Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.082060

ARTICLE

Variational Graph Autoencoder–Based Timing-Driven Initialization Placement

Ziyi Ju[1], Ping Yu[1], Rui Song[1] and Tonglin Chen[1,2,*]

[1]School of Computer Science and Technology, Fudan University, Shanghai, China

[2]Fudan University-China Railway 24th Bureau Joint Laboratory for Intelligent Construction Engineering Technologies on Operational Railway Lines, The China Railway 24th Bureau Group Corporation Limited, Shanghai, China

*Corresponding Author: Tonglin Chen. Email: tlchen18@fudan.edu.cn

Received: 11 March 2026; Accepted: 04 June 2026; Published: 23 July 2026

ABSTRACT: In modern high-performance chip design, achieving timing closure is essential to design success. With the increasing scale and complexity of modern chips, timing-driven placement has become increasingly important. Traditional placement methods primarily focus on minimizing wirelength, but lack timing optimization, making it difficult to meet the strict timing closure requirements of modern designs. Therefore, developing an efficient timing-driven placement method has become a critical challenge in modern chip design. This paper presents a novel timing-driven placement framework that integrates a variational graph autoencoder (VGAE) with a nonlinear mixed-size placement optimizer. The framework identifies timing-violation paths through static timing analysis and dynamically adjusts interconnect weights based on pin-level timing slack, enabling the VGAE to generate an initial placement that prioritizes critical-path optimization. Experimental results on the ICCAD2015 benchmarks show that the proposed method achieves a 25.4% improvement in worst negative slack and a 18.1% improvement in total negative slack compared with DREAMPlace4.0. These results demonstrate its effectiveness in improving timing quality.

KEYWORDS: Timing optimization; timing-driven placement; variational graph autoencoder; weight updating; nonlinear optimization

1 Introduction

Achieving timing closure is a fundamental requirement in modern very large scale integration (VLSI) design. A chip can operate reliably at its target clock frequency only when all timing constraints are met. Otherwise, clock frequency reduction is inevitable, which significantly degrades chip performance [1]. In the physical design flow of integrated circuits, placement is a critical step that strongly affects performance and timing closure. As technology nodes advance and circuits scale up, neglecting timing during placement often triggers expensive downstream iterations or even rollbacks, thereby prolonging the entire design cycle.

Placement has traditionally focused on wirelength minimization, as illustrated in the left column of Fig. 1, which outlines the standard five-stage workflow. With increasingly strict timing-closure requirements, timing optimization has been progressively incorporated into the placement stage, giving rise to timing-driven placement. Existing timing-driven placement techniques can be broadly categorized into three classes: net-based, path-based, and learning-based approaches. Net-based methods [2–4] and more recent approaches [5–7] adjust net weights based on static timing analysis (STA) to guide the placer toward reducing net lengths. Path-based methods [8–11] and more recent approaches [12,13] directly optimize worst negative slack (WNS) and total negative slack (TNS) by manipulating spatial relationships along critical

paths. More recently, machine learning, particularly graph neural networks (GNNs) has found applications in placement prediction [14–17], routing estimation [18], and timing analysis [19,20].

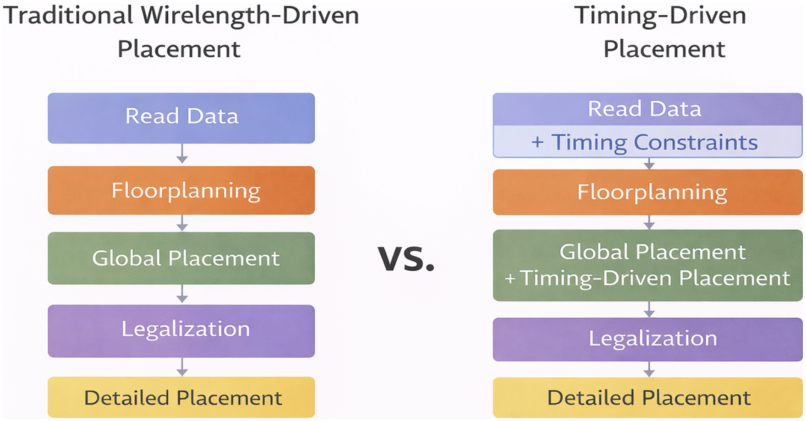


Figure 1: Comparison of wirelength-driven and timing-driven placement frameworks. Timing optimization is usually applied in the later stages of global placement. While legalization and detailed placement refine the result.

Net-based methods offer strong scalability and are well-suited for large-scale circuits, but their optimization granularity is limited because timing violations arise from entire paths rather than individual nets. Path-based approaches are more directly aligned with timing objectives but often suffer from scalability bottlenecks as circuit size and the number of critical paths increase. Existing learning-based methods, although powerful in modeling structural patterns, typically focus on local structures, such as individual nets or isolated timing paths, and often use neural networks as optimization surrogates rather than as global placement guides. Furthermore, as shown in the right column of Fig. 1, conventional timing-driven placement introduces timing optimization only in the later stages of global placement. Early-stage timing analysis is unreliable due to severe cell overlap, which delays timing adjustment. Consequently, suboptimal early placements become difficult to escape, and critical paths may remain trapped in local optima. These limitations motivate a placement framework that (i) captures timing information from a global, landscape-level perspective and (ii) incorporates timing awareness at earlier stages of the placement flow. In this work, we leverage the expressive power of graph neural networks to guide placement decisions globally, enabling earlier and more effective timing optimization.

Building upon this motivation, we propose a framework that generates a timing-optimized initial placement, providing a starting point before conventional timing-driven refinement begins. Our method reshapes the spatial configuration of timing-critical structures at initialization, thereby reducing the likelihood that critical paths become confined to poor local optima. By improving the initial solution itself, the framework enables subsequent timing-driven placement stages to operate from a more favorable geometric configuration. The circuit is first partitioned into clusters such that the number of cells in each cluster remains approximately balanced while the total inter-cluster connectivity is minimized. This abstraction transforms fine-grained cell-level interactions into a coarser group-level representation, reducing modeling complexity while preserving dominant structural dependencies. A variational graph autoencoder (VGAE) is used to learn latent geometric relationships from the clustered circuit graph. By leveraging graph neural networks, the model captures global structural and timing-related patterns across the circuit. This global modeling perspective increases the likelihood that timing-critical clusters are placed closer together in the generated initial placement. Timing-critical paths are dynamically identified through static timing analysis (STA) performed by OpenTimer [21], which provides pin-level slack values and highlights the

interconnections contributing most significantly to timing violations. Based on this timing information, inter-cluster connectivity weights are adaptively updated so that clusters connected by critical nets receive higher emphasis. This guides the VGAE to further reduce their spatial distance in subsequent iterations. This placement–timing feedback loop continues until the worst negative slack (WNS) converges and no further meaningful improvement is observed. The initial placement is then refined by a nonlinear analytical placer such as DREAMPlace [22], which performs global spreading, legalization, and detailed placement, allowing components within each cluster to gradually and evenly expand into the available layout space while preserving the timing advantages established during initialization. Through this integration of clustering, global graph modeling, and adaptive timing feedback, the proposed framework mitigates the scalability limitations of traditional path-based methods and shifts timing awareness from a late-stage correction mechanism to an initialization-level structural guidance strategy.

Experimental studies on synthetic circuits and industrial benchmarks demonstrate that timing-aware initialization improves both path contraction and timing performance in VLSI. On synthetic datasets, qualitative visualizations show that the VGAE model produces structurally coherent placements while effectively contracting timing-critical paths. Quantitative results further confirm stable reductions in critical-path length, validating the model's ability to improve timing-critical path contraction. Component-wise comparison and ablation results further demonstrate the effectiveness of the proposed design choices.

The main contributions of this work are as follows:

- We design a variational graph autoencoder that learns latent geometric relationships reflecting both connectivity and timing-critical interactions. The variational formulation improves robustness to timing uncertainty and increases the likelihood that timing-critical modules are encoded closer in latent space.

  To translate this learned prior into effective timing improvement, we introduce an interconnect weight update mechanism based on pin-level slack derived from STA results. This mechanism progressively injects timing-critical information into the netlist graph, allowing the learned representation to better reflect timing sensitivity and enabling effective timing-aware behavior from the early stages of optimization.
- The VGAE-generated timing-aware initial placement integrates seamlessly with analytical placers and conventional STA engines. It improves WNS and TNS and reduces placement iterations without requiring modifications to downstream industrial tools.

