DiffDEG: Diffusion-Enhanced Design Evolution Graph Representation Learning for Post-Layout Optimization

Jiajie Xu[1,3], Leilei Jin[2], Ziyue Han[1,3], Yanlong Mao[1,3], Liangji Wu[4], Chenpu Shi[1,3], Yunfan Zuo[1,3], Lizheng Ren[1,3], Hao Yan[1,3,*], Xingquan Li[1,3], and Longxing Shi[1,3]

[1]Southeast University [2]CUHK [3]National Technology Innovation Center for EDA [4]Fudan University

Abstract

Post-layout optimization is a critical step in modern chip design. However, existing Machine Learning (ML)-assisted methods struggle to capture the cross-stage reasoning dependencies that underlie the optimization challenges. While large language models (LLMs) excel at semantic reasoning, the lack of structural and physical awareness limits their effectiveness in post-layout optimization. To address these limitations, we propose DiffDEG, a diffusion-enhanced, reasoning-aware foundation model that bridges LLM-based semantic reasoning with circuit-level structural and physical representations. DiffDEG reformulates the conventional timing graph into a Design Evolution Graph (DEG), enabling cross-stage reasoning through text-annotated netlists. By leveraging directional diffusion and self-supervised pretraining, DiffDEG jointly interprets semantic, timing, and physical information, forming a unified representation adaptable to diverse optimization tasks. Experimental results show that DiffDEG consistently enhances optimization outcomes across multiple optimization paradigms, achieving an average 12.5% performance improvement and 4× speedup over commercial tools.

1 Introduction

Engineering Change Orders (ECO) refer to post-layout optimizations that serve as a critical step before chip signoff. However, as physical effects grow more intricate at advanced technology nodes, inaccurate evaluation of layout impacts during ECO can lead to suboptimal optimizations, ultimately hindering the design from achieving its intended performance. To balance optimization quality with linear computational complexity, commercial tools typically rely on physical-aware ECO flows that iteratively interact with placement and routing (P&R) tools to achieve convergence [1]. Nonetheless, such repeated re-placement and re-routing significantly increase runtime overhead, making efficiency a critical concern.

In contrast to the heuristic strategies commonly employed in commercial tools, recent research efforts have increasingly explored Machine Learning (ML) techniques to achieve more efficient ECO optimization [2–10]. These methods typically employ diverse encoder architectures to transform circuit netlists and layouts into learnable embeddings. The resulting embeddings are then utilized either for predictive acceleration [2–4] or integrated with learning-based frameworks such as reinforcement learning (RL) [5, 8, 10] and gradient-based methods [6, 7, 9] to achieve superior performance.

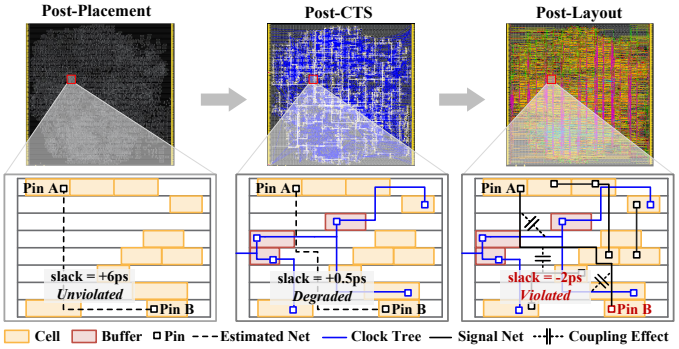

Figure 1: Interdependence exists across physical design stages. Timing violations in later stages typically accumulate from early design stages.

Nevertheless, the challenges encountered during the ECO stage are intrinsically linked to the progressive nature of the physical design process. As illustrated in Figure 1, *Pin B* initially exhibits a sufficient timing margin after placement. This margin diminishes during clock-tree synthesis (CTS) due to buffer-induced density increases and further degrades after routing as coupling effects intensify, ultimately resulting in a timing violation. A comprehensive understanding of this progression enables the optimization engine to more accurately identify and optimize critical regions. However, such cross-stage reasoning dependencies remain difficult for existing ML-based models to capture, thereby limiting their capability to address post-layout optimization problems.

Recently, large language models (LLMs) have demonstrated remarkable capabilities in capturing temporal and reasoning relationships [11, 12]. Nevertheless, LLMs are primarily optimized for learning semantic regularities in textual data, whereas circuit implementation is characterized by structured dependencies and spatial constraints, leading to a substantial modality mismatch. This mismatch impedes the direct transfer of LLM-style reasoning to ECO optimization, where effective reasoning requires the joint analysis of layout features and netlist structural information.

In this work, we propose **DiffDEG**, a diffusion-enhanced reasoning-aware foundation model tailored for ECO optimization. DiffDEG bridges the semantic reasoning capability of LLMs with netlist structural representations and layout physical context. Unlike previous ECO-specific models, DiffDEG operates on a **Design Evolution Graph (DEG)**, a cross-modal circuit linkage graph that fuses pin-level textual embeddings with physical design information to form a unified representation. Through self-supervised pre-training, the LLM-derived embeddings are refined to acquire physical significance, enabling more accurate and

*Corresponding author.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
DAC '26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804019

interpretable circuit representations. These representations can be seamlessly adapted to diverse downstream ECO optimization tasks, allowing ML models to achieve faster and more effective post-layout optimization.

