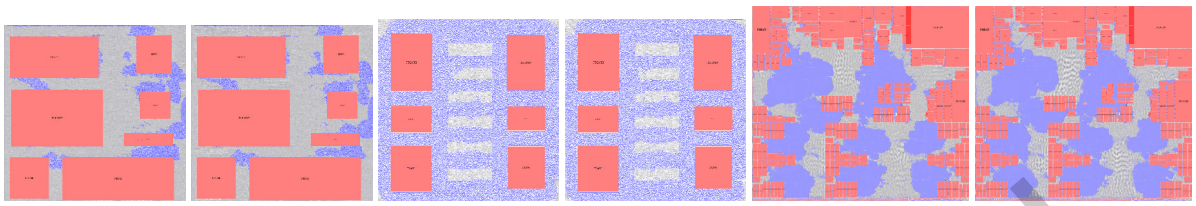


Fig. 7. Loss curve during the AutoMarks's GNN training.

Stealthiness: To verify the stealthiness of AutoMarks, we include the watermarked layout and the non-watermarked layout from both ISPD'2015 [7], ISPD'2019 [27], and ICCAD'2015 [20] benchmarks in Fig. 8. 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)

Fig. 8. Watermarked and Non-watermarked examples.

6 Conclusion and Future Work

This work 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.

In future work, we plan to extend AutoMarks along the following directions. First, we will incorporate additional design metrics from later stages of the workflow (e.g., post–clock-tree synthesis and post-routing) to further improve watermarking performance. Second, we will investigate the transferability of the GNN-based watermark region identification strategy to other hardware platforms, such as FPGAs and full-custom IC designs, where signatures can be embedded in lookup tables or at the transistor level.

Acknowledgments

This work was supported by NSF TILOS AI Institute award number 2112665.

References

[1] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[2] Milad Taleby Ahvanooey, Qianmu Li, Xuefang Zhu, Mamoun Alazab, and Jing Zhang. 2020. ANiTW: A novel intelligent text watermarking technique for forensic identification of spurious information on social media. *Computers & Security* 90 (2020), 101702.

[3] Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu. 2023. 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)*. Association for Computing Machinery, New York, NY, USA, 83–90. doi:10.1145/3566097.3568345

[4] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2021. 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 (2021), 2435–2448.

[5] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2022. 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 (2022), 2435–2448. doi:10.1109/TCAD.2021.3110807

[6] Fujun Bai, Zhiqiang Gao, Yi Xu, and Xueyu Cai. 2007. A watermarking technique for hard IP protection in full-custom IC design. In *2007 International Conference on Communications, Circuits and Systems*. IEEE, 1177–1180.

[7] Ismail S Bustany, David Chinnery, Joseph R Shinnerl, and Vladimir Yutsis. 2015. 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*. 157–164.

[8] Xueyu Cai, Zhiqiang Gao, Fujun Bai, and Yi Xu. 2007. A watermarking technique for hard IP protection in post-layout design level. In *2007 7th International Conference on ASIC*. 1317–1320. doi:10.1109/ICASIC.2007.4415879

[9] Rajat Subhra Chakraborty and Swarup Bhunia. 2010. RTL hardware IP protection using key-based control and data flow obfuscation. In *2010 23rd International Conference on VLSI Design*. IEEE, 405–410.

[10] Hsi-An Chien and Ting-Chi Wang. 2014. Redundant-via-aware ECO routing. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 418–423.

[11] Upma Gandhi, Ismail Bustany, William Swartz, and Laleh Behjat. 2019. 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)*. IEEE, 1–6.

[12] Jiaqi Gu, Zixuan Jiang, Yibo Lin, and David Z Pan. 2020. DREAMPlace 3.0: Multi-electrostatics based robust VLSI placement with region constraints. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.

[13] Anwer Mustafa Hilal, Fahd N Al-Wesabi, Abdelzahir Abdelmaboud, Manar Ahmed Hamza, Mohammad Mahzari, and Abdulkhaleq QA Hassan. 2022. A hybrid intelligent text watermarking and natural language processing approach for transferring and receiving an authentic english text via internet. *Comput. J.* 65, 2 (2022), 423–435.