2 Related Work

Although numerous timing-driven placement strategies have been explored in the past decades, their design philosophies can generally be grouped into three categories based on how timing information is incorporated: net-based, path-based, and machine-learning-driven approaches.

Net-Based Study. Net-based timing-driven placement improves timing by adjusting weights or imposing constraints on timing-critical nets. Early approaches [23] combined net-length constraints with partitioning but lacked iterative refinement, while later works integrated force-directed placers [24] to enable incremental optimization [25]. More principled formulations, such as the Lagrangian relaxation framework in [26], dynamically update net weights via slack-based multipliers, with performance depending on the underlying optimization engine. Similarly, study [27] adopts a momentum-based weight update inspired by backpropagation [28] and integrates it with an electrostatics-based analytical placer [22] for effective timing refinement. Other representative net-based methods [2–4,6] demonstrate strong scalability; work [7] improves a multilevel analytical placement framework by introducing slack-aware reweighting of timing-critical nets. however, since timing is optimized indirectly through net-level adjustments rather than explicit path modeling, their correlation with path-level violations remains limited.

Path-Based Study. Path-based timing-driven placement explicitly optimizes end-to-end timing paths rather than individual nets. Early work [29] integrates path-delay analysis into simulated annealing to guide placement decisions, while Kahng et al. formulate timing-driven placement as a continuous min–max optimization problem [9] that directly minimizes worst-path delay through STA-based edge reweighting. Constraint-based approaches such as [10] introduce pseudolinks to transform path-delay objectives into force-directed constraints, and incremental frameworks like FlowPlace [11] optimize delay sensitivities weighted by path slack. Other representative methods [8,12] explicitly model source-to-sink timing propagation to improve WNS. More recently, Shi et al. [13] propose a GPU-accelerated path-based timing-driven global placement framework that integrates critical-path information into DREAMPlace through fine-grained pin-to-pin attraction and RC-aware quadratic distance loss. TD-Placer [30] is another path-based method targeting heterogeneous FPGAs, but its FPGA-specific assumptions limit its applicability to general VLSI placement. Despite their higher timing fidelity compared to net-based abstractions, path-based approaches are fundamentally limited by poor scalability due to the rapid growth of timing paths in modern circuits.

Machine Learning-Based Study. Recent advances in machine learning have introduced neural-network-based approaches into electronic design automation(EDA), demonstrating improved placement quality through learned structural representations [15]. The differentiable timing-driven placement framework in [20] embeds static timing analysis (STA) into the neural computation graph by treating STA as forward propagation, enabling end-to-end optimization of WNS/TNS via gradient-based coordinate updates. Other works [14,16,17,19,31] leverage graph neural networks for placement modeling. A separate line of recent work casts placement as a sequential decision problem and applies reinforcement learning, including GraphPlacement [32] and MaskPlace [33]; these methods, however, target macro placement and wirelength minimization rather than timing closure, and therefore address a different problem setting than ours. Moreover, recent studies have also explored generative adversarial learning for placement optimization. For example, GAN-Place [34] improves open-source GPU placers by adversarially transferring placement quality from Synopsys ICC2, thereby enhancing wirelength, congestion, and timing-related metrics. However, its effectiveness relies on commercial placement databases as imitation targets and the learned objective mainly captures similarity to existing commercial solutions rather than explicitly modeling timing-critical circuit structures. However, existing learning-based methods generally do not incorporate path-level slack information into graph topology nor perform timing-driven placement at the initialization stage.

In summary, each class of methods offers distinct advantages and faces specific limitations. Net-based techniques provide excellent scalability but have limited correlation with timing objectives. Path-based strategies directly optimize timing but struggle with the explosive growth of timing paths. Recent learning-based methods introduce data-driven modeling yet are seldom tailored for timing optimization. Our approach fills this gap by embedding STA timing information directly into the VGAE's inference process, enabling timing-aware spatial consideration during placement.

3 Method

This section provides a detailed description of the proposed timing-driven placement methodology, covering the overall framework as well as the core technical components. The overall framework of the proposed timing-driven placement methodology is illustrated in Fig. 2. First, the circuit modules are clustered into several groups, and the inter-group connectivity is analyzed to generate corresponding training data for our variational graph autoencoder (VGAE). The trained VGAE then generates an initial placement. Based on this placement, critical paths are extracted through static timing analysis (STA). A novel weight update mechanism is applied to the inter-group connections, and the updated weights are fed back into the

VGAE to generate a refined placement. This process is iteratively performed until the timing is optimized. Finally, the legal placement is provided by legalization and detail placement.

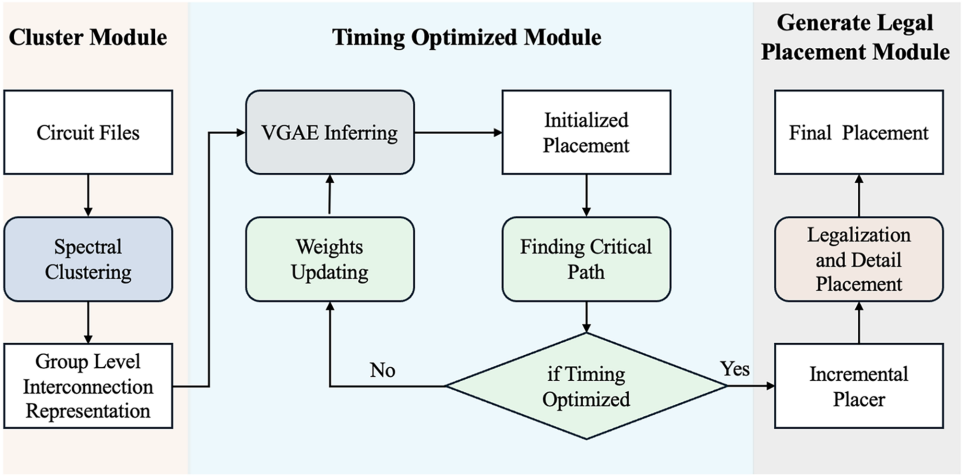


Figure 2: The framework of our timing driven placement has three sections, including cluster modules, timing optimization modules and generate legal placement module. In the flowchart, data are illustrated as white-filled boxes, methods as blue blocks, and operations as rounded rectangles.

3.1 Cluster Module

To manage the computational complexity of million-scale circuit graphs, we employ spectral clustering to compress standard cells into a smaller number of coarse-grained modules. The circuit netlist is first represented by its graph Laplacian L. Since finding an optimal k-way discrete partition is NP-hard, spectral clustering relaxes the problem by using the eigenvectors of L as a continuous embedding space.

Instead of computing the full set of k eigenvectors—which is infeasible for large sparse matrices—we follow the approximation strategy in [35]. As reported in [35], the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix L provides structural position information, and node partitions can be determined by identifying large gaps in this one-dimensional embedding. Specifically, we extract the top 10 non-trivial eigenvectors of L to obtain a low-dimensional representation that preserves major structural variations. The resulting spectral embedding is then clustered using k-means, compressing millions of cells into approximately $k = 400$ modules.

Although this approximation introduces minor loss in clustering precision, it substantially reduces computational cost and enables scalable timing-driven graph learning in subsequent stages. The choice $k = 400$ is the result of a sensitivity sweep over $k \in \{200, 300, 400, 500, 600\}$; smaller k aggregates too many cells per cluster and dilutes timing-critical sub-structures, while larger k inflates spectral-clustering and VGAE-inference cost without further timing gains. The supporting numbers are reported in Section 4.6.

3.2 Timing Optimization Module

3.2.1 Synthetic Training Data Construction

The synthetic training dataset is constructed as follows. We begin by creating an empty square chip area and uniformly dividing it into an $m \times n$ grid, where each grid cell represents one module. For each module, its neighbors are determined in ascending order of geometric distance, and a larger number of connections is assigned to closer neighbors. To generate an "optimal" placement dataset, two conditions must

be satisfied: 1) the inter-module connectivity must be consistent with the minimal total wirelength implied by the placement, and 2) no overlap is allowed between modules. Following methodologies similar to those in [16,36], the resulting pseudo-placement preserves both ideal spatial locality and the underlying graph topology. In particular, modules with larger interconnect weights are placed closer to each other, ensuring that the generated data reflects the fundamental principle that topologically important modules should also be placed close to one another. The synthetic training data generation pipeline is shown in Fig. 3. This dataset provides an effective supervisory signal for training the variational graph autoencoder and approximates the structural constraints observed in real VLSI layouts.