The main contributions are summarized as follows:

- **Cross-stage reasoning–aware text annotation:** We reformulate the timing graph as a DEG and annotate cell pins with multi-stage physical information to capture cross-stage dependencies and optimization history.
- **Directional graph diffusion–enhanced embeddings:** A fine-tuned LLM encoder (ReasonLLM) embeds textual reasoning into the DEG, while directional graph diffusion injects topology-aligned anisotropic noise to unify semantic and structural information.
- **Physical knowledge fusion foundation:** A graph transformer (DEGFormer) further incorporates physical knowledge, and self-supervised pretraining yields a unified representation with strong cross-task generalization.
- **Comprehensive evaluation across ECO paradigms:** DiffDEG consistently improves optimization quality across prediction-, RL-, and gradient-based frameworks, achieving 12.5% performance gain and 4× speedup over commercial tools.

2 Preliminary

2.1 Post-layout Optimization

Achieving fast and accurate timing and power optimization at the post-layout stage has long been a critical objective in physical design closure. Existing methods can be broadly classified into two categories. Analytical approaches rely on sensitivity analysis [13], convex optimization [14], and Lagrangian relaxation (LR) formulations [15]. ML–based approaches employ supervised learning for faster convergence [2–4, 16], RL for higher optimization quality [5, 8, 10], and gradient-based models for balancing efficiency and accuracy [6, 7, 9]. Compared with analytical methods, ML techniques exhibit stronger generalization, better scalability to complex physical effects, and improved modeling of nonlinear correlations. However, their optimization effectiveness critically depends on efficient and informative circuit representations.

2.2 Circuit Representation Learning

Circuit representation learning has recently become a pivotal research direction for enabling ML-driven analysis and optimization of electronic circuits [17]. Existing approaches are primarily based on synthesized netlists. They either aim to capture circuit behavior for early-stage functional verification [18–22], or model performance differences before and after physical design [23, 24]. However, these representations exhibit significant limitations when applied to post-layout optimization. On one hand, circuit logic functionality is not strongly correlated with the bottlenecks of performance optimization. On the other hand, post-layout optimization requires cross-stage reasoning capabilities to interpret the evolutionary dependencies across design stages. Consequently, these methods demonstrate limited applicability in ECO optimization.

2.3 Problem Formulation

To address the aforementioned limitations, we formally define the problem as follows:

Problem 1 (Post-layout Optimization Oriented Circuit Representation Learning). *Given a post-layout circuit design and its corresponding physical design database, the objective is to learn a circuit representation that can be effectively utilized by ML-based post-layout optimization algorithms.*

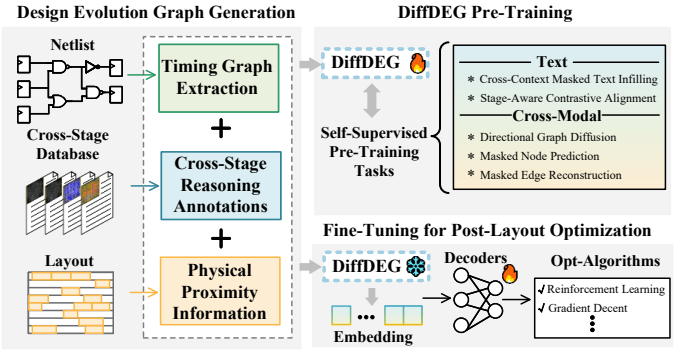

Figure 2: DiffDEG flow overview.

3 Methodology

3.1 Overview

Figure 2 provides an overview of the DiffDEG workflow. Post-layout netlist timing graphs are augmented with cross-stage reasoning annotations and physical proximity information to construct the DEG, as detailed in Section 3.2. Each DEG is subsequently partitioned into logic cones, where the textual components are encoded by a fine-tuned LLM and further enhanced through directional graph diffusion to strengthen the structural context. The resulting textual embeddings are then fused with timing and physical features via cross-modal pretraining (Section 3.6). This process produces unified DiffDEG embeddings that jointly capture semantic, structural, and physical knowledge within a single representation. These pretrained embeddings provide a general foundation for a broad range of downstream post-layout optimization tasks.

3.2 Design Evolution Graph Generation

The detailed process of DEG generation is illustrated in Figure 3. For each pin, we generate a textual attribute t_i that records its evolutionary trajectory across the physical design stages (shaded region in Figure 3). Each t_i includes timing states, spatial coordinates, and cell power variations, together with a lineage entry that traces optimization operations applied during implementation. For example, during place optimization (PlaceOpt), cell *U221* undergoes both gate sizing and threshold-voltage reassignment; its lineage records the parent pin, cell type, and corresponding optimization actions. Pin correspondence across design stages is first established through Weisfeiler–Lehman graph hashing based on pin naming. For unmatched cases, similarity between fan-in/fan-out neighborhoods is computed using a MinHash-based local subgraph comparison, and the closest predecessor is selected, with the *conf* field indicating the confidence score. Pins without valid matches are assigned *None*. Cross-stage annotation is performed on netlists extracted from four representative design stages: placement, PlaceOpt, CTS, and routing. Collectively, the textual attributes form $\mathcal{T} = t_1, \ldots, t_m$.

