BEYOND TOKENS: ENHANCING RTL QUALITY ESTIMATION VIA STRUCTURAL GRAPH LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Estimating the quality of register transfer level (RTL) designs is crucial in the electronic design automation (EDA) workflow, as it enables instant feedback on key performance metrics like area and delay without the need for time-consuming logic synthesis. While recent approaches have leveraged large language models (LLMs) to derive embeddings from RTL code and achieved promising results, they overlook the structural semantics essential for accurate quality estimation. In contrast, the control data flow graph (CDFG) view exposes the design's structural characteristics more explicitly, offering richer cues for representation learning. In this work, we introduce StructRTL, a novel structure-aware graph self-supervised learning framework for improved RTL design quality estimation. By learning structure-informed representations from CDFGs, StructRTL significantly outperforms prior art on various quality estimation tasks. To further boost performance, we incorporate a knowledge distillation strategy that transfers low-level insights from post-mapping netlists into the CDFG-based predictor. Experimental results demonstrate that StructRTL establishes new state-of-the-art results, highlighting the effectiveness of combining structural learning with cross-stage supervision.

1 INTRODUCTION

As AI models continue to grow in size and computational demands, their success depends not only on algorithmic innovation but also on advances in the underlying hardware. Notably, it has been estimated that more than half of the performance gains in AI systems over the past decade can be attributed to improvements in hardware alone (Erdil & Besiroglu, 2022; Ho et al., 2024). This highlights the critical role of hardware innovation in driving and sustaining AI progress. At the same time, the growing demand for specialized and high-performance hardware places greater pressure on the efficiency of the hardware design process itself. Consequently, there is a pressing need for new methodologies that can accelerate design cycles and improve design quality and performance.

Modern hardware design is a complex, multi-stage process that begins with high-level specifications in natural language, which are manually translated into hardware description languages (HDLs) such as Verilog or VHDL, before synthesizing into circuit elements. At the heart of this process lies the register transfer level (RTL), a critical abstraction that bridges architectural intent and low-level circuit implementation, enabling designers to model intricate digital systems in a way that is both expressive and amenable to synthesis into gate-level representations. Typically, RTL design is refined iteratively, with engineers assessing the quality of the synthesized netlist using key metrics such as area and delay. However, this feedback loop is inherently slow and computationally expensive, as each iteration requires invoking a full logic synthesis toolchain. To reduce design turnaround time and improve productivity, there is an increasing demand for fast and reliable methods that can estimate design quality directly from RTL, enabling early feedback without sacrificing accuracy.

To address the aforementioned issue, prior research has explored machine learning-based approaches that represent hardware code as graphs, *e.g.*, data flow graphs (DFGs) and abstract syntax trees (ASTs), and use hand-crafted features derived from these graphs for quality estimation (Zhou et al., 2019; Lopera et al., 2021; Sengupta et al., 2022; Fang et al., 2023). While these methods have shown encouraging results, their reliance on manually designed features significantly limits their expressiveness and ability to capture the rich structural semantics in RTL designs. As a result, these shallow representations often fail to model the complex patterns that affect downstream performance, con-

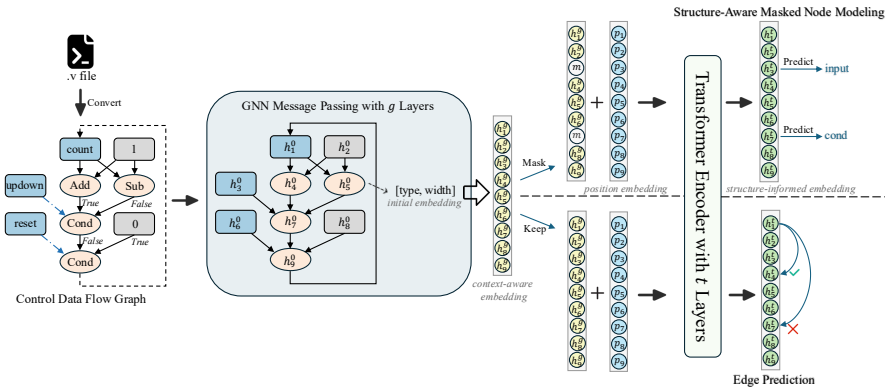


Figure 1: Overview of the StructRTL framework for structure-aware graph self-supervised learning. It employs two pretraining tasks: structure-aware masked node modeling, where the Transformer encoder predicts masked node types from post-GNN context-aware embeddings, and edge prediction, which recovers graph connectivity from the same embeddings without masking.