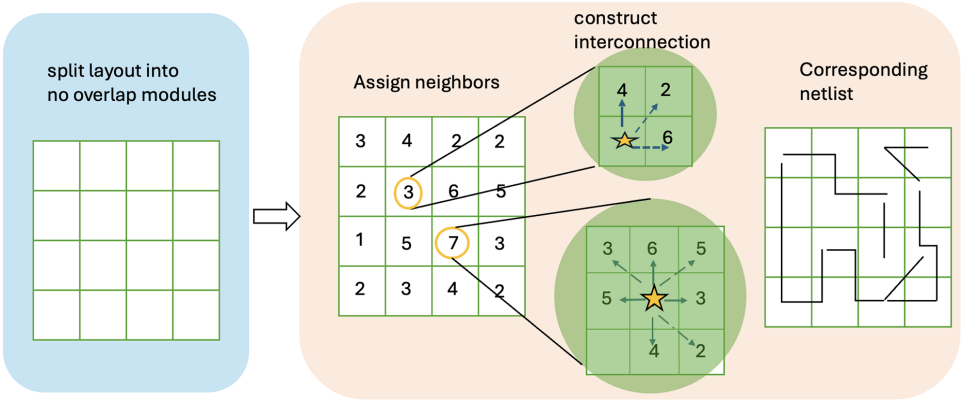


Figure 3: Pipeline for synthetic training dataset generation. The layout is first partitioned into non-overlapping modules. Neighboring modules are then assigned interconnections, and the resulting connections are converted into the corresponding netlist.

In our experiments, the synthetic dataset contains 1000 circuits in total, which are divided into training, validation, and test sets with a ratio of 8:1:1. The model is trained with a batch size of 400, where each batch corresponds to one circuit graph.

3.2.2 Variational Graph Autoencoder Model

The proposed placement initializer is built upon a variational graph autoencoder (VGAE) [37], whose architecture is illustrated in Fig. 4. The model takes the inter-module connectivity as input and predicts the corresponding module positions in the placement plane. Formally, the encoder takes the adjacency matrix A and the node feature matrix F as inputs, where each entry w_{ij} in A represents the number of connections between modules i and j. The node feature of each module is defined as the normalized degree, which reflects the relative connectivity strength of the node in the graph as defined in Eq. (1), where d_i denotes the degree of node i.

$$x_i = \frac{d_i}{\max d} \tag{1}$$

The VGAE encoder maps each node to a latent Gaussian distribution in Eq. (2), where the mean and variance are generated by a graph neural network. If two modules have higher connectivity, their latent representations are more strongly coupled. After node information is aggregated through the graph neural network(GNN) layers, two parallel graph networks $g(\cdot)$ and $h(\cdot)$ generate the mean μ_x and variance σ_x of a Gaussian latent variable Z. After obtaining the mean and variance, we should sample the latent representation. Sampling directly will break the backpropagation, therefore a reparameterization trick is employed in VGAE $z_i = \mu_i + \sigma_i \odot \epsilon$, where ϵ is a learnable parameters for each module.

$$q(\mathcal{Z}_i|X,A) = \mathcal{N}(\mu_i, \sigma_i^2 I); \quad \mu = g_\mu(X, A_\omega); \quad \sigma = h_\sigma(X, A\omega) \tag{2}$$

The decoder, implemented using multilayer perceptrons with nonlinear activations, maps latent samples Z back to predicted module positions (e.g., 2-D coordinates). Unlike the traditional unsupervised VGAE, our model is supervised using pseudo-optimal module placements: the target labels encode module coordinates derived from optimal or near-optimal synthetic layouts. Through supervised learning on these training samples, the model learns to generate placements that approximate the desired spatial relationships while respecting the graph topology.

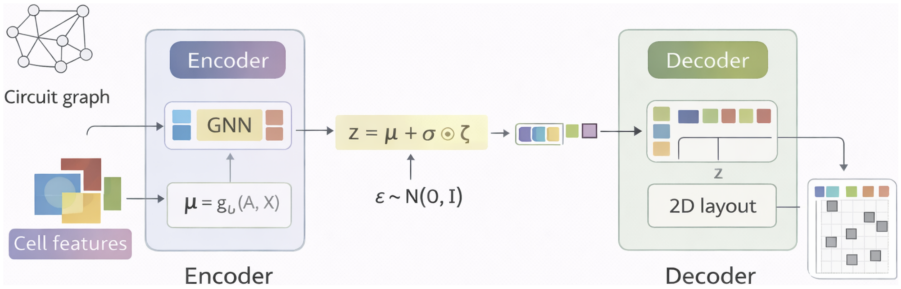


Figure 4: The architecture of variational graph autoencoder.

The loss function of the proposed variational graph autoencoder consists of two parts. The first part in Eq. (3) measures the loss between the predicted placement and the target (pseudo-optimal) module positions.

$$L_{\text{place}} = \sum_{i=1}^{m \times n} (\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2 \tag{3}$$

Here, $\hat{x}_i, \hat{y}_i$ denote the predicted coordinates of module i, while x_i, y_i are the corresponding ground-truth labels. The second part in Eq. (4) is the Kullback–Leibler divergence that regularizes the latent distribution learned by the encoder:

$$D_{\text{KL}}(Q \parallel P) = D_{\text{KL}}(Q(z|X,A) \parallel P(z)) = \sum_x \frac{1}{2}(\mu_x^2 + \sigma_x^2 - \log \sigma_x^2 - 1) \tag{4}$$

Here, P denotes the prior Gaussian distribution. Specifically, the prior distribution is defined as $P = p(z) = \mathcal{N}(0,1)$, which assumes an independent standard Gaussian latent distribution. Q is the predicted latent distribution, also referred to as posterior distribution, i.e., $Q = q(z|X,A)$ generated by VGAE model. Forcing the predicted latent distribution to align with a standard Gaussian prior helps ensure that nodes with similar connectivity are embedded close to each other. The overall training loss function shown in Eq. (5) is the sum of the reconstruction loss and the Kullback–Leibler divergence:

$$\text{Loss} = L_{\text{place}} + D_{\text{KL}}(Q \parallel P) \tag{5}$$

This combined loss enables the model to learn the underlying mapping between inter-module connectivity and spatial locations. Once trained, the VGAE can infer 2D coordinates directly from real circuit connectivity, providing timing-aware initial positions for all modules.

Implementation details. The encoder includes two parallel branches $g_\mu(\cdot)$ and $h_\sigma(\cdot)$ share the same encoder backbone and produce μ and $\log \sigma^2$, respectively. Each branch implemented as two stacked

graph convolutional layers with hidden dimension 2×64 and 64×32. The decoder is a two-layer MLP ($32 \to 16 \to 2$) with ReLU activations that maps latent code z_i to the predicted 2-D coordinates of module i. The model is trained on the synthetic dataset described in Section 3.2.1 for 10,000 epochs using the Adam optimizer with learning rate 1×10^{-3}. All weights are initialized with Xavier uniform initialization.

3.2.3 Timing-Aware Weight Update Mechanism

After obtaining a placement from the trained VGAE, we perform static timing analysis (STA) and use the resulting critical-path information to refine the inter-module graph without updating the VGAE parameters. For each subnet e on the selected critical path, we map the driver and sink cells to their corresponding modules i and j. A timing criticality score is then computed as

$$\rho_e = \mathrm{clip}\left(\frac{\min(s_e, 0)}{\mathrm{WNS}}, 0, 1\right), \tag{6}$$

where s_e is the arrival-time slack of subnet e reported by STA. The $\mathrm{clip}(\cdot, 0, 1)$ operator plays two roles: (i) it zeros out non-violating subnets so that they do not contribute to the inter-module reinforcement score, and (ii) it caps numerical residues that may exceed 1 due to floating-point rounding. Hence $\rho_e \in [0, 1]$ measures the relative criticality of subnet e, with $\rho_e = 1$ corresponding to the worst-violating subnet. The module-pair reinforcement score is defined as

$$g_{ij}^{(t)} = \sum_{e \in \mathcal{E}_{ij} \cap \mathcal{P}_K} \rho_e, \tag{7}$$

where $\mathcal{E}_{ij}$ denotes the set of subnets whose driver and sink belong to modules i and j, and $\mathcal{P}_K$ is the set of top-K critical paths returned by STA. Instead of retraining the VGAE, we update the adjacency weights and employ the trained model for iterative inference:

$$w_{ij}^{\mathrm{cand}} = w_{ij}^{(0)} \left(1 + g_{ij}^{(t)}\right), \tag{8}$$

where $w_{ij}^{(0)}$ is the original interconnect weight derived from the netlist, and the candidate update is then smoothed with a momentum coefficient $\alpha \in (0, 1)$:

$$w_{ij}^{(t+1)} = (1 - \alpha) w_{ij}^{(t)} + \alpha\, w_{ij}^{\mathrm{cand}}. \tag{9}$$

The updated adjacency matrix is clipped and normalized before being fed back into the VGAE. In this way, timing-critical module pairs are progressively strengthened in the graph representation, encouraging the frozen VGAE to generate placements with shorter distances along timing-sensitive connections.