Building on these textual attributes, we construct the structural backbone of the DEG using the circuit timing graph, which serves as the analytical basis for performance evaluation and is widely adopted in post-layout optimization [9, 25, 26]. We first refine the graph structure by removing constraint-related arcs and redundant multi-edges, ensuring unique timing dependencies between pin pairs (green dashed box in Figure 3). To incorporate physical context, virtual edges are added

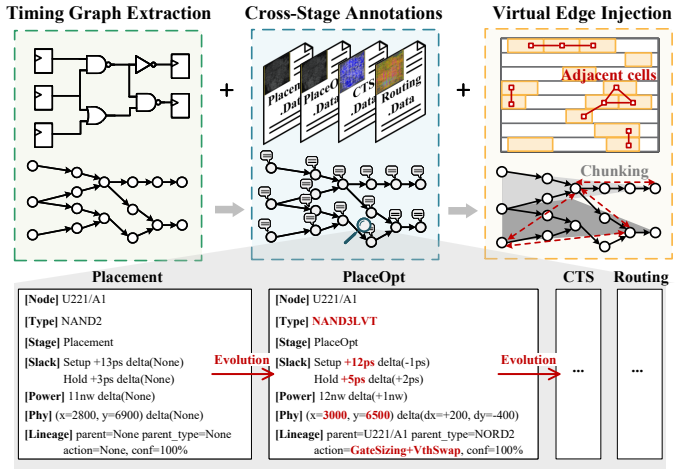

Figure 3: The Generation of Design Envolution Graph.

between output pins of spatially adjacent cells (yellow dashed box). To maintain scalability for large circuits, we apply a logic-cone chunking strategy while preserving one-hop virtual-edge connectivity. The resulting representation is a text-attributed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{T}, \mathcal{E}_T, \mathcal{E}_L\}$, where $\mathcal{V}$ denotes pins, $\mathcal{T}$ their textual attributes, $\mathcal{E}_T$ timing relations, and $\mathcal{E}_L$ layout-proximity relations.

3.3 DiffDEG Model Architecture

The overall architecture of DiffDEG is illustrated in Figure 4. The framework consists of three tightly integrated modules: ❶ a ReasonLLM that generates textual reasoning embeddings, ❷ a DiffGUNet structural encoder that injects structural awareness into the LLM-derived representations, and ❸ a DEGFormer that integrates physical knowledge. The architecture of DiffDEG is detailed below.

❶ ReasonLLM of DiffDEG. The ReasonLLM module is initialized from an LLM-based text embedding model to encode pin-level textual attributes into initial node embeddings. Unlike conventional encoder–decoder architectures (e.g., BERT [27], T5 [28]), decoder-only LLMs demonstrate superior text encoding capability owing to their rich knowledge priors and improved sample efficiency [29]. Accordingly, ReasonLLM adopts a bidirectional attention–based LLM backbone, which maps the textual attributes of pins t_i to their corresponding embeddings $\mathbf{Z}_{\text{LLM}}$ as:

$$\mathbf{Z}_{\text{LLM}} = \text{ReasonLLM}(t_i). \tag{1}$$

❷ DiffGUNet of DiffDEG. While LLMs are effective at modeling sequential linguistic dependencies, circuit optimization requires reasoning over hierarchical and topological structures. To bridge this gap, we introduce a *directional graph diffusion framework* that refines LLM-derived embeddings and injects graph-aware structural awareness. In contrast to isotropic diffusion, which applies uniform Gaussian perturbations and may obscure directional relations, directional diffusion introduces data-dependent anisotropic noise aligned with the topology of the DEG [30]. During progressive denoising, the model reconstructs node representations while preserving directional coherence and relational hierarchy. For denoising, we adopt a UNet-style GAT (GUNet) architecture with skip connections to propagate encoder features to their corresponding decoder layers:

$$\mathbf{H}_{\text{dec}}^{(m)} = \sigma\Big(\text{GAT}\Big(\Big[\mathbf{H}_{\text{dec}}^{(m-1)} \,\Big\|\, \mathbf{W}_{\text{skip}}^{(m)} \mathbf{H}_{\text{enc}}^{(s(m))}\Big], \mathbf{A}\Big)\Big), \tag{2}$$

where $\mathbf{H}_{\text{dec}}^{(m)}$ denotes the m-th decoder-layer representation, $\mathbf{H}_{\text{enc}}^{(s(m))}$ is the corresponding encoder feature passed through the skip connection, $\mathbf{W}_{\text{skip}}^{(m)}$ projects feature dimensions, $\mathbf{A}$ is the adjacency matrix, $[\cdot\|\cdot]$ denotes concatenation, and $\sigma(\cdot)$ is the activation function. The structural-aware textual embeddings are derived from the decoder by concatenating the activations of its hidden layers:

$$\mathbf{Z}_{\text{Diff}} = \text{Concat}\left(\mathbf{H}_{\text{dec}}^{(1)}, \mathbf{H}_{\text{dec}}^{(2)}, \ldots, \mathbf{H}_{\text{dec}}^{(L_d)}\right), \tag{3}$$

where $\mathbf{Z}_{\text{Diff}}$ denotes the aggregated representation formed by hierarchical decoder features, and L_d is the number of decoder layers.

❸ DEGFormer of DiffDEG. The DEGFormer serves as a graph transformer that integrates textual embeddings with the physical knowledge of the circuit netlist. To efficiently capture global dependencies, we adopt the linear-complexity attention formulation from SGFormer [31]:

$$\text{LinearAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \phi(\mathbf{Q})\big(\phi(\mathbf{K})^\top \mathbf{V}\big) \,\Big/\, \phi(\mathbf{Q})\big(\phi(\mathbf{K})^\top \mathbf{1}\big), \tag{4}$$

where $\phi(\cdot)$ is a kernel mapping function that projects queries and keys into a high-dimensional feature space, allowing attention computation in linear time with respect to the number of nodes.