straining their effectiveness for accurate quality estimation. Moreover, DFGs provide only a partial perspective of the design by focusing solely on data dependencies while neglecting control flow, and ASTs primarily reflect syntactic hierarchy, effectively serving as an augmented form of the original code without explicitly encoding structural semantics. In contrast, control data flow graphs (CDFGs) integrate both control and data dependencies, yielding a holistic and semantically rich representation of RTL designs. This makes CDFGs particularly well suited for learning representations that are sensitive to the structural properties critical for accurate quality estimation. Recently, with the rise of large language models (LLMs) (Grattafiori et al., 2024; Hurst et al., 2024; Guo et al., 2025), some approaches have attempted to derive embeddings from RTL code using LLMs specifically trained for Verilog generation (Pei et al., 2024; Zhao et al., 2025; Liu et al., 2025b), achieving state-of-the-art performance in quality estimation tasks (Moravej et al., 2025). However, a fundamental gap exists between the objectives of code generation and quality estimation, which may limit the transferability of representations learned from generation-focused pretraining. Besides, compared to CDFG, the token-based view encodes the structural semantics of RTL designs implicitly, providing fewer cues for learning, which may hinder the final performance.

In this work, we introduce StructRTL, a novel structure-aware graph self-supervised learning framework designed to improve RTL design quality estimation. Instead of using token-based view, StructRTL operates on the CDFG view, which more explicitly captures the structural semantics of RTL designs. Besides, rather than relying on shallow, hand-crafted features, StructRTL employs two self-supervised pretraining tasks, *i.e.*, structure-aware masked node modeling and edge prediction, to learn rich structure-informed representations from CDFGs. These pretraining objectives enable the model to capture complex dependencies and design patterns, leading to significant performance improvements over existing approaches on various quality estimation tasks. Furthermore, inspired by VeriDistill (Moravej et al., 2025), we incorporate a knowledge distillation strategy that transfers low-level insights from post-mapping netlists into the CDFG-based predictor, further enhancing the model's predictive capabilities. Experimental results demonstrate that StructRTL achieves new state-of-the-art performance, highlighting the effectiveness of combining structural representation learning with cross-stage supervision.

2 Related Work

RTL Design Quality Estimation. Early efforts to estimate RTL design quality have primarily relied on features extracted from graph-based representations of RTL code. For instance, Lopera et al. (2021) explore delay prediction by converting RTL designs into DFGs and extracting internal cell connectivity statistics as features to predict post-synthesis critical path delay. However, this approach has only been applied to simple combinational circuits such as adders and is trained on synthetic design variants generated by an RTL generator, limiting its practical applicability. Sengupta et al. (2022) represent RTL code as ASTs and extract hand-crafted features like total input and output bits and average wire width, which are then used to train traditional machine learning models like XGBoost (Chen & Guestrin, 2016) for quality estimation. Similarly, MasterRTL (Fang et al., 2023) converts RTL code into a bit-level design representation called the simple operator graph (SOG) and

extracts features from it for the same purpose. While these approaches have shown some promise, they share two key limitations. First, their reliance on hand-crafted features limits the expressiveness of the representation and fails to capture the rich structural semantics of RTL designs. Second, their training and evaluation are conducted on small datasets, typically fewer than 100 designs in total, raising concerns about the models' generalization capabilities. Moreover, the construction of SOG requires invoking logic synthesis steps, placing it closer to the gate-level netlist stage rather than purely RTL. As such, it falls outside the scope of our work.

Recently, the emergence of LLMs (Grattafiori et al., 2024; Hurst et al., 2024; Guo et al., 2025) has sparked growing interest in fine-tuning these models for Verilog code generation (Pei et al., 2024; Zhao et al., 2025; Liu et al., 2025a;b;c). Following this trend, VeriDistill (Moravej et al., 2025) has proposed leveraging LLMs specifically trained for Verilog generation to extract embeddings from RTL code, achieving state-of-the-art performance on RTL quality estimation tasks using a large dataset of over 10,000 designs. However, there remains a fundamental mismatch between the objectives of code generation and quality estimation, which may limit the transferability of representations learned from generation-focused pretraining. Moreover, unlike CDFG, the token-based view encodes the structural semantics of RTL designs implicitly, offering fewer cues for learning. This lack of explicit structural information can make it more difficult for models to capture the complex patterns critical for accurate quality prediction.

