LEGO-Size: LLM-Enhanced GPU-Optimized Signoff-Accurate Differentiable VLSI Gate Sizing in Advanced Nodes

Yi-Chen Lu
yilu@nvidia.com
NVIDIA
Santa Clara, CA, USA

Kishor Kunal
kkunal@nvidia.com
NVIDIA
Santa Clara, CA, USA

Geraldo Pradipta
gpradipta@nvidia.com
NVIDIA
Santa Clara, CA, USA

Rongjian Liang
rliang@nvidia.com
NVIDIA
Austin, TX, USA

Ravikishore Gandikota
rgandikota@nvidia.com
NVIDIA
Santa Clara, CA, USA

Haoxing Ren
haoxingr@nvidia.com
NVIDIA
Austin, TX, USA

Abstract

On-Chip Variation (OCV)-aware and Path-Based Analysis (PBA)-accurate timing optimization achieved by gate sizing (including V_{th}-assignment) remains a pivotal step in modern signoff. However, in advanced nodes (e.g., 3*nm*), commercial tools often yield suboptimal results due to the intricate design demands and the vast choices of library cells that require substantial runtime and computational resources for exploration. To address these challenges, we introduce LEGO-Size, a generative framework that harnesses the power of Large Language Models (LLMs) and GPU-accelerated differentiable techniques for efficient gate sizing. LEGO-Size introduces three key innovations. First, it considers timing paths as sequences of tokenized library cells, casting gate sizing prediction as a language modeling task and solving it with self-supervised learning and supervised fine-tuning. Second, it employs a Graph Transformer (GT) with a linear-complexity attention mechanism for netlist encoding, enabling LLMs to make sizing decisions from a global perspective. Third, it integrates a differentiable Static Timing Analysis (STA) engine to refine LLM-predicted gate size probabilities by directly optimizing Total Negative Slack (TNS) through gradient descent. Experimental results on 5 **unseen** million-gate industrial designs in a commercial 3*nm* node show that LEGO-Size achieves up to 125x speed up with 37% TNS improvement over an industry-leading commercial signoff tool with minimal power and area overhead.

CCS Concepts

• **Hardware → Methodologies for EDA**; **Physical synthesis**.

Keywords

generative gate sizing, differentiable Static Timing Analysis (STA)

ACM Reference Format:

Yi-Chen Lu, Kishor Kunal, Geraldo Pradipta, Rongjian Liang, Ravikishore Gandikota, and Haoxing Ren. 2025. 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 (ISPD '25), March 16–19, 2025, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3698364.3705351

ISPD '25, March 16–19, 2025, Austin, TX, USA

ACM ISBN 979-8-4007-1293-7/25/03.
https://doi.org/10.1145/3698364.3705351

1 Introduction

Gate sizing is a fundamental optimization step in Physical Design (PD) that is widely used from synthesis to sign-off to improve power, performance, and area (PPA) metrics. However, in advanced technologies (i.e., 5*nm* and below), achieving desirable gate sizing results has become an increasingly challenging task. Commercial tools rely on multiple sizing iterations to meet PPA targets, which demands vast computational resources and extended runtime as modern designs easily consist of millions of gates. Furthermore, the relentless pursuit of high-performance and low-power designs forces signoff tools to adopt Path-Based Analysis (PBA) for timing verification to counter the pessimism of Graph-Based Analysis (GBA) and minimize over-design, which aggravates the runtime burden. Hence, the need for more efficient, scalable, and generalizable gate sizing methods has become extremely urgent.

Recently, Machine Learning (ML) has emerged as a promising solution to improve the gate sizing process in commercial tools. Several pioneering works, including ECO-GNN [24], DAGSizer [2], and TransSizer [27], have focused on building supervised prediction models aimed at boosting design productivity by enabling rapid sizing predictions. However, these methods fail to generalize across unseen designs and are fundamentally constrained by the training dataset, making them impossible to outperform the tools they aim to emulate. For example, in TransSizer [27], the prediction accuracy reaches 89% on training designs, but for unseen designs, it drops sharply to 61%. Finally, to achieve better-than-tool results, [23] introduced RL-Sizer, framing gate sizing as a Markov Decision Process (MDP) and solving it via Reinforcement Learning (RL). However, although RL-Sizer can outperform a leading PD tool, its runtime overhead makes it impractical for production flows.

In commercial tools, gate sizing is performed iteratively, with each iteration followed by Static Timing Analysis (STA) for verification, which is extremely time-consuming in advanced nodes [9]. To address this, AGD [29] introduces a differentiable gate sizing framework that relies on TimingGCN [13] for predictive propagation, where arrival and slack values at each level are predicted via neural networks, allowing gradients of global timing metrics, such as Total Negative Slack (TNS), to be easily computed with respect to input variables (e.g., gate sizes), forming an end-to-end differentiable sizing process. Nonetheless, AGD is constrained by its learning-based nature, which fails to generalize beyond the training dataset. Moreover, even when tested on an outdated 130nm

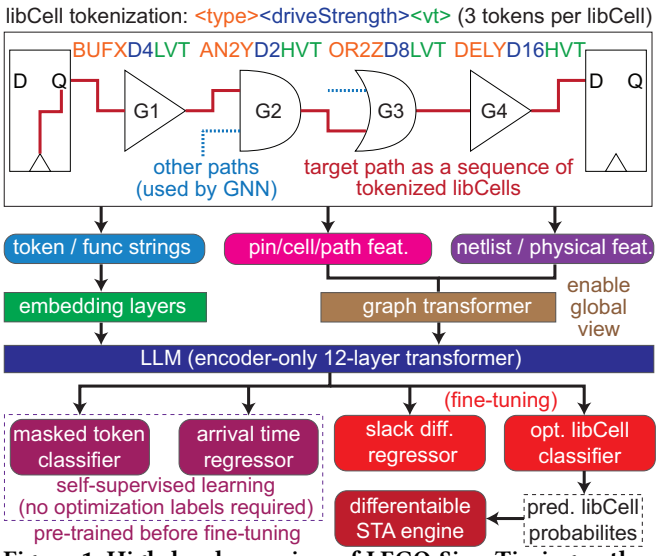

Figure 1: High-level overview of LEGO-Size. Timing paths are considered as sequences of tokenized library cells, and the gate sizing prediction task is solved through a language modeling approach. To achieve better PPA beyond the commercial signoff tool, a differentiable STA engine is developed to refine LLM-predicted probabilities.

technology comprising fewer than a hundred library cells, AGD is reported to be 60x slower than a commercial tool in reaching comparable optimization results. This inefficiency prohibits its use in industrial flows, since modern technologies easily involve tens of thousands of library cells.

The goal of this work is to develop a scalable, generalizable framework that is capable of delivering instant, better-than-tool signoff timing optimization results on **unseen** designs in advanced nodes. To achieve this, we develop LEGO-Size, a generative framework that leverages the power of Large Language Models (LLMs) and GPU-accelerated differentiable techniques for gate sizing prediction and optimization. We demonstrate that LEGO-Size outperforms an industry-leading signoff tool[1], at a commercial 3nm node with more than 10,000 library cells across 5 unseen industrial designs, each with millions of cells. Our contributions are as follows:

- We are the first work that casts the gate sizing prediction problem as a language modeling task. Unlike TransSizer [27] that only relies on manually defined features for representations, we develop the concept of library cell tokens for exact gate size prediction.
- Previous gate sizing prediction works [2, 24, 27] all confine to supervised learning settings. We are the first to introduce self-supervised learning tasks, including masked token prediction and arrival time increment prediction, to enhance the generalizability.
- We present the first signoff-accurate differentiable gate sizing framework in PD, which refines gate size probabilities through gradient descent. Unlike AGD [29] depending on TimingGCN [13] for predictive STA propagation via ML models, our framework is deterministic, On-Chip Variation (OCV) aware, and signoff-accurate. Notably, our framework achieves a 0.99 endpoint slack correlation with the leading signoff tool in OCV-mode.

[1]The name of the tool is withheld in compliance with the license agreement.