The final design-level representation is obtained as

$$\mathbf{Z}_{\text{DEG}} = \text{DEGFormer}\left(\text{Concat}\left(\mathbf{Z}_{\text{LLM}}, \mathbf{Z}_{\text{Diff}}\right), \mathbf{X}_{\text{cir}}, \mathcal{E}_T, \mathcal{E}_L\right), \tag{5}$$

where $\mathbf{X}_{\text{cir}}$ denotes circuit-level features, including physical attributes (e.g., *slack*, *arrival time*, *cell leakage*) and layout metrics (e.g., *spatial coordinates* (x, y), *congestion*, *density*). Library-aware features are extracted via an auxiliary encoder [32], and for each pin, mean-pooled over its associated timing arcs. The resulting features are passed through a Multilayer Perceptron (MLP) and concatenated with the textual embeddings before being input to the graph transformer.

3.4 Pre-Training Step 1: Enhancing Cross Stage Reasoning in LLM

To equip the LLM with cross-stage dependency awareness, we employ the following self-supervised pretraining tasks:

Task #1.1 Cross-Stage Text Infilling. In this task, key textual segments from one design stage are randomly masked, and the model is trained to reconstruct them using contextual information from other correlated stages. This objective encourages cross-stage reasoning, enabling the model to infer omitted design context based on inter-stage relationships. Formally, given a design text sample t drawn from the dataset $\mathcal{T}$, let $\mathcal{S}(t)$ denote the set of key segments extracted from t, and $s \sim \mathcal{S}(t)$ represent a randomly selected segment to be masked. The learning objective uses a cross-entropy reconstruction loss, defined as:

$$\mathcal{L}_{\text{Text}}^{1} = -\mathbb{E}_{t\sim\mathcal{T}}\mathbb{E}_{s\sim\mathcal{S}(t)} \log P_\theta\left(t^{(s)} \mid t^{(\neg s)}\right), \tag{6}$$

where $t^{(s)}$ denotes the masked segment and $t^{(\neg s)}$ represents all remaining visible segments that provide contextual information. This pretraining objective applies only to nodes whose upstream stages have valid predecessors.

Task #1.2 Cross-Stage Contrastive Learning. While the previous task focuses on reconstructive reasoning, this task aims to learn discriminative and stage-consistent representations. Each pin text encodes the evolving state of a specific pin across different stages of the physical design process, making it distinguishable from texts of other pins. To achieve this, texts corresponding to the same pin at different stages are treated as positive pairs, while texts from different pins serve as negative samples. The LLM is pretrained with a contrastive objective

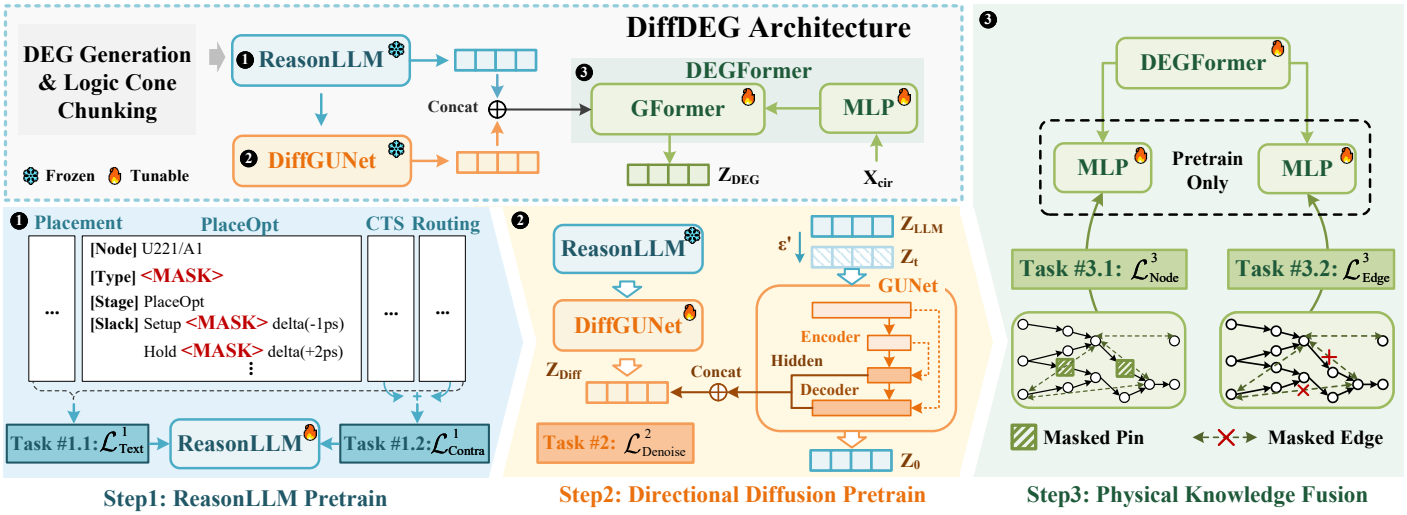

Figure 4: Illustration of DiffDEG architecture and pre-training workflow.

based on the InfoNCE loss. Let z_i and z_i^+ denote embeddings of the same pin at different stages, and z_i^- denote embeddings of negative samples, i.e., pin texts from other pins within the batch. The loss is defined as:

$$\mathcal{L}^1_{\text{Contra}} = -\log \frac{\exp\big(\text{sim}(z_i, z_i^+)/\tau\big)}{\sum_{j=0}^{K} \exp\big(\text{sim}(z_i, z_j)/\tau\big)}, \tag{7}$$

where $\text{sim}(\cdot,\cdot)$ denotes cosine similarity and τ is the temperature parameter, with j indexing the contrastive set $\{z_i^+\} \cup \{z_i^-\}$.