For the timing-aware weight update, the critical paths are ranked by slack, and the worst paths are selected as candidates to guide the edge-weight refinement. Regarding the iterative VGAE/STA refinement, the update loop continues as long as a better WNS is obtained than in previous iterations. The process stops once no further improvement in the best WNS is observed.

3.3 Legal Placement Generation Module

After obtaining a timing-optimized initial placement, we employ an analytical placer for incremental refinement within our framework. Because the VGAE only predicts coordinates for cluster centroids, we map cluster-level positions back to individual cells before invoking the analytical placer: every cell that belongs to cluster c is initialized at the centroid coordinate of c. The resulting cell-level coordinates are then

handed to DREAMPlace4.0, which performs cell spreading, density relaxation, legalization and detailed placement to obtain a fully legal layout. Specifically, DREAMPlace4.0 [27] is used as the backend engine. Although DREAMPlace4.0 supports timing-driven placement, it only activates timing optimization during the later stages of global placement. In contrast, our timing-aware initial placement provides a better starting point for DREAMPlace4.0, enabling the placer to converge more efficiently. DREAMPlace4.0 then performs global placement beginning from our initialization and continues until either the target density is achieved or the maximum iteration limit is reached. Finally, legalization and detailed placement [38] are applied to produce a fully legal and routable final layout within our framework.

4 Experiments Section

In this section, we comprehensively evaluate the proposed framework through a series of carefully designed experiments. The evaluation comprises five parts: placement learning, critical path optimization, component-wise comparative analysis, ablation study, and sensitivity analysis. Collectively, these experiments serve complementary purposes: evaluations on synthetic circuits validate the ability of our Variational Graph Autoencoder (VGAE) to generate high-quality placements and shorten target timing paths, while experiments on industrial datasets demonstrate the timing improvements achieved in realistic VLSI circuits. Detailed results and corresponding discussions are presented below.

Experimental environment. All experiments are conducted on a Linux workstation equipped with a 96-core 2.40 GHz Intel Xeon CPU, 128 GB of main memory, and an NVIDIA GeForce RTX 3090 GPU with 24 GB of GPU memory. The operating system is Ubuntu Linux (64-bit), and all implementations are executed in a Python–C++ hybrid environment. The graph neural network components are implemented using PyTorch with CUDA acceleration enabled, while timing analysis is performed using OpenTimer integrated into the placement framework. For fair comparison, all baseline methods are executed under the same hardware and software configurations. Multi-threading is enabled for CPU-based components where applicable, and GPU acceleration is used exclusively for neural network training and inference.

4.1 Dataset Description

Synthetic circuit. To emulate timing-critical structures without relying on technology-specific delay models, we construct a set of synthetic critical paths. Starting from randomly selected modules, paths are generated by traversing neighboring modules according to the adjacency matrix until a predefined length is reached or no further unvisited neighbor exists. Each module is assigned to exactly one path, resulting in a collection of disjoint paths $P = \{p_1, p_2, \ldots, p_n\}$, sorted by decreasing length. The longest path is treated as the most timing critical. This synthetic circuit is used to evaluate whether our model can generate high quality placement and effectively shorten the geometric length of target paths, thereby validate its capability to optimize timing performance.

Real industrial dataset ICCAD2015. The ICCAD2015 dataset is a timing-driven placement benchmark suite [39]. In our experiments, all components are treated as movable, while fixed I/O pins retain their original locations. The detailed circuit statistics are summarized in Table 1. We evaluate the proposed framework by comparing placement results against those obtained from DREAMPlace4.0 [27] under its default configuration, which serves as the baseline. We keep all DREAMPlace hyperparameters at their default values. Timing analysis is performed using OpenTimer [21], which is an open-source static timing analysis engine. It reports the slack of each pin and the components on critical paths.

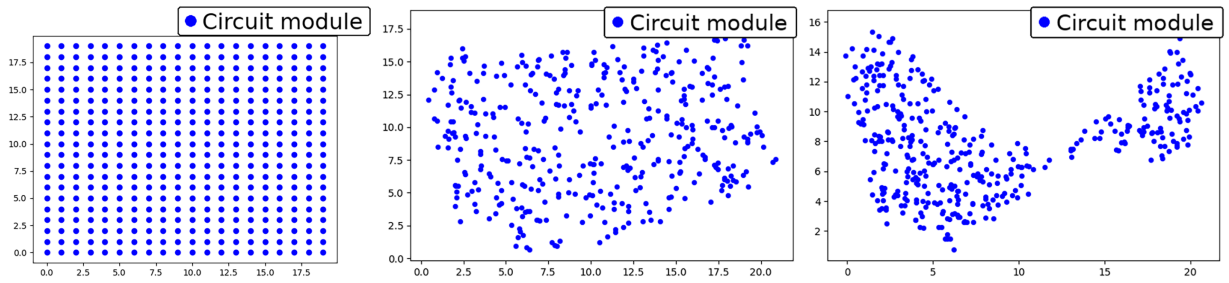
Table 1: Features of ICCAD2015 dataset.

Design	Movable Components	Fixed I/Os	Nets	Total Pins	Movable Pins
superblue1	1,209,716	6528	1,215,710	3,767,494	3,760,966
superblue3	1,213,253	6482	1,224,979	3,905,321	3,898,839
superblue4	795,645	6623	802,513	2,497,940	2,491,317
superblue5	1,086,888	4129	1,100,825	3,246,878	3,242,749
superblue7	1,931,639	6501	1,933,945	6,372,094	6,365,593
superblue10	1,876,103	12,257	1,898,119	5,560,506	5,548,249
superblue16	981,559	4449	999,902	3,013,268	3,008,819
superblue18	768,068	3978	771,542	2,559,143	2,555,165

4.2 Placement Learning

We evaluate the proposed placement framework in two stages. First, experiments on synthetic circuits demonstrate its ability to produce reasonable layouts and shorten critical paths. Then, evaluations on realistic industrial circuits (ICCAD2015) show its effectiveness in practical timing optimization in complex VLSI designs.

Results on synthetic circuit. To illustrate the placement capability of our model, we present placement results generated by VGAE model trained on synthetic circuits in Fig. 5. Fig. 5a shows the ground-truth placement, where the interconnections features among modules obey the rules described in Section 3.2.1. Fig. 5b presents the placement produced by our well-trained VGAE model, demonstrating its ability to generate structurally coherent placement. In contrast, Fig. 5c shows the result from an insufficiently trained model, which fails to organize modules effectively, highlighting the importance of proper model training.

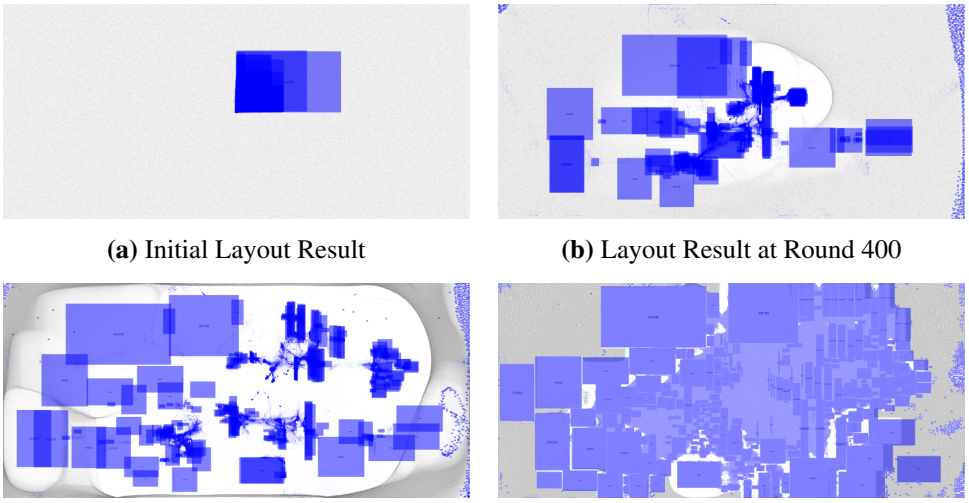


(a) Ground-truth Result **(b)** Well Trained VGAE Result **(c)** Insufficiently Trained Result

Figure 5: The placement results on synthetic circuit.