Table 1: Key differences: TransSizer [27] vs. LEGO-Size.

features	TransSizer [27]	LEGO-Size (ours)
model architecture	encoder-decoder	encoder-only
prediction style	iterative gate-by-gate	all gates simultaneous
language-based	no	yes (string tokens)
exact prediction	no (bin-based)	yes
pre-training tasks	no	yes
prob. refinement	heuristic search in bin	differentiable STA

2 LEGO-Size Overview

This section provides a high-level overview of LEGO-Size as illustrated in Figure 1. LEGO-Size features two key functions: first, an LLM-enhanced, PBA-accurate, path-based prediction model. Second, a differentiable STA engine that refines the predicted gate size probabilities to reach better-than-tool optimization results.

2.1 Tool-Optimized Gate Size Prediction

Given the necessity of PBA-accurate analysis to overcome GBA pessimism, gate sizing for timing optimization during signoff is typically conducted on a path-by-path basis. Hence, as illustrated in Figure 1, LEGO-Size performs path-based gate sizing predictions to facilitate signoff fixing (same as TransSizer [27]). Nonetheless, to ensure a global perspective in path-based predictions, a Graph Transformer (GT) [31] with a linear-complexity attention mechanism is developed to encode features across the entire graph.

One of the key innovations in LEGO-Size is that timing paths are considered as sequences of tokenized library cells (i.e., string-based tokens), and the gate sizing prediction task is cast as a language modeling problem solved by LLMs. Specifically, each library cell is tokenized into exactly three tokens denoting gate type, driving strength, and threshold voltage type (V_{th}). Note that this approach is fundamentally different from TransSizer [27], which lacks the concept of tokenization and only relies on handcrafted features to represent each cell (same as other works [2, 24]). Therefore, while TransSizer adopts the transformer-based architecture [33] for gate size predictions, it is not a language-based model as LEGO-Size.

Furthermore, unlike TransSizer [27] adopting an encoder-decoder transformer architecture that casts gate sizing as an iterative gate-by-gate prediction task, we introduce a PD-customized encoder-only transformer model that predicts gate sizes for all cells on a path simultaneously. Additionally, we develop self-supervised pre-training tasks, including masked token prediction and arrival time increment (i.e., stage delay) prediction to warm up token embeddings and trainable parameters within LEGO-Size. These pre-training tasks are crucial for enhancing the model's generalizability and are shown to significantly accelerate training convergence in subsequent Supervised Fine-Tuning (SFT) tasks, including path-based slack improvement prediction and final gate size prediction.

Note that while the SFT tasks leverage the tool's PBA-based signoff sizing results as labels, the self-supervised pre-training tasks do not require optimization labels. That is, both input features and groundtruths of the pre-training tasks are gathered from the design state **before** performing the time-consuming signoff optimization. This greatly mitigates the data scarcity challenge in the realm of PD, because such pre-optimized databases are abundant in any semiconductor company. Finally, the key differences between TransSizer [27] and LEGO-Size are summarized in Table 1.

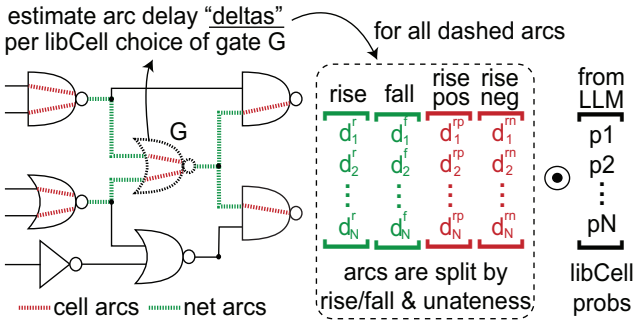

Figure 2: Illustration of the expected delta delay computation. Assume a target gate G has N sizing options, each associated with a probability predicted by the LLM model. For each option, we use the tool's estimateECO command to estimate the delay change d for neighboring arcs (dashed, 3-hop of G), categorized by rise/fall and unateness (positive/negative). A dot product between the estimated delay changes and the corresponding probabilities is applied to compute the total expected delta delay.

2.2 From Prediction to Optimization via GPU-accelerated Differentiable STA

The ultimate goal of this work is not only to match the optimization quality of the reference tool through prediction, but to achieve better-than-tool results through optimization. To accomplish this, we develop a GPU-accelerated differentiable STA engine that refines LLM-predicted gate size probabilities by optimizing the global TNS metric directly through gradient descent. Similar to DREAMPlace [20], we leverage PyTorch [28] C++/CUDA extension to build custom kernels for differentiable STA propagation. Specifically, we design a forward kernel for arrival time propagation from timing startpoints, and a backward kernel for gradient propagation from endpoints. During the propagation, the LLM-predicted gate size probabilities of each cell are used to re-annotate the original cell delays with the "expected" cell delays, and are made as leaf optimization variables (i.e, directly updated by the gradient descent optimizer), creating a fully end-to-end differentiable process.

Recently, extensive research has focused on GPU-accelerated STA, with prior works leveraging GPU power for both GBA-based [10, 11] and PBA-based [8, 9] timing analysis. However, these efforts primarily aim to improve STA runtime through parallelism, without introducing differentiability into the propagation process. Additionally, same as AGD [29] as aforementioned, they all do not account for OCV during propagation, and their timing correlation with commercial signoff tools remains unexplored. Another work [12] presents a timing-driven placement framework that employs the Non-Linear Delay Model (NLDM) to compute cell delays, which are differentiable with respect to pin locations using the chain rule. However, the use of NLDM, while effective for gradient computation, has proven to be inaccurate in advanced nodes, where Composite Current Source (CCS) models are preferred [19]. Hence, until now, no existing GPU-accelerated STA framework is suitable for the signoff gate sizing task in advanced nodes that we are solving in this work, as none provides both the differentiability required for gate size selection and the signoff accuracy that matches commercial tools, which motivates us to develop one.

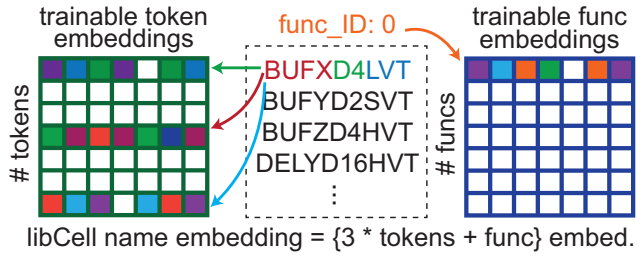

Figure 3: Illustration of constructing library cell embeddings through token and function embeddings in LEGO-Size. Each library cell name is tokenized into three distinct tokens, representing gate type (red), drive strength (green), and V_{th} type (blue). Additionally, the function of the library cell (func_ID) is incorporated as a trainable token to further capture the cell's characteristics. Note that only librarcy cells with same function are swappable.

2.3 Refining Probability with Delay Estimation

Unlike prior GPU-accelerated [8–11] or differentiable STA works [12, 29] that mainly focus on full graph STA update, our differentiable STA propagation is based on an incremental (or delta-based) philosophy. Specifically, given the initial state of a netlist before signoff optimization, and the LLM-predicted gate size probabilities for refinement, we first sync the arc delays from the initial state, and then re-annotate them by multiplying the probabilities with the reference tool's what-if analysis of delay changes (incurred by new gate sizes). For instance, given a cell arc t with an original delay D^t_{org}, and the LLM-predicted probability distribution $P = [p_1, p_2, p_3]$ across three potential library cell candidates, we first obtain the estimated delay changes $D = [d_1, d_2, d_3]$ with the what-if analysis. Then, the total expected delay change for arc t is calculated as $P \cdot D$, leading to a new delay of $D^t_{new} = D^t_{org} + P \cdot D$. Note the this what-if analysis to compute the expected delay changes D is common in commercial signoff tools and can be performed efficiently using CPU multi-threading as it does not introduce any real update (i.e., commit) to the underlying netlist.