3.5 Pre-Training Step 2: Enhancing LLM Embeddings with Directional Diffusion

We adopt graph-level self-supervised objectives to fuse semantic-rich pin embeddings from pre-trained and frozen ReasonLLM with global circuit structure. The pre-training task is illustrated as follow:

Task #2 Directional Diffusion. We follow the standard diffusion modeling paradigm, where a denoising network f_θ is trained to recover the clean node embeddings $\mathbf{Z}_0$ from their noisy counterparts $\mathbf{Z}_t$. The forward diffusion process constructs the noisy graph representation $G_t = (\mathbf{A}, \mathbf{Z}_t)$ by progressively perturbing $\mathbf{Z}_0$ as:

$$\mathbf{Z}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{Z}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon', \tag{8}$$

where $\bar{\alpha}_t$ is the cumulative noise schedule and ϵ' denotes the directional noise. The denoising network is optimized via:

$$\mathcal{L}^2_{\text{Denoise}} = \mathbb{E}_{\mathbf{Z}_0,t} = ||f_\theta(\mathbf{Z}_t, \mathbf{A}, t) - \mathbf{Z}_0||^2. \tag{9}$$

Unlike isotropic Gaussian noise used in conventional diffusion, we introduce directional noise to preserve structural consistency within the DEG. Specifically, we align the perturbation with the semantic orientation of the clean features:

$$\overline{\epsilon} = \mu + \sigma \odot \epsilon, \qquad \epsilon' = \text{sgn}(z_{0,i}) \odot |\overline{\epsilon}|, \tag{10}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, $\odot$ denotes the Hadamard product, and $\text{sgn}(\cdot)$ enforces directional alignment. This modification ensures that noise respects the relational geometry of the graph rather than disrupting it. Through reverse diffusion, the model reconstructs node embeddings in a way that preserves hierarchical and topological dependencies, yielding structure-aware representations.

3.6 Pre-Training Step 3: Physical Knowledge Fusion for Better Optimization

After obtaining comprehensive textual embeddings, a set of physical design–aware self-supervised tasks is designed to seamlessly integrate textual, timing, and physical information, resulting in final embeddings with explicit physical interpretability.

Task #3.1 Masked Node Prediction. In this task, a subset of circuit-level attributes for certain nodes is randomly masked, and DEGFormer is trained to infer the missing physical information from the remaining textual and structural context. This task strengthens the model to correlate semantic representations derived from text with corresponding physical meanings. The task is formulated as a regression problem. Let $\mathbf{y}_i^{\text{phys}}$ denote the ground-truth physical features of node i. An auxiliary regressor MLP_{regr} takes the masked node embedding Z_i^{mask} as input and predicts the masked physical attributes. The objective function is:

$$\mathcal{L}^3_{\text{Node}} = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{y}_i^{\text{phys}} - \text{MLP}_{\text{regr}}(Z_i^{mask})||_2^2, \tag{11}$$

where the mean squared error encourages accurate reconstruction of masked physical features, aligning text-derived representations with underlying circuit-level physical semantics.

Task #3.2 Masked Edge Reconstruction. This task trains DEGFormer to infer physical proximity by reconstructing masked edges from contextual embeddings. A subset of physical edges is randomly masked, and the model is required to recover the missing connections based on node features. The task is framed as a contrastive learning problem, where the true edge endpoint is the positive sample and non-adjacent neighborhood nodes are treated as negatives. Through this task, the model learns to encode layout-level neighborhood relations and capture spatial coherence in the circuit netlist. The loss function is defined as:

$$\mathcal{L}^3_{\text{Edge}} = -\log \frac{\exp\big(\text{sim}(Z_i, Z_i^+)/\tau\big)}{\sum_{j=0}^{K} \exp\big(\text{sim}(Z_i, Z_j)/\tau\big)}, \tag{12}$$

where Z_i^+ denotes the embedding of a physically adjacent node, and Z_j those of contrastive candidates including non-adjacent nodes.

The overall self-supervised pre-training objective for DiffDEG is then formulated as follows:

$$\mathcal{L}_{\text{DiffDEG}} = \begin{cases} \mathcal{L}^1_{\text{Text}}, \mathcal{L}^1_{\text{Contra}}, & \text{(Step1)} \\ \mathcal{L}^2_{\text{Denoise}}, & \text{(Step2)} \\ \mathcal{L}^3_{\text{Node}} + \mathcal{L}^3_{\text{Edge}}. & \text{(Step3)} \end{cases} \tag{13}$$

The pre-trained DiffDEG produces structurally grounded embeddings with cross-stage reasoning ability and robust physical interpretability, which can be effectively transferred to a wide range of post-layout optimization tasks.

4 Experimental Results

4.1 Experimental Setup

For data preparation, we compile a diverse set of open-source circuit designs from the IWLS benchmarks [33], OpenCores [34], and Chipyard [35]. All RTL designs are synthesized using Synopsys Design Compiler [36] with a commercial 12nm technology library. Physical design is performed in Cadence Innovus [37], and design quality is analyzed using Synopsys PrimeTime [1]. The statistical details of the dataset are provided in Table 1.

Table 1: Benchmark design information.

Benchmarks	#Designs	Design Size Range		
		#K Cells	#K Pins	#K EndPoints
IWLS	12	2-56	7-203	1-20
OpenCores	4	14-237	47-827	4-79
Chipyard	3	193-971	672-2810	2-138

For implementation, DiffDEG is developed in Python 3.12 and trained with PyTorch 2.4.1. The ReasonLLM module is initialized from LLM2Vec [29] based on Meta Llama-3.1-8B [38]. The output dimension of NetTAG is set to 768. ReasonLLM is pretrained using LoRA [39] for one epoch per task in Step 1. The GUNet denoiser is trained for 64 epochs with 520 diffusion steps, and the latent representation at the 100th diffusion step is used as the encoder output. The DEGFormer component is pretrained for 50 epochs. All experiments are conducted on a Linux machine equipped with two AMD EPYC 7B13 CPUs, 768GB of memory, two NVIDIA GeForce RTX4090 GPUs, and two NVIDIA RTX6000 Ada Generation GPUs.