Results on ICCAD2015 dataset. The following compares the placement evolution of DREAMPlace4 (Fig. 6a–d) and the proposed VGAE-based framework (Fig. 7a–d). Note the grey particles in these figures represent the fillers which has no connections with any components in the circuit. These fillers are used to occupy the empty space in the layout region to assist the placement algorithm in achieving a more balanced distribution of components. All components are movable. Each blue square or particle represents a circuit component, with its area proportional to the component size. The subfigures illustrate the placement results at different stages of the optimization flow. These two figures show clear differences between the two methods, from the initial placement to the final legal placement. As the placement iterations progress, the components gradually spread across the layout region while maintaining the connection relationships. In Fig. 6a, all

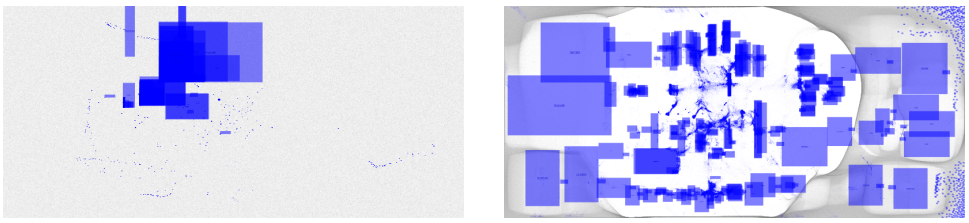
components are initially concentrated in the center of the layout region, whereas in Fig. 7a, the timing-aware initial placement generated by our VGAE model already incorporates timing optimization. Meanwhile, because the components in the initial placement are more evenly distributed, the number of subsequent placement iterations is also reduced. Taking the same iterations such as 600th as an example, the components in Fig. 6c still overlap seriously, however, in Fig. 7c, most components have been well distributed across the layout region resulting in a smaller number of iterations to reach convergence. Finally, both methods achieve a legal placement without overlap, as shown in subfigures (d) of both figures. Table 2 shows the WNS, TNS and iteration counts for our approach and DREAMPlace4 method. From the table, we can claim that our approach improves the timing performance in industrial dataset as well. Our method shows a clear average advantage over DREAMPlace4.0, achieving roughly 18% improvement in TNS, 25% improvement in WNS, and about 25% fewer iterations. To further evaluate the tradeoff between timing improvement and global wirelength, we also report HPWL for the proposed method and DREAMPlace4.0, as shown in Table 3. The results indicate that our method consistently incurs a moderate HPWL increase, ranging from 4.0% to 8.0% across the evaluated benchmarks. This behavior is expected because the proposed framework explicitly prioritizes the contraction of timing-critical paths. While this improves WNS and TNS, it can also stretch non-critical nets and therefore increase the total wirelength. These results suggest that the timing improvement of the proposed method is achieved at the cost of a moderate wirelength overhead.



(a) Initial Layout Result **(b)** Layout Result at Round 400

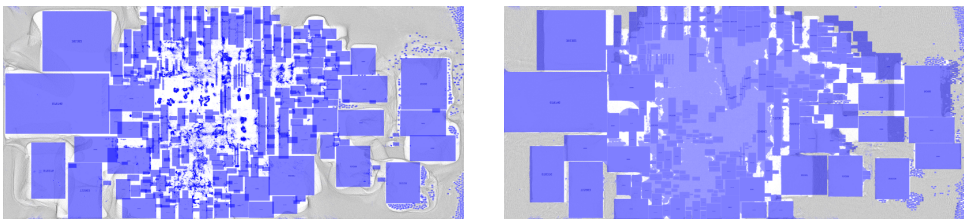
(c) Layout Result at Round 600 **(d)** Legalized Layout Result

Figure 6: Placement process of DREAMPlace4.0 on superblue5.



(a) Initial Layout Result **(b)** Layout Result at Round 400

Figure 7: (Continued)



(c) Layout Result at Round 600 **(d)** Legalized Layout Result

Figure 7: Placement process of our approach on superblue5.

Table 2: The comparisons of timing performance and iteration about DREAMplace4 and our method. TNS:($\times 10^5 ps$), WNS:($\times 10^3 ps$), $Improvement_{WNS\&TNS\&iteration} = \frac{DREAMPlace4.0 - Our\ result}{DREAMPlace4.0}$.

Benchmark	DREAMPlace4.0			Ours			Improvement		
	TNS	WNS	Iteration	TNS	WNS	Iteration	TNS	WNS	Iteration
superblue1	−960.6	−49.9	1112	−559.3	−30.3	819	41.8%	39.3%	26.3%
superblue3	−1526.9	−177.8	977	−1163.2	−125.5	864	23.8%	29.4%	11.6%
superblue4	−355	−29.4	1165	−294.4	−22.9	782	17.1%	22.1%	32.9%
superblue5	−1166.6	−74.9	962	−990.2	−52.8	707	15.1%	29.5%	26.5%
superblue7	−170.9	−29.5	1100	−162.5	−25.3	804	4.9%	14.2%	26.9%
superblue10	−1143.4	−84.6	1166	−918	−52.9	782	19.7%	37.5%	32.9%
superblue16	−775.5	−52.2	662	−686.7	−44.3	592	11.5%	15.1%	10.6%
superblue18	−91.6	−33.5	962	−81.3	−28.2	655	11.2%	15.8%	31.9%

Table 3: HPWL compaisons of different timing-driven placement methods.

Benchmark	Our Framework	DREAMPlace4.0	HPWL Overhead
superblue1	624.8	584.0	6.99%
superblue3	609.1	569.2	7.01%
superblue4	398.2	379.6	4.9%
superblue5	678.0	639.6	6.0%
superblue7	717.3	670.4	7.0%
superblue10	1261.4	1212.9	4.0%
superblue16	555.1	528.7	4.99%
superblue18	320.3	296.6	7.99%

To position our work against existing timing-driven placement frameworks, we further include in Table 4 a comparison with [7,20]. We report the WNS/TNS values that the original paper publishes on the corresponding ICCAD2015 superblue designs. Recent works provide competitive performance than our method. However, our goal is not to outperform all existing timing-driven approaches, but rather to propose a timing-aware initialization framework that can be seamlessly integrated into existing placement flows. From an industrial perspective, our method can be better integrated with industrial placement tools for three main reasons. First, it maintains compatibility with existing analytical placement frameworks by operating as a timing-aware initialization stage rather than replacing the entire flow. Second, the use of

continuous RC modeling and fine-grained timing optimization (at path and pin levels) aligns well with the inputs and outputs of standard STA tools, enabling seamless iterative interaction. Third, unlike methods that rely on indirect net reweighting or assume pre-placed macros, our framework performs unified optimization without strong assumptions, making it more adaptable to realistic mixed-size placement scenarios.

Table 4: Comparison with timing-driven placement baselines on ICCAD2015. TNS: ($\times 10^5$ ps), WNS: ($\times 10^3$ ps).

Benchmark	Diff-TDP [20]		Distribution-TDP [7]		Ours	
	TNS	WNS	TNS	WNS	TNS	WNS
superblue1	−74.9	−10.7	−9.3	−42.1	−559.3	−30.3
superblue3	−39.4	−12.4	−12.2	−26.6	−1163.2	−125.5
superblue4	−82.9	−8.4	−8.9	−123.3	−294.4	−22.9
superblue5	−108.1	−25.2	−31.6	−70.4	−990.2	−52.8
superblue7	−46.4	−15.2	−17.2	−95.9	−162.5	−25.3
superblue10	−558.1	−22.0	−25.9	−691.1	−918	−52.9
superblue16	−87.0	−10.9	−12.2	−56.0	−686.7	−44.3
superblue18	−19.3	−8.0	−5.3	−19.2	−81.3	−28.2

Beyond timing quality, we also report the runtime of the additional components introduced by our framework on superblue1 in Table 5. The dominant overhead is the one-time spectral clustering step, while the GPU-based VGAE inference and the per-iteration weight update are negligible compared with the cost of an STA call. As shown in the last row of the table, although the weight update mechanism introduces additional overhead in each iteration, it also substantially reduces the total number of iterations. As a result, the overall placement runtime can be reduced when preprocessing time is excluded. Even after accounting for the additional preprocessing stage introduced by the proposed framework, the total time required to obtain a timing-converged design remains lower than that of a placement flow that does not incorporate timing awareness during placement.