Figure 2 illustrates the expected delta delay computation process in LEGO-Size. When sizing a cell G, we consider the impacted arcs as the dashed cell (red) and net (green) arcs in the figure, which approximately span a 3-hop neighborhood around G. Note that while the theoretical timing impact of sizing a cell can propagate throughout the netlist, this ripple effect is pointed out to diminish quickly in RL-Sizer [23], where a 3-hop neighborhood is shown to be sufficient for capturing the relevant timing impact.

Finally, after estimating the new delays for all arcs in the netlist impacted by the LLM-predicted gate size probabilities P, we leverage our STA kernels to perform instantaneous timing propagation with the newly annotated delays in an end-to-end differentiable manner. Particularly, our STA engine considers the gate size probabilities P as leaf differentiable variables, and calculates the gradient of TNS with respect to each gate size choice (i.e., $\frac{\partial TNS}{\partial P_i}$) directly. These gradients are used to refine the LLM-predicted probabilities, optimizing TNS in a truly global, full-graph optimization approach.

3 LEGO-Size Algorithm

In this work, we focus on gate sizing for timing optimization of combinatorial cells as TransSizer [27]. However, unlike TransSizer

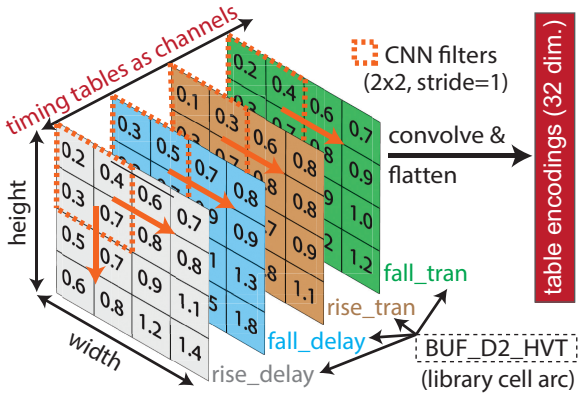

Figure 4: Illustration of CNN-based encoding of timing tables. Each table is treated as a grid channel, and rise/fall delay and transition tables are concatenated to capture joint relationships. CNNs then exploit the spatial structure in these grids, where neighboring entries reflect their proximity in the physical context.

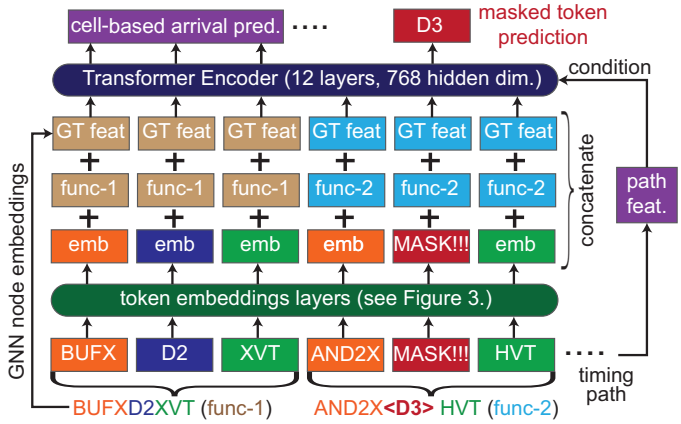

Figure 5: Illustration of the self-supervised tasks in LEGO-Size including masked token prediction and cell-based arrival time prediction. In the first task, tokens are randomly masked along the path, and the model must leverage the remaining context to predict the missing tokens. The "GT feat" refers to the node embeddings from the GT model.

struggling to generalize beyond the training designs, LEGO-Size is engineered to effectively handle unseen designs while achieving better-than-tool optimization results. An overview of LEGO-Size is presented in Figure 1. Below, we explore each component in detail.

3.1 LibCell Embeddings: Unlock Generalizable Gate Sizing via Language Modeling

It is widely acknowledged that the rise of LLMs has a foreseeable impact in the chip design process [21]. However, in PD, existing efforts have primarily focused on enhancing productivity through human-machine interaction (e.g., prompting) using natural language such as ChatEDA [35]. Up to now, no prior works have demonstrated the use of LLMs to solve combinatorial nature tasks such as gate sizing [14]. In this work, we present the first LLM-based framework that solves the gate sizing task via language modeling.

As aforementioned, we consider a timing path as a sequence of tokenized library cells. As illustrated in Figure 3, a library cell name is tokenized into 3 tokens, denoting gate type, drive strength, and V_{th} type of the cell. The key advantage of tokenization is that it significantly reduces the number of tokens that are required to characterize the entire vocabulary. For example, although the commercial 3nm node used in this work contains over 10,000 library cells (exact numbers cannot be disclosed due to confidentiality), they are formed by 300 tokens with our strategy. The key rationale can be explained with this equation:

$$\#\text{ all libCells} = \{\text{type tokens}\} \otimes \{\text{drv tokens}\} \otimes \{V_{th}\text{ tokens}\}, \quad (1)$$

where $\{\cdot\}$ denotes a token set, and $\otimes$ denotes the Cartesian product, meaning each element from one set is combined with every element from the other sets. The sum of tokens across the three sets in Equation 1 is much smaller than the total number of combinations generated by the Cartesian product. Finally, as shown in Figure 3, beside the library cell name tokenization, we create a "function token" per library cell group denoting the functionality.

Finally, while token and function embeddings capture some characteristics of a library cell, they fail to represent crucial physical properties such as timing tables, pin capacitance, and resistance, which are essential for accurate gate sizing. To address this, we

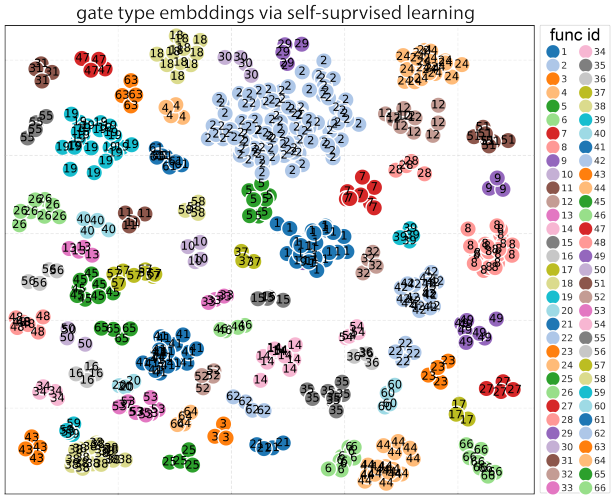

Figure 6: t-SNE [26] visualization results of gate type (e.g., BUFX) embeddings after self-supervised learning.

introduce a CNN-based timing table encoder as shown in Figure 4 to encode cell delay and transition tables. The rationale for applying a CNN to encode timing tables is that proximity among table values reflects key properties of library cells, governed by index vectors representing output loads and input slews. Thus, neighboring grids in the timing tables signal their proximity in the physical context, making convolution-based methods particularly suited for capturing these spatial relationships. The resulting table embeddings, along with pin capacitance and resistance, are concatenated with name embeddings to form the final cell embedding for the relevant arc in the timing path.