4.2 Downstream Tasks Performance Evaluation

We evaluate the pre-trained DiffDEG on three representative post-layout optimization objectives, spanning from performance estimation to optimization policy enhancement, thereby providing a comprehensive assessment of its representational effectiveness. Without loss of generality, we focus on timing improvement and adopt gate sizing as the representative optimization operation.

Objective 1: Optimization result estimation. This objective assesses the ability to accurately estimate post-optimization circuit performance. Fast performance estimation can serve as a surrogate model or provide differentiable guidance within optimization loops. Since layout perturbations significantly influence parasitic effects, we evaluate the timing quality by examining per-pin slack variations. As shown in Table 2, DiffDEG surpasses both the GNN-only baseline [6] and the multimodal approach [25], achieving an average R^2 of 0.94 and a mean absolute error (MAE) of 7.22 ps. These results demonstrate that DiffDEG provides accurate and efficient timing estimation suitable for integration into optimization frameworks.

Objective 2: Reinforcement learning acceleration. We evaluate the benefit of DiffDEG embeddings in RL-based post-layout optimization, adopting the framework in DAC21 [5] and the Soft Actor–Critic algorithm [40] to ensure stable convergence. As shown in Figure 5, the RL agent built upon DiffDEG embeddings achieves substantially faster convergence and higher solution quality than the heuristic feature-based RL-GNN from DAC21. Specifically, RL-DiffDEG attains 1.75× and 2.78× faster convergence on two benchmark designs, and improves total negative slack (TNS) by 11% on Design 10. These results indicate that DiffDEG provides more expressive and useful state representations for RL-driven optimization.

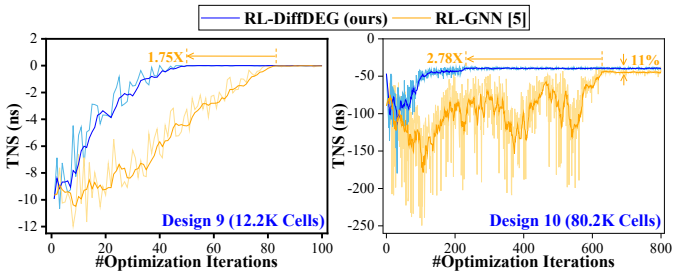

Figure 5: RL optimization result comparison.

Table 2: Post-optimization timing results prediction accuracy comparison. The unit of "MAE" is *ps*.

Unseen designs (#Cells)	DAC23 [6]		TCAD24 [7]		**DiffDEG (ours)**	
	MAE	R^2	MAE	R^2	MAE	R^2
Design1 (5.5k)	19.13	0.73	15.03	0.83	**6.99**	**0.92**
Design2 (8.9k)	28.78	0.78	23.68	0.79	**9.72**	**0.95**
Design3 (14.4k)	19.20	0.70	14.21	0.83	**5.44**	**0.94**
Design4 (11.8k)	32.43	0.70	13.92	0.85	**8.46**	**0.94**
Design5 (17.7k)	22.49	0.78	18.56	0.85	**6.76**	**0.93**
Design6 (56.0k)	11.37	0.77	9.39	0.84	**4.21**	**0.96**
Design7 (237.8k)	22.72	0.78	19.73	0.80	**8.32**	**0.94**
Design8 (971.9k)	40.43	0.72	27.15	0.85	**7.87**	**0.96**
Average	24.57	0.75	17.71	0.83	**7.22**	**0.94**

Table 3: Optimization outcome prediction accuracy (F1 score).

Unseen designs	ICCAD22 [2]	TODAES23 [4]	ISPD25 [9]	**DiffDEG**
Design1	0.636	0.558	0.857	**0.901**
Design2	0.742	0.676	0.870	**0.967**
Design3	0.646	0.517	0.825	**0.916**
Design4	0.687	0.538	0.882	**0.919**
Design5	0.666	0.583	0.843	**0.897**
Design6	0.675	0.657	0.872	**0.939**
Design7	0.759	0.698	0.831	**0.882**
Design8	0.616	0.521	0.793	**0.873**
Average	0.675	0.586	0.841	**0.904**

Objective 3: Optimization solution prediction. This objective aims to directly predict the optimization strategy that would be produced by commercial physical-aware optimization flows, thereby providing high-quality initial solutions and reducing the number of required optimization iterations. Golden reference strategies are obtained by executing the commercial timing and physical design flow [1, 37] until convergence. DiffDEG is compared with models adapted from ICCAD22 [2], TODAES23 [4], and ISPD25 [9], using the F1 score to evaluate prediction quality. As shown in Table 3, DiffDEG achieves an F1 score of 0.904 on unseen designs, surpassing ICCAD22 (0.675), TODAES23 (0.586), and ISPD25 (0.841), demonstrating strong generalization in optimization solution prediction.

Extension: Final timing optimization results comparison. Building on the strategy probability distribution predicted in Objective 3, we

Table 4: Optimization result comparison. "WNS" and "NVE" denote the Worst Negative Slack and the Number of Violated Endpoints, respectively. The unit of "TNS" and "WNS" are *ns*. The red percentages indicate the improvement over the commercial tool.