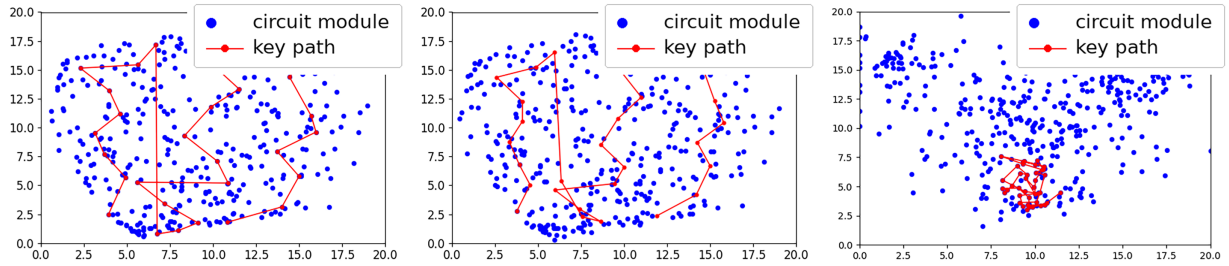
Table 5: Runtime compaisons with our method and DREAMplace4 on superblue 1. "–" indicates that the stage is not present in the baseline.

Stage	Our Framework	DREAMPlace4.0
Cluster modules	125.0	–
VGAE inference (per call)	0.001	–
STA (per call)	19.1	19.1
Weight update (per call)	0.4	–
Iterations	819	1112

4.3 Critical Path Optimization

Building on the visual results, we further quantify the reduction in the geometric length of the target critical path. By reporting precise numerical results on synthetic instances, this experiment provides measurable evidence that our approach consistently shortens critical paths. We additionally conduct the same path-length evaluation on the ICCAD2015 benchmark suite to examine whether similar reductions can be observed in realistic large-scale designs.

Results on synthetic circuits. As illustrated in Fig. 8, the placements generated by the VGAE model vary significantly under different interconnect weight settings. Fig. 8a shows the placement obtained with the original interconnect weights, where the critical path, highlighted in red, spans nearly the entire layout region. Fig. 8b presents the placement produced with increased interconnect weights, in which the same critical path becomes shorter as the modules along the path are spatially contracted. Finally, Fig. 8c illustrates the placement produced when the interconnect weights exceed the effective operating range of the model; in this case, excessive weighting distorts the global layout structure and leads to degraded placement quality. As we can see the placement structure has been destroyed.



(a) Baseline Interconnect Weights **(b)** Increased Interconnect Weights **(c)** Significantly Enhanced Weights

Figure 8: Placement on synthetic circuit. Red line represents the most critical path.

In addition to the visualized demonstrations, we evaluate the geometric length of critical paths. Fig. 9 presents the length evolution of the most critical path as the interconnect weight increases. Although minor fluctuations are observed as the interconnect weight increases, the overall trend shows a clear reduction in path length.

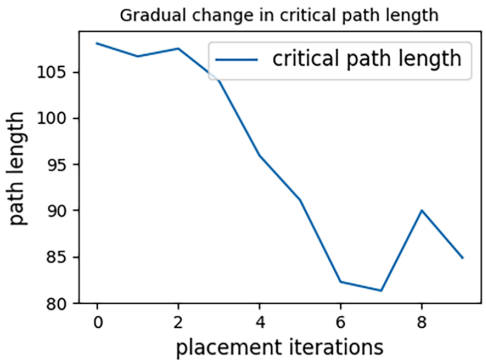
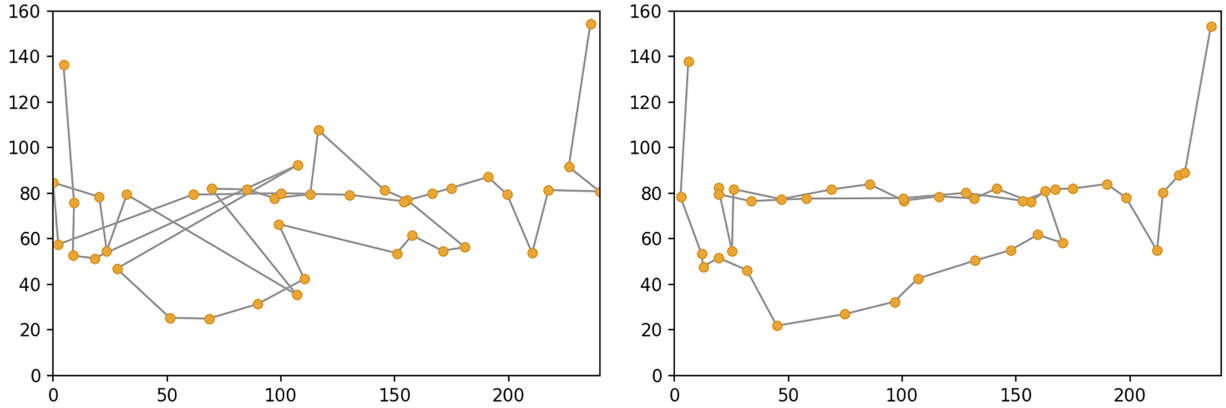


Figure 9: Critical path length variation.

Results on ICCAD2015 dataset. As shown in Fig. 10, the most critical path in superblue1 changes significantly before and after optimization. Fig. 10a presents the critical path generated by our model using the original circuit interconnections, where the modules along the path are distributed across the layout region, resulting in a long path length of 1258.7. Such a dispersed placement increases interconnect distance and routing complexity, leading to larger wire delay and degrading timing performance. In contrast, Fig. 10b shows the optimized placement produced by our framework, where the modules along the same critical path are placed more compactly, reducing the path length to 967.3. This spatial contraction reflects a more timing-aware placement strategy. By globally modeling cell interactions through the VGAE-based framework and incorporating slack information from critical paths, the optimization avoids poor local configurations and

guides the layout toward shorter timing-critical connections. The reduced path length lowers wire delay and improves signal propagation, demonstrating effective timing optimization in realistic industrial circuits.



(a) Critical Path Length on VLSI Circuit= 1258.7 **(b)** Critical Path Length on VLSI circuit = 967.3

Figure 10: Visualization of the most critical path optimized on industrial-scale circuit.

4.4 Component-Wise Comparative Analysis

We conduct a component-wise comparative analysis on ICCAD2015 timing-driven placement benchmarks, reporting final WNS and TNS after the complete placement flow. To ensure fairness, all methods share the same downstream analytical placer configuration and timing analysis pipeline; only the component under evaluation is changed. We fix training schedule, all hyperparameters and timing metrics on the Superblue18.

Comparison of clustering methods. On Superblue18, connectivity-aware clustering consistently outperforms structure-agnostic strategies. As shown in Table 6, random balanced clustering [40] produces the worst timing results, indicating that size balancing alone is insufficient when connectivity is ignored. Greedy graph-growth clustering [41] improves both WNS and TNS by preserving local adjacency during cluster formation. Louvain clustering [42] further improves timing by grouping densely connected regions, although its performance depends on explicit size constraints. Balanced METIS k-way partitioning [43] achieves additional gains by minimizing inter-cluster cut edges under balance constraints. The best results are obtained with spectral clustering, which leverages the normalized Laplacian to capture global connectivity structure and therefore produces clusters that better preserve circuit topology. These results indicate that clustering quality strongly influences the fidelity of the coarse module graph and directly affects the effectiveness of timing-aware placement initialization.

Table 6: Clustering method comparison on ICCAD2015 (superblue18). Larger WNS and TNS indicate better timing performance.

Clustering Method	WNS (10^3 ps)	TNS (10^5 ps)
Random (balanced)	−34.9	−98.7
Greedy graph clustering	−32.8	−93.6
Louvain (size-capped)	−31.6	−90.4
METIS k-way (balanced)	−30.3	−87.1
Spectral (normalized Laplacian)	**−28.2**	**−81.3**

The bold text is used to highlight the best performance value in the table.