3.2 Self-Supervised Embedding Learning

In Natural Language Processing (NLP), word embeddings are critical for successful downstream Supervised Fine-Tuning (SFT) tasks [4]. Directly training embeddings from scratch during SFT has been shown to be less effective compared to pre-training with self-supervised

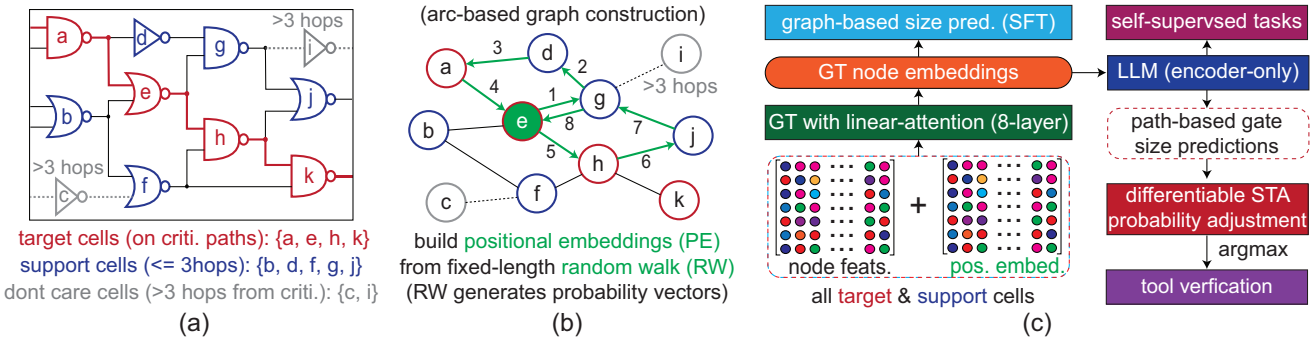

Figure 7: Node embedding construction in LEGO-Size via a scalable GT model [32] with a linear-complexity attention mechanism [3]. (a) Definition of target cells, support cells, and don't care cells. (b) Positional embeddings (PE) of each node are derived from random walks. (c) A 8-layer GT is utilized to construct GNN node embeddings. Note that a graph-based gate sizing prediction task is introduced to guide the GT for generating task-relevant embedding.

learning tasks [22]. We observe the same pattern in LEGO-Size. By pre-training the embeddings with PD-aware self-supervised tasks, LEGO-Size demonstrates significantly better generalization on unseen designs than previous works [2, 24, 27] that rely on direct SFT for gate sizing prediction.

Figure 5 illustrates the self-supervised tasks used to pre-train token embeddings and initialize LEGO-Size's parameters. These tasks include masked token prediction and cell-based arrival time increment prediction. In the first task, certain tokens are randomly masked, such as the drive strength token (D3) in the figure, which the model reconstructs by leveraging PD features, including path, cell, and pin attributes, as well as netlist information encoded by the GT model (detailed in the next section). In the second task, "arrival increment" represents the stage delay (cell arc + net arc delays). By using a CNN-based model to encode timing tables (Figure 4), the model predicts arrival time changes with a goal to enhance its PD awareness. Figure 6 visualizes the gate type token embeddings after self-supervised learning. Each dot represents a gate type token (e.g., BUFX), with embeddings projected onto a 2D plane using t-SNE [26]. Gate types with same logic function (e.g., BUFX, BUFY, ...) form distinct clusters. Experiments show that these pre-trained embeddings are essential for improving model's generalization to handle unseen designs (i.e., never used in SFT) effectively.

3.3 Netlist Encoding via Graph Transformer in Linear-Complexity Attention Mechanism

While LEGO-Size follows the path-by-path timing optimization methods as the commercial signoff timing ECO and TransSizer [27], we recognize that graph-based netlist attributes are vital for optimization. The decision on which cells to resize depends on factors such as the number of violating paths a cell sits on, its fanout/fanin structure, and its sensitivity to other endpoints. To capture this information, we employ Graph Neural Networks (GNNs). However, prior GNN approaches in PD, like ECO-GNN [24], limit their scope to local neighborhoods, using a 3-hop proxy for instance size changes. Similarly, DAGSizer [2] and TimingGCN [13] propagate information level-by-level but fail to capture long-range dependencies, akin to limitations in Recurrent Neural Networks (RNNs). Hence, a scalable GNN that jointly considers all cells is needed.

Table 2: Initial node features for the GT netlist encoding including both physical (first box) and netlist (second box) features. A node denotes a cell in the netlist graph.

feature	# dim.	description
cell density	3	densities in 3x3, 5x5, 7x7 bin granularities
congestion	2	horizontal and vertical routing congestions
bounding box	4	lower-left and upper-right coordiantes
wst. input slew	1	max input transition of input pin(s)
wst. output slew	1	max transition of output pin(s)
wst. output slack	1	min setup slack of output pin(s)
wst. input slack	1	min setup slack of input pin(s)
wst. output arrival	1	max arrival of output pin(s)
wst. input arrival	1	max arrival of input pin(s)
tot. input cap	1	sum of input pin cap
avg fanin cap	1	average capacitance of fan-ins
tot. pin res	1	sum of lib pin resistance
avg. pin res	1	average lib pin resistance
tot. drv. load	1	sum of driving load capacitance

To address this issue, we use a GT model [31] to perform attention across all nodes simultaneously. However, traditional self-attention mechanisms [33] have $O(n^2)$ complexity as:

$$\text{Self-Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2}$$

where $Q, K, V \in \mathbb{R}^{n\times d_k}$ are the query, key, and value matrices, and d_k is the hidden dimension. The quadratic complexity results from the softmax kernel, which computes the dot product of every query with every key, prohibiting practical use for large designs.

To overcome this challenge, we adopt linear-complexity attention from Performer [3], approximating softmax with a kernel feature map $\phi(\cdot)$ based on random Fourier features [30] as:

$$\text{Linear-Attention}(Q,K,V) = \phi(Q)\left(\phi(K)^T V\right), \tag{3}$$

$$\phi(x) = \frac{1}{\sqrt{m}}\left[\exp(i\omega_1^T x), \ldots, \exp(i\omega_m^T x)\right]^T, \tag{4}$$

with $\omega_1, \ldots, \omega_m$ sampled from $\mathcal{N}(0, I)$. This reduces complexity to $O(n)$, allowing efficient computation of node embeddings. To further improve computational efficiency, instead of feeding the entire netlist into the GT model, we focus on relevant cells as shown in Figure 7(a), including those on PBA-violating paths (target cells),

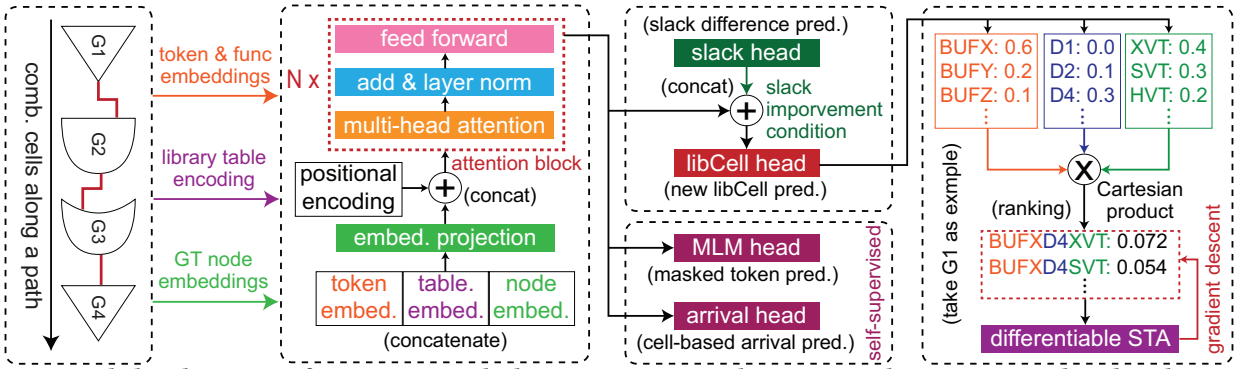

Figure 8: Detailed architecture of LEGO-Size. A slack improvement conditioning mechanism is introduced to the gate size prediction head (libCell head), which is followed by a differentiable STA engine for refinement of the LLM-predicted probabilities of library cells to achieve better-than-tool optimization results.

within a 3-hop neighborhood (support cells), or the remaining don't care cells. This reduces unnecessary computation, as the majority of cells have positive slack and minimal impact on sizing.

To encode graph Positional Embeddings (PE) into the GT model, we use Random Walk PE (RWPE) [7], where the random walk operator is $RW = AD^{-1}$. The k-step random walk RW^k defines a node's PE as:

$$\text{RWPE}(i) = \left[RW_{ii}, RW^2_{ii}, \ldots, RW^k_{ii}\right], \tag{5}$$

with k=12 in our work. These PE vectors are computed once per node and used throughout the learning process.