Unseen designs	Before Optimization			Tool Reference [1]			Gradient-Only [26]			GFormer + Gradient [9]			**DiffDEG + Gradient (ours)**		
	WNS	TNS	NVE	WNS	TNS	NVE	WNS	TNS	NVE	WNS	TNS	NVE	WNS	TNS	NVE
Design1	-0.080	-20.457	594	-0.035	-0.522	87	-0.036	-0.570	92	-0.036	-0.502	87	-0.035	**-0.388** (-25.6%)	87
Design2	-0.136	-21.736	792	-0.024	-0.155	59	-0.025	-0.222	71	-0.024	-0.144	48	-0.024	**-0.122** (-21.3%)	44
Design3	-0.099	-12.947	436	-0.118	-5.845	212	-0.116	-5.547	195	-0.112	-5.607	198	-0.108	**-5.248** (-10.2%)	171
Design4	-0.169	-27.165	348	-0.014	-0.107	50	-0.011	-0.096	32	-0.014	-0.107	49	-0.015	-0.107 (-0%)	49
Design5	-0.111	-7.835	249	-0.108	-5.807	199	-0.106	-5.846	204	-0.088	-5.235	92	-0.081	**-4.874** (-16.1%)	87
Design6	-0.059	-4.136	198	-0.033	-0.279	53	-0.122	-171.547	7460	-0.398	-0.820	110	-0.032	**-0.250** (-10.4%)	40
Design7	-0.117	-69.194	2778	-0.074	-53.981	2379	-0.079	-60.712	2396	-0.076	-53.375	2259	-0.074	**-52.226** (-3.3%)	2093
Design8	-1.861	-204.802	252	-1.763	-128.788	224	-1.571	-123.468	236	-1.517	-119.218	227	-1.492	**-111.748** (-13.2%)	222

Table 5: Runtime comparison (seconds).

Benchmarks	EDA Tool	**DiffDEG (Avg.)**			
	Phy. ECO*	Data†	Infer		Total
			LLM	Other	
IWLS	1221	70	281	5	356
OpenCores	2930	159	633	8	800
Chipyard	7348	332	851	11	1194

* Physical-aware ECO iteratively coupling timing analysis [1] with P&R [37].
† Preprocessing and data extraction for model input.

employ DiffDEG to guide timing-gradient-based optimization. We integrate the probabilistic strategy formulation from ISPD25 [9] with the differentiable timing engine [26], forming the unified Optimization framework. As shown in Table 4, our framework consistently achieves the best timing improvement. Unlike gradient-only optimization [26], which may cause severe timing degradation after subsequent P&R (Design6), DiffDEG benefits from learned cross-stage physical reasoning and avoids such regressions. Compared with [9], DiffDEG leverages deeper cross-stage knowledge fusion, enabling more accurate modeling of layout-dependent parasitics and improved initialization quality. On average, our framework achieves a 12.5% improvement in TNS over commercial tools, demonstrating strong practical effectiveness.

4.3 Runtime Analysis

As summarized in Table 5, we report the runtime of DiffDEG across multiple benchmarks. The iterative re-placement and re-routing steps in physical-aware ECO flows are extremely time-consuming. In contrast, DiffDEG achieves approximately a 4× speedup over commercial tools. Most of the runtime is spent on data processing and ReasonLLM inference. This suggests substantial potential for acceleration, as pin-level text attribution is highly parallelizable and LLM inference can be further reduced by scaling up GPU resources.

4.4 Ablation Study

Figure 6 presents the ablation study of DiffDEG. Removing the DEG and using only current-stage features results in a significant performance drop across all objectives, confirming that cross-stage knowledge is critical for effective post-layout optimization. Eliminating Task #1.1 and Task #1.2 leads to a notable reduction in optimization quality prediction accuracy, demonstrating their role in enabling the LLM to capture design evolution and stage-level dependencies. Task #2 has a pronounced effect on optimization outcome prediction, highlighting the importance of structural understanding in modeling post-layout optimization behavior. Task #3.1 and Task #3.2 substantially influence both prediction accuracy and RL convergence efficiency, indicating that physical knowledge fusion is essential for aligning semantic representations with physically

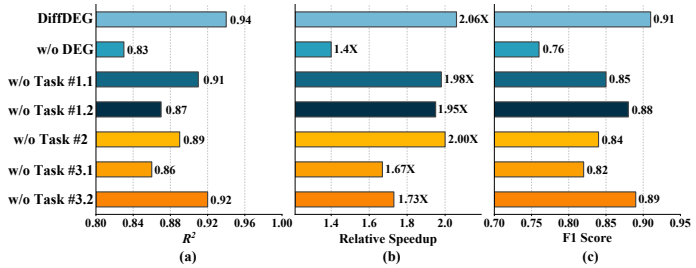

Figure 6: Ablation studies on (a) Objective 1, (b) Objective 2, and (c) Objective 3.

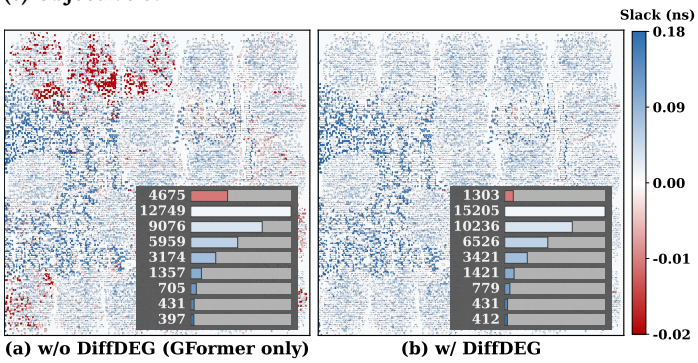

Figure 7: Visualization of pin slack optimization result on Design4, with slack statistics shown in the lower right.

meaningful circuit characteristics. Finally, Figure 7 visualizes the optimization results obtained using only GFormer-based representations compared with those enhanced by DiffDEG. The number of pins with negative slack decreases from 4675 to 1303, demonstrating the effectiveness of DiffDEG in improving post-layout optimization outcomes.