Model backbone comparative analysis. With spectral clustering fixed, we compare several learning models for predicting the timing-aware initial placement. As shown in Table 7, models that do not exploit graph structure (multiple layer perceptron (MLP) and autoencoder) perform significantly worse, confirming that inter-module connectivity is critical for capturing placement-relevant dependencies. Deterministic GNN predictors such as graph convolutional network (GCN) and GraphSAGE achieve notable gains in both WNS and TNS, indicating that neighborhood aggregation effectively captures local connectivity patterns. Among all models, the proposed VGAE achieves the best timing performance. Compared with deterministic GNNs, VGAE introduces a latent-variable formulation that learns a probabilistic embedding of the circuit graph, which is more beneficial for timing improvement. This property is particularly beneficial for placement initialization, where modules with strong connectivity or timing interaction should be spatially correlated, and this relationship is naturally captured by the latent similarity learned by VGAE.

Table 7: Model comparison on ICCAD2015 (superblue18) with spectral clustering fixed. Larger WNS and TNS indicate better timing performance.

Model	WNS (10^3 ps)	TNS (10^5 ps)
MLP (no graph structure)	−56.7	−105.8
Autoencoder (no graph structure)	−49.2	−99.6
GCN (deterministic GNN)	−31.0	−88.2
GraphSAGE (deterministic GNN)	−34.1	−86.7
VGAE (ours)	**−28.2**	**−81.3**

The bold text is used to highlight the best performance value in the table.

4.5 Ablation Study

We perform a systematic ablation study by selectively removing key components—clustering module, VGAE-based representation learning, and iterative weight update mechanism—in various combinations. The specific experimental configurations and corresponding results are shown in Table 8. The Baseline corresponds to the default timing-driven mode of DREAMPlace4, where all modules are initially spread from the layout center. The Clustering + VGAE setting illustrates the effectiveness of our learned initial placement without iterative timing-aware refinement. Compared with the baseline, this configuration substantially reduces the number of iterations (962 → 688), demonstrating improved convergence behavior. However, without the weight update mechanism, the final timing quality degrades, resulting in worse WNS and TNS than the baseline. This suggests that connectivity-based learned initialization alone is insufficient for timing optimization. The reason is that structural connectivity in the original netlist does not always align with timing-critical relationships; some timing-critical modules may not be geometrically close or strongly correlated in the connectivity graph. These results show that the weight-update mechanism is an essential component of the framework rather than an optional enhancement, since it resolves the mismatch between connectivity topology and actual timing structure. In other words, the learned initialization provides a useful placement prior, but meaningful timing improvement is achieved only when it is combined with the STA-guided weight update mechanism, which aligns the graph representation with timing-critical information.

Table 8: Ablation study on key components of the proposed framework on superblue18. Larger WNS and TNS indicate better timing quality.

Setting	Clustering	VGAE	Weight Update	WNS (10^3 ps)	TNS (10^5 ps)	Iteration
Baseline	×	×	×	−33.5	−91.6	962
Clustering + VGAE	✓	✓	×	−39.7	−106.5	688
Full Framework (ours)	✓	✓	✓	**−28.2**	**−81.3**	**655**

The bold text is used to highlight the best performance value in the table.

4.6 Sensitivity Analysis

We conduct three sensitivity studies to characterize the robustness of the proposed framework.

(a) Robustness experiments. We re-execute the full framework with 5 runs and record the mean ± standard deviation in Table 9. The relative standard deviation of WNS and TNS stays below 2% on designs, and in every individual run the proposed framework outperforms DREAMPlace4.0 in both timing metrics. The reported improvements are therefore robust to the stochasticity of the pipeline.

Table 9: Robustness evaluation of the proposed framework (MEAN ± STD).

Benchmark	WNS (10^3 ps)	TNS (10^5 ps)
superblue1	−31.2 ± 1.4	−559.1 ± 7.4
superblue3	−125.2 ± 0.7	−1167.3 ± 12.9
superblue4	−23.0 ± 1.0	−295.9 ± 5.3
superblue5	−52.6 ± 0.3	−988.0 ± 4.9

(b) Sensitivity to the cluster count k. We sweep $k \in \{200, 300, 400, 500, 600\}$ on superblue18 with all other settings unchanged, and report the resulting WNS, TNS, and total preprocessing time in Table 10. When k is too small each cluster aggregates too many cells and the model loses the ability to distinguish timing-critical sub-structures, leading to weaker timing improvements; when k is large spectral clustering and VGAE inference become more expensive, while individual clusters become too small to form meaningful coarse-grained nodes for the GNN, causing within-cluster congestion in the downstream placement. $k = 400$ yields the best timing quality at acceptable preprocessing cost, which is why we adopt it as the default in all reported experiments.

Table 10: Sensitivity of timing metrics and preprocessing time to the cluster count k on superblue18.

k	WNS (10^3 ps)	TNS (10^5 ps)	Iteration	Cluster Run Time (s)
200	−36.4	−113.7	746	55
300	−34.2	−98.0	98.4	87
400	**−28.2**	**−81.3**	655	125
500	−32.1	−95.6	604	225
600	−27.6	−86.9	573	279

The bold is used to highlight the best results in the table.

(c) Sensitivity to the momentum coefficient α. We sweep $\alpha \in \{0.1, 0.3, 0.5, 0.7\}$ on superblue18 with all other settings unchanged, and report the resulting WNS in Table 11. The best performance is

achieved at $\alpha = 0.3$. When α is small, the influence of weight update is limited. When α becomes too large, the candidate weights dominate the newly constructed pairwise connections, exceeding the model's generalization capability and leading to degraded performance. We therefore adopt $\alpha = 0.3$ as the default and recommend $\alpha \in [0.1, 0.5]$ in practice.

Table 11: Sensitivity of timing metrics to the momentum coefficient α.

α	WNS (10^3 ps)	TNS (10^5 ps)
0.1	−28.6	−82.0
0.3	**−28.2**	**−81.3**
0.5	−35.3	−102.5
0.7	−39.4	−165.3

The bold is used to highlight the best results in the table.

5 Conclusion

This paper presents a novel timing-driven placement framework that integrates a variational graph autoencoder (VGAE) with the analytical placer DREAMPlace. By introducing a timing-aware initialization strategy, the proposed method learns the latent relationship between interconnect topology and spatial proximity, enabling early contraction of timing-critical paths. Unlike path-based approaches that suffer from scalability issues due to the large number of critical paths, the proposed framework encodes slack information into graph edge weights and performs timing-aware representation learning without explicit path enumeration. Experimental results on ICCAD2015 benchmarks show that the proposed method improves worst negative slack (WNS) by 25.4% and total negative slack (TNS) by 18.1% compared with DREAMPlace4.0.

Overall, the main contribution of this work lies not only in the achieved timing improvements, but also in providing a practical timing-aware initialization mechanism with strong industrial compatibility. The framework can be integrated with mainstream STA tools and timing-driven placers, requires only limited hyperparameter tuning, and improves timing quality without disrupting the downstream optimization flow. Although the proposed method does not outperform all recent timing-driven placement approaches in final timing metrics, it offers an effective and scalable solution for enhancing the starting point of downstream timing-driven optimization, and therefore provides a practical direction for improving placement quality and accelerating design closure in advanced-node VLSI systems.

Acknowledgement: The authors wish to express their gratitude to colleagues for their support.

Funding Statement: Shanghai Municipal Special Program for Promoting High-Quality Industrial Development in 2025. No. 2025-GZL-RGZN-02034.

Author Contributions: Ziyi Ju, Ping Yu, Tonglin Chen and Rui Song mainly conducted experiments and wrote this manuscript. Ping Yu and Tonglin Chen guided the method design and experiments. Tonglin Chen forward suggestions for method improvement. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support this study are openly available in https://drive.google.com/file/d/1xeauwLR9lOxnYvsK2JGPSY0INQh8VuE4/view?usp=sharing.