Our GT model builds node embeddings using the transformation method from [25], where each node represents a cell, and each edge represents a timing arc, with skip connections added between startpoints and endpoints. The initial node features include physical, timing, and RC attributes (Table 2), and arc delays are used as edge features. The aggregation process at each layer l is summarized as:

$$X^{(l+1)}_{MP}, E^{(l+1)} = \text{GINE}\left(X^{(l)}, E^{(l)}, A\right), \tag{6}$$

$$X^{(l+1)}_{LA} = \text{Linear-Attention}\left(X^{(l)}\right), \tag{7}$$

$$X^{(l+1)} = \text{MLP}\left(X^{(l+1)}_{MP} + X^{(l+1)}_{LA}\right), \tag{8}$$

where GINE [15] is a message passing network, A is the adjacency matrix, and $X^{(l)}$, $E^{(l)}$ are the node and edge features at layer l. At each level, node and edge embeddings from the previous level are processed through the GINE module for local aggregation (Equation 6), and the linear attention layer for global netlist encoding (Equation 7). The outputs of these two modules are then merged and passed through a Multi-Layer Perceptron (MLP) to generate the next-level embeddings $X^{(l+1)}$ (Equation 8). Ultimately, as shown in Figure 7(c), the embeddings $X^{(L)}$ from the last layer L are fed to the LLM for gate size predictions. During SFT, a graph-based size prediction task is introduced to further improve the quality of node embeddings, making it more task-relevant.

3.4 Slack-Improvement-Conditioned Path-Based Gate Size Prediction using LLM

After pre-training the token and function embeddings (Section 3.2), we fine-tune LEGO-Size using SFT tasks, including path-based slack improvement and optimized gate size prediction, with groundtruth labels from a commercial signoff tool. Unlike TransSizer [27], which predicts gate sizes sequentially, LEGO-Size predicts all gate sizes on a path simultaneously. Figure 8 outlines the gate size prediction process. First, we tokenize the PBA-violating path into a sequence of library tokens (Figure 3) and extract timing table embeddings via CNN (Figure 4). Concurrently, the GT model aggregates node features (Table 2), generating embeddings that are concatenated with the token and table embeddings as inputs to the LLM. Rotary PE is applied to preserve the order of library tokens, with the self-attention mechanism operating across $N = 12$ layers.

The key idea of slack improvement conditioning is to train the model to learn the precise amount of gate size change required to achieve the target slack improvement (i.e., the condition), which enables flexible trade-offs between timing and power. During SFT, the conditioning is achieved through teacher forcing [34], where the groundtruth slack improvement value is used as input. However, at inference, the model uses predicted slack improvement values to predict gate sizes. Another innovation is that we incorporate an entropy term as a regularization loss alongside the standard Cross-Entropy (CE) loss for gate size prediction. Given a probability vector $P = [p_1, p_2, \ldots, p_n]$, the entropy function $H(P) = -\sum_{i=1}^{n} p_i \log(p_i)$. This entropy regularization helps prevent overconfidence in the model's predictions by avoiding cases where a single gate size dominates with an excessively high probability, ensuring better generalization. Finally, the SFT loss functions for both slack improvement and gate size predictions are defined as follows:

$$\mathcal{L}_{\text{size}} = \text{CE}(\hat{Y}_{\text{size}}, Y_{\text{size}}) + \lambda \cdot H(\hat{Y}_{\text{size}}), \tag{9}$$

$$\mathcal{L}_{\text{slack}} = \text{RMSE}(\hat{Y}_{\text{slack}}, Y_{\text{slack}}), \tag{10}$$

where $\{\hat{Y}\}$ denote the predictions, $\{Y\}$ denote the groundtruths, λ denotes the regularization coefficient, and RMSE denotes the root-mean-squared-error loss. In the experiments, we set $\lambda = 0.001$.

3.5 Cross-Attention for Effective Embedding Learning and Cell Constraints Handling

In this work, we introduce a cross-attention mechanism during the decoding step to enable effective token embedding learning and robust handling of cell constraints as illustrated in Figure 9. The rationale is that unlike conventional NLP tasks, our path-based gate-sizing prediction task is a sequence-to-sequence problem in

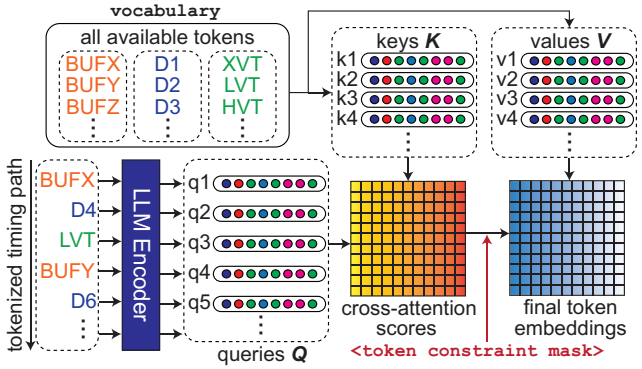

Figure 9: Cross-attention mechanism in LEGO-Size for effective embedding learning and accurate constraint handling.

which each input token directly maps to a corresponding output token because only gate sizes are changed and the tokenization rules are exact. For example, a drive-strength token can only be replaced by another drive-strength token, and a gate-type token must be mapped to another gate type with the same functional identifier. We exploit this property by performing cross-attention between each input token (as query Q) and its corresponding permitted tokens in the vocabulary serving as keys K and values V, which are governed by a predefined token constraint mask T. We then calculate attention scores based on Equation 2 to generate the final token embeddings of the input sequence. Note that the constraint mask ensures that any illegal tokens receive zero scores and are effectively excluded. Finally, we want to emphasize that this constraints-guided cross-attention mechanism can handle more complex cell constraint rules. For example, we can limit a buffer-function cell to only "delay cells" instead of all regular buffers for hold optimization while mitigating setup violations by modifying the token constraint mask.

3.6 Differentiable STA for Graph-Based Probability Refinement

The last piece of LEGO-Size is a GPU-accelerated differentiable STA engine that refines LLM-predicted gate size probabilities using a delta-based approach as described in Section 2.2, which allows us to achieve better-than-tool gate sizing results. As shown in Figure 2, for each cell G, given its sizing candidates and the probabilities predicted by the LLM, we first sync the initial arc delays from the pre-optimized netlist to our engine, and then re-annotate the delays of the cell and net arcs that are impacted by sizing (dashed arcs in Figure 2) by computing the dot product between the LLM-predicted probabilities and the delay estimates provided by the estimateECO command in the reference tool, which performs fast local what-if analyses over the impacted arcs.

Although the estimateECO command is highly parallelizable, allowing the estimation of millions of timing arcs within seconds, it operates under the assumption that the neighboring cells remain unchanged for estimation. Consequently, the delay estimation for each arc is conditioned on the stability of surrounding cells. To account for this assumption, we first prioritize gates to be sized in current iteration by sorting them based on the cumulative slack of PBA-violating paths they lie on. And then similar to RL-Sizer [23], we introduce a subgraph blocking mechanism, which prevents

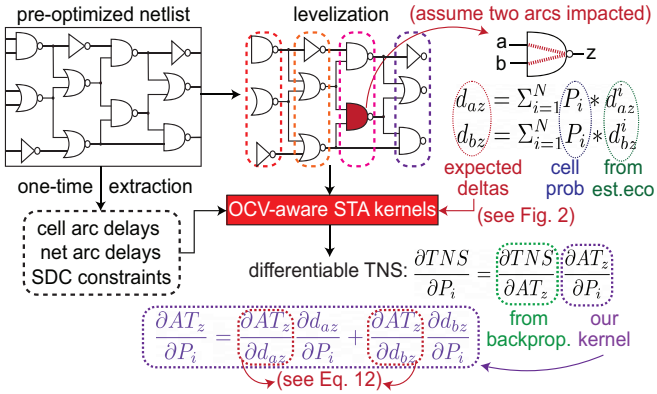

Figure 10: Our differentiable STA engine for LLM-predicted gate size probability refinement via gradient descent on the TNS value with re-annotated delays from estimateECO.

gates with overlapping subgraphs from being selected in the same iteration for differentiable gate sizing. Once the gates are selected, the LLM-predicted library cell probabilities for those gates are treated as differentiable leaf variables during the STA propagation. Specifically, these probabilities are used to update the initial arc delays related to the selected gates, as detailed in Section 2.2, directly influencing the calculation of the arrival time at each pin.