5 Conclusion

In this paper, we present DiffDEG, a cross-stage reasoning–aware framework for post-layout optimization. By reformulating the conventional timing graph into a Design Evolution Graph, DiffDEG effectively fuses textual reasoning with physical design information through self-supervised pre-training. Future work will explore extending DiffDEG to additional ECO targets (e.g., power) and assess its applicability across other physical design stages.

Acknowledgments

This work was supported by the NSFC (62474038, 62274034), the Key R&D Program of Jiangsu Province (BE2023003-2), the NSF of Jiangsu Province (BK20232005).

References

[1] Synopsys, "Primetime User Guide", https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html, 2024.

[2] S. Nath *et al.*, "Transsizer: A novel transformer-based fast gate sizer," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[3] C.-K. Cheng *et al.*, "Dagsizer: A directed graph convolutional network approach to discrete gate sizing of vlsi graphs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 4, pp. 1–31, 2023.

[4] Y.-C. Lu *et al.*, "Eco-gnn: Signoff power prediction using graph neural networks with subgraph approximation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 4, pp. 1–22, 2023.

[5] Y.-C. Lu *et al.*, "Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 733–738.

[6] P. Pham *et al.*, "Agd: A learning-based optimization framework for eda and its application to gate sizing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[7] Y. Ye *et al.*, "Learning-driven physically-aware large-scale circuit gate sizing," *arXiv preprint arXiv:2403.08193*, 2024.

[8] W. Jiang *et al.*, "Ir-aware eco timing optimization using reinforcement learning," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024, pp. 1–7.

[9] Y.-C. Lu *et al.*, "Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes," in *Proceedings of the 2025 International Symposium on Physical Design*, 2025, pp. 152–162.

[10] J. Xu *et al.*, "Towards efficient eco automation: Timing and drcs concurrent optimization using reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2026.

[11] Z. Jin *et al.*, "Cladder: Assessing causal reasoning in language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 31 038–31 065, 2023.

[12] H. Chi *et al.*, "Unveiling causal reasoning in large language models: Reality or mirage?" *Advances in Neural Information Processing Systems*, vol. 37, pp. 96 640–96 670, 2024.

[13] J. Hu *et al.*, "Sensitivity-guided metaheuristics for accurate discrete gate sizing," in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 233–239.

[14] K. Kasamsetty *et al.*, "A new class of convex functions for delay modeling and its application to the transistor sizing problem [cmos gates]," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, pp. 779–788, 2000.

[15] G. Flach *et al.*, "Effective method for simultaneous gate sizing and v th assignment using lagrangian relaxation," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 33, no. 4, pp. 546–557, 2014.

[16] X. Zhou *et al.*, "Heterogeneous graph neural network-based imitation learning for gate sizing acceleration," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[17] L. Chen *et al.*, "Large circuit models: opportunities and challenges," *Science China Information Sciences*, vol. 67, no. 10, p. 200402, 2024.

[18] M. Li *et al.*, "Deepgate: Learning neural representations of logic gates," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 667–672.

[19] Z. Shi *et al.*, "Deepgate2: Functionality-aware circuit representation learning," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[20] Z. Shi *et al.*, "Deepgate3: Towards scalable circuit representation learning," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[21] Z. Wang *et al.*, "Functionality matters in netlist representation learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 61–66.

[22] C. Deng *et al.*, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[23] W. Fang *et al.*, "Nettag: A multimodal rtl-and-layout-aligned netlist foundation model via text-attributed graph," *arXiv preprint arXiv:2504.09260*, 2025.

[24] Y. Zhao *et al.*, "Deeplayout: Learning neural representations of circuit placement layout," in *Forty-second International Conference on Machine Learning*.

[25] A. Stefanidis *et al.*, "Autonomous application of netlist transformations inside lagrangian relaxation-based optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 8, pp. 1672–1686, 2020.

[26] Y.-C. Lu *et al.*, "Insta: An ultra-fast, differentiable, statistical static timing analysis engine for industrial physical design applications," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.

[27] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[28] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[29] P. BehnamGhader *et al.*, "Llm2vec: Large language models are secretly powerful text encoders," *arXiv preprint arXiv:2404.05961*, 2024.

[30] R. Yang *et al.*, "Directional diffusion models for graph representation learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 32 720–32 731, 2023.

[31] Q. Wu *et al.*, "Sgformer: Simplifying and empowering transformers for large-graph representations," *Advances in Neural Information Processing Systems*, vol. 36, pp. 64 753–64 773, 2023.

[32] J. Xu *et al.*, "FACT: Fast and accurate multi-corner predictor for timing closure in commercial EDA flows," *ACM Transactions on Design Automation of Electronic Systems*, vol. 31, no. 4, mar 2026. [Online]. Available: https://doi.org/10.1145/3768166

[33] C. Albrecht *et al.*, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS)*, vol. 9, 2005.

[34] OpenCores: The reference community for Free and Open Source gateware IP cores, https://opencores.org/.

[35] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *Ieee Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[36] Synopsys,"Design Compiler User Guide", https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, 2024.

[37] Cadence, "Innovus Reference Manual", https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html, 2021.

[38] A. Grattafiori *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[39] E. J. Hu *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[40] T. Haarnoja *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.