[14] Hao-Hsiang Hsiao, Yi-Chen Lu, Pruek Vanna-Iampikul, and Sung Kyu Lim. 2024. FastTuner: Transferable Physical Design Parameter Optimization using Fast Reinforcement Learning. In *Proceedings of the 2024 International Symposium on Physical Design (ISPD '24)*. Association for Computing Machinery, New York, NY, USA, 93–101. doi:10.1145/3626184.3633328

[15] Sheikh Ariful Islam, Love Kumar Sah, and Srinivas Katkoori. 2020. High-level synthesis of key-obfuscated RTL IP with design lockout and camouflaging. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 1 (2020), 1–35.

[16] Xun Jiang, Zizheng Guo, Zhuomin Chai, Yuxiang Zhao, Yibo Lin, Runsheng Wang, and Ru Huang. 2023. 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)*. 1–9. doi:10.1109/ICCAD57390.2023.10323938

[17] Andrew B Kahng et al. 2001. Constraint-based watermarking techniques for design IP protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, 10 (2001), 1236–1252.

[18] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. 2011. *VLSI physical design: from graph partitioning to timing closure*. Vol. 312. Springer.

[19] Andrew B Kahng, Stefanus Mantik, Igor L Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang, and Gregory Wolfe. 1998. Robust IP watermarking methodologies for physical design. In *Proceedings of the 35th annual Design Automation Conference*. 782–787.

[20] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. 2015. ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 921–926. doi:10.1109/ICCAD.2015.7372671

[21] Dongfang Li, Wenchao Liu, Xuecheng Zou, and Zhenglin Liu. 2015. Hardware IP protection through gate-level obfuscation. In *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*. IEEE, 186–193.

[22] Yuanzhi Li and Yang Yuan. 2017. Convergence analysis of two-layer neural networks with relu activation. *Advances in neural information processing systems* 30 (2017).

[23] Wei Liang, Xingming Sun, Zhihua Xia, Decai Sun, and Jing Long. 2011. A chaotic IP watermarking in physical layout level based on FPGA. *Radioengineering* 20, 1 (2011), 118–125.

[24] Jai-Ming Lin, Chung-Wei Huang, Liang-Chi Zane, Min-Chia Tsai, Che-Li Lin, and Chen-Fa Tsai. 2021. 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)*. IEEE, 1–8.

[25] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. 2019. DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[26] Jinwei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline F. Y. Young. 2020. CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. doi:10.1109/DAC18072.2020.9218646

[27] Wen-Hao Liu, Stefanus Mantik, Wing-Kai Chow, Yixiao Ding, Amin Farshidi, and Gracieli Posser. 2019. ISPD 2019 initial detailed routing contest and benchmark with advanced routing rules. In *Proceedings of the 2019 International Symposium on Physical Design*. 147–151.

[28] Daniela Sánchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Souvik Hazra, Robert Wille, and Wolfgang Ecker. 2021. A Survey of Graph Neural Networks for Electronic Design Automation. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. 1–6. doi:10.1109/MLCAD52597.2021.9531070

[29] Yi-Chen Lu, Wei-Ting Chan, Vishal Khandelwal, and Sung Kyu Lim. 2022. 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* (Virtual Event, China) *(MLCAD '22)*. Association for Computing Machinery, New York, NY, USA, 97–102. doi:10.1145/3551901.3556476

[30] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. 2020. TP-GNN: A Graph Neural Network Framework for Tier Partitioning in Monolithic 3D ICs. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. doi:10.1109/DAC18072.2020.9218582

[31] Yi-Chen Lu and Sung Kyu Lim. 2022. On Advancing Physical Design Using Graph Neural Networks. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. doi:10.1145/3508352.3561094

[32] Yi-Chen Lu, Sai Pentapati, and Sung Kyu Lim. 2021. The Law of Attraction: Affinity-Aware Placement Optimization using Graph Neural Networks. In *Proceedings of the 2021 International Symposium on Physical Design* (Virtual Event, USA) *(ISPD '21)*. Association for Computing Machinery, New York, NY, USA, 7–14. doi:10.1145/3439706.3447045

[33] Yi-Chen Lu, Tian Yang, Sung Kyu Lim, and Haoxing Ren. 2022. Placement Optimization via PPA-Directed Graph Clustering. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD* (Virtual Event, China) *(MLCAD '22)*. Association for Computing Machinery, New York, NY, USA, 1–6. doi:10.1145/3551901.3556482

[34] Yi-Chen Lu, Tian Yang, Sung Kyu Lim, and Haoxing Ren. 2022. Placement optimization via ppa-directed graph clustering. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*. 1–6.