Ethics Approval: This study did not involve human participants, animals, or any personal or sensitive data. Ethical approval was therefore not required.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pan DZ, Halpin B, Ren H. Timing-driven placement. In: Handbook of algorithms for physical design automation. Boca Raton, FL, USA: Auerbach Publications; 2008. p. 423–46.
2. Kong T. A novel net weighting algorithm for timing-driven placement. In: Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design; 2002 Nov 10–14; San Jose, CA, USA. p. 172–6.
3. Burstein M, Youssef MN. Timing influenced layout design. In: Proceedings of the 22nd ACM/IEEE Design Automation Conference; 1985 Jun 23–26; Las Vegas, NV, USA. p. 124–30.
4. Chang H, Shragowitz E, Liu J, Youssef H, Lu B, Sutanthavibul S. Net criticality revisited: an effective method to improve timing in physical design. In: Proceedings of the 2002 International Symposium on Physical Design; 2002 Apr 7–10; San Jose, CA, USA. p. 155–60.
5. Wang TY, Tsai JL, Chen CCP. Sensitivity guided net weighting for placement driven synthesis. In: Proceedings of the 2004 International Symposium on Physical Design; ACM. New York, NY, USA; 2004. p. 124–31.
6. Liao P, Guo D, Guo Z, Liu S, Lin Y, Yu B. Dreamplace 4.0: timing-driven placement with momentum-based net weighting and lagrangian-based refinement. IEEE Trans Comput-Aided Design Integ Circuit Syst. 2023;42(10):3374–87.
7. Lin JM, Chang YY, Huang WL. Timing-driven analytical placement according to expected cell distribution range. In: Proceedings of the 2024 International Symposium on Physical Design; 2024 Mar 12–15; Taipei, Taiwan. p. 177–84.
8. Swartz W, Sechen C. Timing driven placement for large standard cell circuits. In: Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference; 1995 Jun 12–16; San Francisco, CA, USA. p. 211–5.
9. Kahng AB, Mantik S, Markov IL. Min-max placement for large-scale timing optimization. In: Proceedings of the 2002 International Symposium on Physical Design; 2002 Apr 7–10; San Diego, CA, USA. p. 143–8.
10. Chou YC, Lin YL. Effective enforcement of path-delay constraints in performance-driven placement. IEEE Trans Comput-Aided Design Integ Circuit Syst. 2002;21(1):15–22. doi:10.1109/43.974133.
11. Dutt S, Ren H, Yuan F, Suthar V. A network-flow approach to timing-driven incremental placement for ASICs. In: Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design; 2006 Nov 5–9; San Jose, CA, USA. p. 375–82.
12. Chen W, Huang H, Wei M, Zou P, Chen J. Virtual-path-based timing optimization for VLSI global placement. In: Proceedings of the 2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT); 2022 Oct 25–28; Nangjing, China. p. 1–3.
13. Shi Y, Xu S, Kai S, Lin X, Xue K, Yuan M, et al. Timing-driven global placement by efficient critical path extraction. In: Proceedings of the 2025 Design, Automation & Test in Europe Conference (DATE); 2025 Mar 31–Apr 2; Lyon, France. p. 1–7.
14. Lu YC, Pentapati S, Lim SK. VLSI placement optimization using graph neural networks. In: Proceedings of the 34th Advances in Neural Information Processing Systems (NeurIPS) Workshop on ML for Systems; 2020 Dec 6–12; Virtual. p. 6–12.
15. Poornima S, Naveen K, Manoj Kumar S. A hierarchical graph-aware learning framework for scalable and generalizable VLSI placement. In: Proceedings of the 2025 Third International Conference on Networks, Multimedia and Information Technology (NMITCON); 2025 Aug 1–2; Bengaluru, India. p. 1–6.
16. Liu Y, Ju Z, Li Z, Dong M, Zhou H, Wang J, et al. GraphPlanner: floorplanning with graph neural network. ACM Trans Design Autom Electron Syst. 2022;28(2):1–24.
17. Lim DU, Park H. Graph neural network-based detailed placement optimization framework. In: Proceedings of the 2024 25th International Symposium on Quality Electronic Design (ISQED); 2024 Apr 3–5; San Francisco, CA, USA. p. 1–6.
18. Chang K, Kim T. Pre-route timing prediction and optimization with graph neural network models. Integration. 2024;99(5):102262. doi:10.2139/ssrn.4627834.

19. Ding W, Zhang Z, He G, Cao P. A physical and timing aware placement optimization framework based on graph neural network. In: Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design; 2024 Oct 27–31; New York, NY, USA. p. 1–9.
20. Guo Z, Lin Y. Differentiable-timing-driven global placement. In: Proceedings of the 59th ACM/IEEE Design Automation Conference; 2022 Jul 10–14; San Francisco, CA, USA. p. 1315–20.
21. Huang T, Wong MDF. OpenTimer: a high-performance timing analysis tool. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2015); 2015 Nov 2–6; Austin, TX, USA. p. 895–902.
22. Lin Y, Dhar S, Li W, Ren H, Khailany B, Pan DZ. Dreamplace: deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In: Proceedings of the 56th Annual Design Automation Conference; 2019 Jun 2–6; Las Vegas, NV, USA. p. 1–6.
23. Halpin B, Chen CR, Sehgal N. Timing driven placement using physical net constraints. In: Proceedings of the 38th Annual Design Automation Conference; 2001 Jun 22; Las Vegas, NV, USA. p. 780–3.
24. Eisenmann H, Johannes FM. Generic global placement and floorplanning. In: Proceedings of the 35th Annual Design Automation Conference; 1998 Jun 15–19; San Francisco, CA, USA. p. 269–74.
25. Rajagopal K, Shaked T, Parasuram Y, Cao T, Chowdhary A, Halpin B. Timing driven force directed placement with physical net constraints. In: Proceedings of the 2003 International Symposium on Physical Design; 2003 Apr 6–9; Monterey, CA, USA. p. 60–6.
26. Guth C, Livramento V, Netto R, Fonseca R, Güntzel JL, Santos L. Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression. In: Proceedings of the 2015 Symposium on International Symposium on Physical Design; 2015 Mar 29–Apr 1; Monterey, CA, USA. p. 141–8.
27. Liao P, Liu S, Chen Z, Lv W, Lin Y, Yu B. DREAMPlace 4.0: timing-driven global placement with momentum-based net weighting. In: Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition; 2022 Mar 14–23; Antwerp, Belgium. p. 939–44.
28. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323(6088):533–6. doi:10.1038/323533a0.
29. Marquardt A, Betz V, Rose J. Timing-driven placement for FPGAs. In: Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays; 2000 Feb 10–11; Monterey, CA, USA. p. 203–13.
30. Jiang H, Guo Y, Guo S, Liu H, Li X, Wang N, et al. Critical path aware timing-driven global placement for large-scale heterogeneous FPGAs. IEEE Trans Comput Aided Des Integr Circuits Syst. 2026;99:1.
31. Chang FC, Tseng YW, Yu YW, Lee SR, Cioba A, Tseng IL, et al. Flexible chip placement via reinforcement learning: late breaking results. In: Proceedings of the 59th ACM/IEEE Design Automation Conference; 2022 Jul 10–14; San Francisco, CA, USA. p. 1392–3.
32. Tan Z, Mu Y. Hierarchical reinforcement learning for chip-macro placement in integrated circuit. Pattern Recogn Lett. 2024;179(9):108–14. doi:10.1016/j.patrec.2024.02.002.
33. Lai Y, Mu Y, Luo P. Maskplace: fast chip placement via reinforced visual representation learning. Adv Neural Inform Process Syst. 2022;35:24019–30.
34. Lu YC, Ren H, Hsiao HH, Lim SK. GAN-place: advancing open source placers to commercial-quality using generative adversarial networks and transfer learning. ACM Trans Design Autom Electron Syst. 2024;29(2):1–17.
35. Su Y, Yang F, Zeng X. AMOR: an efficient aggregating based model order reduction method for many-terminal interconnect circuits. In: Proceedings of the 49th Annual Design Automation Conference; 2012 Jun 3–7; San Francisco, CA, USA. p. 295–300.
36. Liu Y, Ju Z, Li Z, Dong M, Zhou H, Wang J, et al. Floorplanning with graph attention. In: Proceedings of the 59th ACM/IEEE Design Automation Conference; 2022 Jul 10–14; San Francisco, CA, USA. p. 1303–8.
37. Kipf TN, Welling M. Variational graph auto-encoders. arXiv:1611.07308. 2016.
38. Lin Y, Li W, Gu J, Ren H, Khailany B, Pan DZ. ABCDPlace: accelerated batch-based concurrent detailed placement on multithreaded CPUs and GPUs. IEEE Trans Comput Aided Des Integr Circuits Syst. 2020;39(12):5083–96.

39. Kim M, Hu J, Li J, Viswanathan N. ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design; 2015 Nov 2–6; Austin, TX, USA. p. 921–6.
40. Stanton I. Streaming balanced graph partitioning algorithms for random graphs. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms; 2014 Jan 5–7; Portland, OR, USA. p. 1287–301.
41. Tabatabaei SS, Coates M, Rabbat M. GANC: greedy agglomerative normalized cut for graph clustering. Pattern Recogn. 2012;45(2):831–43.
42. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. J Stat Mech Theory Exp. 2008;2008(10):P10008. doi:10.1088/1742-5468/2008/10/p10008.
43. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Scientif Comput. 1998;20(1):359–92. doi:10.1137/s1064827595287997.