In arrival time propagation, the max operation is commonly used to select the maximum arrival time at a pin from multiple incoming paths. However, this operation presents a challenge, as it is not differentiable with respect to all inputs. To address this, we replace the max operation with the numerically stable Log-Sum-Exponential (LSE) operator, which provides a smooth approximation while retaining differentiability as:

$$\mathrm{LSE}(AT_1, AT_2, \ldots, AT_n) = M + T \cdot \log\left(\sum_{i=1}^{n} \exp\left(\frac{AT_i - M}{T}\right)\right), \quad (11)$$

where AT denotes the arrival time, M denotes the maximum arrival time $M = \max(AT_1, AT_2, \ldots, AT_n)$, and T denotes the temperature that controls the smoothness of the approximation. The gradient of Equation 11, which is essential for backpropagation in the differentiable STA engine, is given by:

$$\frac{\partial}{\partial AT_i}\mathrm{LSE}(AT_1, \ldots, AT_n) = \frac{\exp\left(\frac{AT_i - M}{T}\right)}{\sum_{j=1}^{n} \exp\left(\frac{AT_j - M}{T}\right)}, \quad (12)$$

which acts as a softmax-like distribution over the input arrival times. This smooth approximation ensures the model remains fully differentiable and allows the optimization to propagate through all relevant paths during backpropagation of gradient descent.

Figure 10 illustrates the workflow of our differentiable STA engine which refines the LLM-predicted gate size probabilities. The process begins with a one-time extraction of key inputs, including cell/net arc delays and SDC constraints that are necessary to build the timing graph. Also, the netlist is levelized once using topological sort to ensure parallel processing of arrival time calculation for pins at the same level. The core of our differentiable STA engine is two GPU-accelerated CUDA-based STA kernels: one for arrival time propagation from startpoints, and the other for gradient propagation from endpoints, respectively. For timing arcs that are

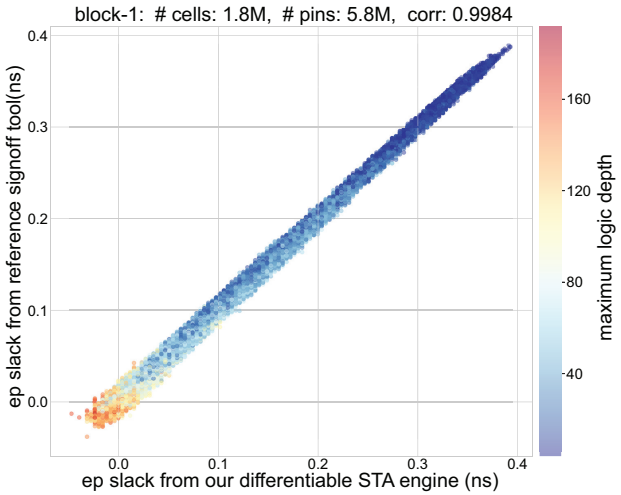

Figure 11: Endpoint slack correlation between the commercial signoff tool and our differentiable STA engine. Each dot denotes an endpoint and is colored by its maximum logic depth. Our GPU-accelerated differentiable STA propagation across the entire netlist (5.8M pins) takes less than 3 seconds.

impacted by sizing (e.g., az and bz arcs in the figure), their delays are re-annotated based on the dot product between LLM-predicted library cell probabilities $\{P\}$ and delta delay estimates $\{d\}$ derived from estimateECO. Note that as illustrated in Figure 2, this delta delay re-annotation, takes into account both rise/fall conditions and unateness. At the end of the propagation, the achieved TNS value is used to update the probabilities $\{P\}$ using gradient descent and chain rule. By leveraging PyTorch's auto-differentiation feature, we focus solely on computing the gradient $\frac{\partial AT_z}{\partial P_i}$ at each propagation level as shown in the purple box, where the key computation involved is shown in Equation 12.

Finally, as our framework targets signoff optimization, it has to be OCV-aware as the reference signoff tool for accurate STA propagation. To achieve this, our arrival kernel propagates both mean and standard deviation of delay distributions. Given an arc $j \rightarrow i$ and its variation attributes, the final arrival time AT at the pin i is determined by:

$$delay\_mean_i = delay\_mean_j + arc\_mean_{j \rightarrow i}, \quad (13)$$

$$delay\_std_i^2 = delay\_std_j^2 + arc\_std_{j \rightarrow i}^2, \quad (14)$$

$$AT_i = delay\_mean_i + \sigma * delay\_std_i, \quad (15)$$

where σ controls the level of pessimism. In alignment with the commercial signoff tool, we set $\sigma = 3.0$. Finally, Figure 11 presents the endpoint slack correlation results between our STA engine and the reference signoff tool on a commercial block with 5.8M pins and 1.8M cells. It is shown that the correlation reaches 0.9984, with the entire differentiable STA propagation takes less than 3 seconds.

4 Experimental Results

4.1 Experimental Setup and Infrastructure

We develop a comprehensive benchmark suite which includes around 28 millions timing paths with sizing optimization labels generated by an industry-standard signoff tool (the tool's name is withheld in compliance with license agreements) across 21 PD implementations (each with different recipes) of 8 million-scale industrial designs in a commercial 3nm technology node with more than 10,000 library cells, In contrast, the most advanced academic node ASAP-7 [5] only has 196 library cells in total. These extracted PBA setup-violating paths span across multiple PVT corners that reflect real-world signoff scenarios, providing rich RC and library information. Training and validation are performed on a Linux-based compute cluster with 8 NVIDIA A100 GPUs (each with 96GB memory), and AMD EPYC 7742 64-Core Processor with 2TB of RAM. The LEGO-Size framework is implemented using PyTorch [28] 2.1.2 and CUDA 12.1 for ML models and GPU-accelerated STA kernels for forward arrival propagation and backward gradient distribution.

4.2 Self-Supervised Pre-Training Results

The self-supervised pre-training phase is illustrated in Section 3.2, which requires approximately 200 real-hours (about 1011 GPU-hours) on our compute cluster. During pre-training, all trainable parameters including token, function embeddings, and the weights of the ML models are updated using the self-supervised tasks as illustrated in Figure 5 without the need of sizing labels. Note that while this computational requirement is substantial, it is a one-time cost that enables the subsequent rapid optimization of unseen designs in advanced technologies that offer a vast number of library cell choices. Once the pre-training is complete, we freeze the token and function embeddings, and proceed with fine-tuning the model on supervised tasks (Figure 8). The pre-training results in terms of gate type embeddings are visualized in Figure 6 with the t-SNE [26] dimension reduction technique.

4.3 LEGO-Size Optimization Results

In this experiment, we activate all components of LEGO-Size, including the GT model, LLM encoder, and differentiable STA engine for probability refinement, to directly compare with a commercial signoff tool on PBA-based timing optimization in a 3nm node. All benchmarks are evaluated in a completely unseen setting, achieved by fine-tuning the pre-trained model on other designs in the benchmark suite. For probability refinement, we use Adam [17] to perform 10 iterations of gradient descent, selecting the sizing solution with the best TNS from our differentiable STA engine. Table 3 presents the optimization results reported by the commercial signoff tool. LEGO-Size consistently surpasses the industry-leading tool across all benchmarks, achieving an average improvement of 26% in TNS and 15.9% in NVE, along with a 113X speedup. This is remarkable as all designs used for validation are **completely unseen**.