[35] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.

[36] Paarth Neekhara, Shehzeen Hussain, Xinqiao Zhang, Ke Huang, Julian McAuley, and Farinaz Koushanfar. 2022. FaceSigns: semi-fragile neural watermarks for media authentication and countering deepfakes. *arXiv preprint arXiv:2204.01960* (2022).

[37] Min Ni and Zhiqiang Gao. 2004. Watermarking system for IC design IP protection. In *2004 International Conference on Communications, Circuits and Systems (IEEE Cat. No. 04EX914)*, Vol. 2. IEEE, 1186–1190.

[38] Tingyuan Nie, Tomoo Kisaka, and Masahiko Toyonaga. 2005. A watermarking system for IP protection by a post layout incremental router. In *Proceedings of the 42nd annual Design Automation Conference*. 218–221.

[39] Rachel Selina Rajarathnam, Yibo Lin, Yier Jin, and David Z. Pan. 2020. ReGDS: A Reverse Engineering Framework from GDSII to Gate-level Netlist. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 154–163. doi:10.1109/HOST45689.2020.9300272

[40] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[41] Haoxing Ren and Jiang Hu. 2022. *Machine Learning Applications in Electronic Design Automation*. Springer.

[42] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. 2014. A Primer on Hardware Security: Models, Methods, and Metrics. *Proc. IEEE* 102, 8 (2014), 1283–1295. doi:10.1109/JPROC.2014.2335155

[43] Debasri Saha, Parthasarathi Dasgupta, Susmita Sur-Kolay, and Samar Sen-Sarma. 2007. 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)*. IEEE, 111–116.

[44] Debasri Saha and Susmita Sur-Kolay. 2007. Fast robust intellectual property protection for VLSI physical design. In *10th International Conference on Information Technology (ICIT 2007)*. IEEE, 1–6.

[45] Euschi Salah, Khaldi Amine, Kafi Redouane, and Kahlessenane Fares. 2021. A Fourier transform based audio watermarking algorithm. *Applied Acoustics* 172 (2021), 107652.

[46] Majid Sarrafzadeh and CK Wong. 1996. *An introduction to VLSI physical design*. McGraw-Hill Higher Education.

[47] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2023. Robust GNN-Based Representation Learning for HLS. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323853

[48] Guangyu Sun, Zhiqiang Gao, and Yi Xu. 2006. A watermarking system for ip protection by buffer insertion technique. In *7th International Symposium on Quality Electronic Design (ISQED'06)*. IEEE, 5–pp.

[49] Jiang Wei and Xiaolei Kong. 2010. A framework to analyze different intellectual property systems. In *2010 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE, 178–182.

[50] Hua Xiang, Li-Da Huang, Kai-Yuan Chao, and Martin DF Wong. 2005. An ECO algorithm for resolving OPC and coupling capacitance violations. In *2005 6th International Conference on ASIC*, Vol. 2. IEEE, 873–876.

[51] Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar. 2024. Automated Physical Design Watermarking Leveraging Graph Neural Networks. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*. 1–10.

[52] Ruisi Zhang, Rachel Selina Rajarathnam, David Z Pan, and Farinaz Koushanfar. 2024. ICMarks: A Robust Watermarking Framework for Integrated Circuit Physical Design IP Protection. *arXiv preprint arXiv:2404.18407* (2024).

[53] Xiaorui Zhang, Xun Sun, Xingming Sun, Wei Sun, and Sunil Kumar Jha. 2022. Robust Reversible Audio Watermarking Scheme for Telemedicine and Privacy Protection. *Computers, Materials & Continua* 71, 2 (2022).

[54] Xin Zhong, Pei-Chi Huang, Spyridon Mastorakis, and Frank Y Shih. 2020. An automated and robust image watermarking scheme based on deep neural networks. *IEEE Transactions on Multimedia* 23 (2020), 1951–1961.

[55] Jie Zhou et al. 2020. Graph neural networks: A review of methods and applications. *AI open* (2020).

Received 14 February 2025; revised 28 August 2025; accepted 11 September 2025