Finally, in the table, we also report the area impact of gate sizing between the commercial signoff tool and LEGO-Size, where a blue entry indicates situations in which LEGO-Size attains lower cell area than the industry reference. Note that although LEGO-Size does not explicitly optimize for area or density constraints, it often yields comparable or even reduced total cell area in advanced-node designs. We attribute this emergent behavior to our model's learning process: the optimization labels provided by the signoff tool inherently reflect a design methodology that prioritizes timing closure with minimal overhead in both area and power. By absorbing these labels during pre-training and fine-tuning, we believe LEGO-Size implicitly learns to inherit the signoff tool's tendency to preserve smaller gate footprints if possible. To further enhance

Table 3: Complete optimization results on totally unseen commercial blocks. Note that we use the same seed across all experiments to completely remove non-deterministic run-to-run variation. The timing unit is *ns* and the power unit is *mW*. Δarea shows the total cell area difference between the commercial tool and LEGO-Size, where blue indicates improvement.

unseen design (# cells)	initial state (pre-optimized)				commercial signoff tool (32 threads)					LEGO-Size (GT + LLM + Differentiable STA)					
	WNS	TNS	#vio. EPs	total power	WNS	TNS (goal)	#vio. EPs	total power	sped-up	WNS	TNS (goal)	#vio. EPs	total power	sped-up	Δarea (um^2)
block1 (1.8M)	-0.066	-17.46	2410	238.6	-0.028	-1.953	271	238.8	1.00	-0.034	-1.237 (-36.7%)	184 (-47.3%)	238.8	125x	-1.42
block2 (1.3M)	-0.041	-30.38	5410	293.4	-0.038	-7.82	1201	293.7	1.00	-0.035	-5.47 (-30.0%)	943 (-27.4%)	293.9	111x	-0.36
block3 (1.2M)	-0.054	-10.03	1539	119.7	-0.032	-5.37	906	119.8	1.00	-0.033	-4.32 (-19.5%)	812 (-10.4%)	119.8	100x	+2.28
block4 (1.5M)	-0.102	-59.25	6955	248.5	-0.097	-24.77	1912	249.1	1.00	-0.083	-20.36 (-17.8%)	1658 (-15.3%)	249.3	119x	+3.52
block5 (1.4M)	-0.072	-11.33	1525	127.2	-0.038	-6.68	620	127.5	1.00	-0.036	-4.85 (-27.4%)	529 (-17.2%)	127.5	114x	-1.15

Table 4: Ablation study on optimization results of predictive models. Refer to Table 3 for commercial tool baseline.

Design	Method	WNS	TNS	NVE	F1	Speedup
block1	TransSizer [27]	-0.045	-7.43	1072	0.65	367x
	ECO-GNN [24]	-0.048	-11.06	1157	0.56	1575X
	GT-only (ours)	-0.041	-9.84	1068	0.61	1427X
	GT+LLM (ours)	-0.032	-2.32	483	0.92	870X
block4	TransSizer [27]	-0.114	-35.49	3419	0.70	328x
	ECO-GNN [24]	-0.104	-32.12	3270	0.67	1362X
	GT-only (ours)	-0.094	-29.83	2709	0.74	1185X
	GT+LLM (ours)	-0.086	-22.33	1852	0.87	627X

area efficiency in gate sizing, we plan to incorporate area as a regularization term within the gradient descent process that optimizes WNS and TNS (Section 3.6). By leveraging the probabilistic gate selections to estimate "expected" area overhead, we can guide final sizing decisions while balancing timing and area objectives.

4.4 Optimization Results of Predictive Models

Now, we conduct an ablation study to analyze the optimization results of predictive models. In LEGO-Size, gate size predictions can be obtained from two key components: the GT model, which enhances node embeddings through a graph-based gate size prediction task (Figure 7), and the LLM encoder (before differentiable STA refinement) which makes path-based predictions. We benchmark our predictive models against ECO-GNN [24], which focuses on graph-based predictions similar to the GT model, and TransSizer [27], which performs path-based predictions as our LLM encoder. Table 4 presents the comparison, showing that the proposed LLM encoder, which incorporates GT node embeddings, achieves the best prediction accuracy among unseen designs. We attribute this improvement to the proposed self-supervised pre-training tasks. Importantly, our LLM encoder demonstrates a much faster runtime compared to TransSizer, as it predicts gate sizes for an entire path simultaneously rather than sequentially (i.e., auto-regressive) as in TransSizer. Finally, for cells on multiple violating paths, we select predictions with the largest drive strengths as the final gate sizes.

4.5 Slack-Conditioning Experiment

As shown in Figure 8, we introduce a slack improvement conditioning mechanism to guide gate size predictions of the LLM encoder. The concept is similar to the "effort level" used in TransSizer [27], however, instead of using discrete levels (e.g., low, medium, high) as TransSizer, our approach conditions the model on the expected slack difference before and after optimization. This fine-grained conditioning enables the model to better predict the extent of gate size changes required to meet specific slack improvement targets.

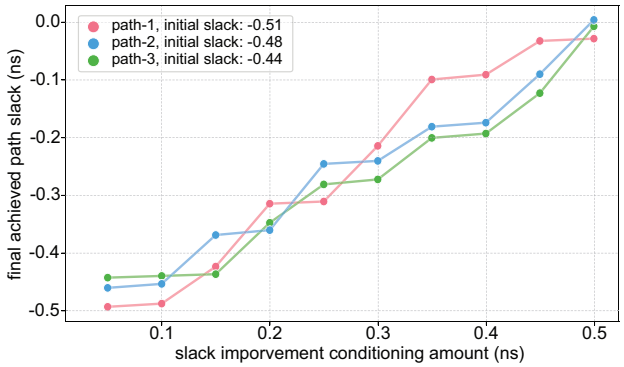

Figure 12: Slack conditioning experiment. We vary the amount of slack improvement condition for LLM and evaluate the sizing predictions using the commercial signoff tool.

Figure 12 illustrates our slack conditioning study, where the x-axis represents the conditioned slack improvement of each path, and the y-axis shows the final achieved path slack. While the relationship is not perfectly linear (which is ideal), a clear positive correlation emerges, indicating that our slack improvement conditioning approach is effectively guiding the LLMs for accurate predictions. We envision this slack improvement conditioning technique can enable flexible trade-off between timing and power metrics.

5 Discussion

In this section, we discuss the key considerations that enable LEGO-Size to achieve robust performance in a commercial 3*nm* node on real-world, high-performance industrial designs. We also outline potential enhancements for further improving optimization quality and integrating LEGO-Size into leading industrial production flows.

5.1 The Power of Differentiable STA Engine

Although prior works [2, 24, 27] have demonstrated the effectiveness of ML-based approaches for gate sizing predictions, they remain limited by the optimization quality of the data and tools they emulate. Our differentiable STA engine overcomes these limitations by offering an entirely new optimization dimension that complements the predictive module. Specifically, once a pre-optimized netlist is obtained from the LLM-encoder, we narrow the high-dimensional solution space to a near-optimal region, then refine the gate sizes within that region by directly performing gradient descent on the global metrics such as TNS to update leaf variables. While commercial signoff tools typically rely on localized analytical methods often due to memory constraints as the number of decision variables grows, our differentiable STA engine can execute a full-graph timing optimization on a single GPU using under 10GB

of memory per design. This enables a holistic view of the netlist and is well-suited for large-scale designs in a 3nm technology node.

In this work, we encode the probabilities of each library cell choice as the leaf optimization variables, where each chosen cell triggers a localized fast re-annotation of the 3-hop neighborhood subgraph (see Figure 2). Therefore, we do not have the rounding issues as we are not leveraging the common continuous relaxation technique for gradient descent. However, a natural challenge in gradient-based optimization is to handle the high-dimensional space of variables, particularly for multi-million-cell netlists [18]. To address this, we focus only on gates whose LLM-predicted library choices differ from their originals, substantially reducing the dimensionality and providing a strong initialization for the differentiable STA engine (in contrast to random guessing as in AGD [29]). This careful restriction of optimization variables, combined with our full-graph GPU-accelerated timing propagation, leads to the ultimate success of LEGO-Size.

5.2 Industrial Consideration: Handling Scale and Complexity via Language Modeling

LEGO-Size is specifically tailored for *industrial design flows* and is validated in this work using a 3*nm* node with real-world, multi-million-cell designs. Unlike academic PDKs such as ASAP7 [5], which offers only 196 library cells, commercial 3*nm* technologies easily present tens of thousands of cell options with complex characteristics, necessitating robust strategies that can handle enormous design spaces without exhausting computational resources.

In LEGO-Size, a key innovation for tackling the above challenges is the simple yet elegant tokenization strategy that converts netlists into "sequences" suitable for language modeling. Together with the token constraint mask approach (described in Figure 9 and Section 3.5), this method effectively prevents the combinatorial explosion of the library search space while enabling accurate learning of relationships among tokens through language modeling. Another key technique is the linear attention mechanism that captures crucial graph connectivity at scale without incurring prohibitive computational overhead. Such graph-based encoding is critical because the final gate size decision for each instance depends not only on the most critical path it sits on, but also on the sub-critical paths as well as the neighboring parasitic effects in its vicinity.

5.3 Tackling Multi-Corner Multi-Mode with Differentiable Gate Sizing

Although Table 3 reports results for a single-scenario setting, LEGO-Size can be naturally extended to handle Multi-Corner Multi-Mode (MCMM). Unlike TransSizer [27] addresses MCMM gate sizing by simply selecting features from the "dominant" scenario, LEGO-Size can easily refine LLM-predicted gate-size probabilities across all desired scenarios via our differentiable STA engine, where we can maintain a separate timing graph for each scenario and merge them into a single PyTorch computational graph, enabling gradient-based updates that jointly improve the gate size selections for every scenario. A straightforward approach is to form a weighted sum of TNS across multiple scenarios, reflecting their relative importance. Our differentiable STA kernels then back-propagate gradients from this weighted-summed global TNS metric into library cell probabilities, guiding the final cell choices toward a solution that optimizes the objective. We envision that by going beyond TransSizer's [27] dominant-scenario focus for MCMM handling and explicitly covering sub-critical coroners, this holistic approach yields a more robust, practical gate sizing solution for real-world industrial flows.

5.4 Scaling Language Models for Gate Sizing

Language modeling has emerged as a powerful technique for capturing rich representations, with scaling laws indicating that performance improves when model capacity and dataset size grow [16]. However, naively enlarging a model without a robust "warmup" stage can lead to slow or suboptimal convergence during the proceeding SFT phase [6]. To address this, we employ a customized pre-training phase in LEGO-Size, consuming over 1000 GPU-hours on high-performance clusters, to construct meaningful token and function embeddings before fine-tuning (Figure 6).

By integrating industrial signoff databases during pre-training, LEGO-Size acquires meaningful embeddings and starts from a well-initialized parameter space. Subsequent fine-tuning then focuses on gate sizing for timing closure, showing faster convergence and higher accuracy on *unseen* designs (Table 3). This multi-stage approach aligns with other findings of applying language models in domain-specific tasks [1], where an effective pre-training routine reliably boosts downstream tasks. Motivated by these results, we plan to scale both the model's size and the dataset in future work, further harnessing language modeling principles and advancing LEGO-Size's accuracy on complex designs.

5.5 Implicit Area-Aware Gate Sizing via Effective Generative Learning

Although LEGO-Size does not explicitly optimize for area efficiency during gate sizing, training on the optimization data generated from the reference commercial signoff tool indirectly incorporates area-aware sizing decisions. This is because the reference tool's optimization engine aims to fix timing with minimal area overhead, LEGO-Size inherits a similar practice, leading to small or even negligible area changes compared to the reference tool. The area impact is shown in Table 3. While certain blocks experience slight increases due to aggressive upsizing and others see modest improvements, the area difference is below 0.01% of total combinational cell area across all benchmarks. While LEGO-Size currently focuses on timing closure alone, looking ahead, we plan to integrate LEGO-Size into a complete ECO signoff flow that encompasses ECO-place/route, EM/IR closure, and other ECO checks.

6 Conclusion and Future Work

In this paper, we have introduced LEGO-Size, a generative framework that significantly advances signoff optimization quality of an industry-leading signoff tool via language modeling and differentiable gate sizing. By framing gate sizing as a language modeling task and utilizing self-supervised learning, LEGO-Size brings a novel approach to handling complex design characteristics in a 3*nm* node with over 10,000 library cells effectively. Furthermore, the integration of a differentiable STA engine enables direct optimization of TNS via gradient descent, yielding better-than-tool optimization results. In the future, we plan to (1) take area as regularization during gate sizing, (2) improve LEGO-Size for multi-scenario handling via differentiable STA, and (3) validate the framework in a complete signoff flow. We believe this work shall open the gate of using language models to solve key PD optimization problems.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[2] C.-K. Cheng, C. Holtz, A. B. Kahng, B. Lin, and U. Mallappa. Dagsizer: A directed graph convolutional network approach to discrete gate sizing of vlsi graphs. ACM Transactions on Design Automation of Electronic Systems, 28(4):1–31, 2023.

[3] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. Rethinking attention with performers. arXiv preprint arXiv:2009.14794, 2020.

[4] K. W. Church. Word2vec. Natural Language Engineering, 23(1):155–162, 2017.

[5] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. Asap7: A 7-nm finfet predictive process design kit. Microelectronics Journal, 53:105–115, 2016.

[6] J. Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[7] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Graph neural networks with learnable structural and positional representations. arXiv preprint arXiv:2110.07875, 2021.

[8] G. Guo, T.-W. Huang, Y. Lin, and M. Wong. Gpu-accelerated critical path generation with path constraints. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2021.

[9] G. Guo, T.-W. Huang, Y. Lin, and M. Wong. Gpu-accelerated path-based timing analysis. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 721–726. IEEE, 2021.

[10] Z. Guo, T.-W. Huang, and Y. Lin. Gpu-accelerated static timing analysis. In Proceedings of the 39th international conference on computer-aided design, pages 1–9, 2020.

[11] Z. Guo, T.-W. Huang, and Y. Lin. Accelerating static timing analysis using cpu–gpu heterogeneous parallelism. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(12):4973–4984, 2023.

[12] Z. Guo and Y. Lin. Differentiable-timing-driven global placement. In Proceedings of the 59th ACM/IEEE Design Automation Conference, pages 1315–1320, 2022.

[13] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In Proceedings of the 59th ACM/IEEE Design Automation Conference, pages 1207–1212, 2022.

[14] Z. He and B. Yu. Large language models for eda: Future or mirage? In Proceedings of the 2024 International Symposium on Physical Design, pages 65–66, 2024.

[15] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265, 2019.

[16] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[18] R. Liang, A. Agnesina, and H. Ren. Medpart: A multi-level evolutionary differentiable hypergraph partitioner. In Proceedings of the 2024 International Symposium on Physical Design, pages 3–11, 2024.

[19] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. F. Young, and M. D. Wong. Gcs-timer: Gpu-accelerated current source model based static timing analysis. In Proceedings of the 61th ACM/IEEE Design Automation Conference, 2024.

[20] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(4):748–761, 2020.

[21] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, et al. Chipnemo: Domain-adapted llms for chip design. arXiv preprint arXiv:2311.00176, 2023.

[22] Y. Liu. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

[23] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 733–738. IEEE, 2021.

[24] Y.-C. Lu, S. Nath, S. Pentapati, and S. K. Lim. Eco-gnn: Signoff power prediction using graph neural networks with subgraph approximation. ACM Transactions on Design Automation of Electronic Systems, 28(4):1–22, 2023.

[25] Y.-C. Lu, T. Yang, S. K. Lim, and H. Ren. Placement optimization via ppa-directed graph clustering. In 2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD), pages 1–6. IEEE, 2022.

[26] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[27] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren. Transsizer: A novel transformer-based fast gate sizer. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, pages 1–9, 2022.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 2019.

[29] P. Pham and J. Chung. Agd: A learning-based optimization framework for eda and its application to gate sizing. In 2023 60th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2023.

[30] A. Rahimi and B. Recht. Random features for large-scale kernel machines. Advances in neural information processing systems, 20, 2007.

[31] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a general, powerful, scalable graph transformer, 2022.

[32] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a general, powerful, scalable graph transformer. Advances in Neural Information Processing Systems, 35:14501–14515, 2022.

[33] A. Vaswani. Attention is all you need. Advances in Neural Information Processing Systems, 2017.

[34] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. Neural computation, 1(2):270–280, 1989.

[35] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu. Chateda: A large language model powered autonomous agent for eda. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2024.