Graph Self-Supervised Learning. Graph self-supervised learning, which learns generalizable representations from unlabeled graph data, has emerged as a popular and empirically successful learning paradigm for graph neural networks (GNNs) (Liu et al., 2022). Broadly, these methods fall into two categories: contrastive and generative approaches. Contrastive methods aim to learn representations that are invariant to different augmented views of a graph (Hassani & Khasahmadi, 2020; You et al., 2020; 2021), while generative methods create supervision signals by masking parts of the graph, such as node attributes or edges, and train the model to reconstruct the missing components (Hou et al., 2022; Li et al., 2023). In this work, we adopt generative approaches to learn structure-informed representations for RTL design quality estimation.

Inspired by the success of BERT (Devlin et al., 2019) in natural language processing (NLP), GraphMAE (Hou et al., 2022) demonstrates that masking a portion of node features and reconstructing them from their surrounding context can yield high-quality representations, achieving state-of-the-art performance on a variety of tasks. Complementarily, MaskGAE (Li et al., 2023) focuses on the structural aspect by masking a subset of edges or even paths and training the model to recover the graph connectivity, thereby learning rich topological features. These approaches compel the model to decipher the underlying relational patterns within the graph, thereby generating robust and informative embeddings. However, directly applying these techniques to CDFGs of RTL designs poses challenges. Unlike general-purpose graphs, CDFGs are tightly coupled with computational semantics, where masking nodes or edges, such as an arithmetic operator, can introduce ambiguity. For instance, if a plus operator is masked, multiple replacements (*e.g.* minus, multiply), could all appear valid, undermining the learning signal. To address this, we design two tailored pretraining objectives: structure-aware masked node modeling and edge prediction. These adaptations allow us to preserve the computational integrity of CDFGs while still leveraging the benefits of generative self-supervision. Details of these techniques are presented in the following section.

3 METHODOLOGY

In this section, we present the details of our proposed framework, StructRTL. It incorporates two tailored pretraining tasks, structure-aware masked node modeling and edge prediction, to learn structure-informed representations that are essential for downstream RTL design quality estimation. To further improve performance, we employ a knowledge distillation strategy that transfers low-level insights from post-mapping (PM) netlists into the CDFG-based predictor, enhancing the model's ability to estimate quality metrics accurately.

3.1 CDFG CONSTRUCTION

RTL is a design abstraction level used in digital circuit design that models the flow of data between registers and the operations performed on that data. It encompasses both combinational logic (*e.g.*,

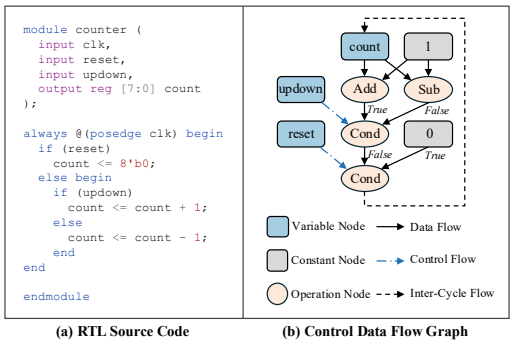


Figure 2: Example RTL design and corresponding CDFG.

arithmetic operations and control branches) and sequential logic (*e.g.*, registers that store state across clock cycles). A common representation of RTL designs is CDFG, where nodes represent operations or storage elements, and directed edges indicate data or control dependencies between them. In sequential circuits, registers hold values between clock cycles, so their incoming edges represent inter-cycle data flow, making CDFGs cyclic graphs.

To construct a CDFG from RTL source code, we first use Yosys (Wolf et al., 2013) to compile the design into RTL intermediate language (RTLIL), a simplified form that preserves functionality while reducing the design to basic assignment and register-transfer operations. This intermediate form makes CDFG extraction more straightforward. We then apply the Stagira Verilog parser (Chen et al., 2023) to generate an AST from the RTLIL. Finally, we traverse the AST to extract the CDFG. An example RTL design and its corresponding CDFG are shown in Figure 2. For more details on the CDFG, please refer to Appendix A.2.

3.2 Model Architecture

As illustrated in Figure 1, StructRTL integrates a GNN with a Transformer encoder (Vaswani et al., 2017). Given a CDFG $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ denotes the set of nodes and $\mathbf{E}$ represents the edges, we first initialize the node embeddings $\{\mathbf{h}_i^0\}$ as the concatenation of the one-hot encoding of the node type and the node width:

$$\mathbf{h}_i^0 = \text{concat}\left(\text{one-hot}(\text{type}(v_i)), \text{width}(v_i)\right) \tag{1}$$

Next, message passing is performed on $\mathbf{G}$ using a graph isomorphism network (GIN) (Xu et al., 2019) to process the node embeddings and generate context-aware embeddings $\{\mathbf{h}_i^g\}$. These updated embeddings are then passed into the Transformer encoder. To preserve the structural information of the graph, we combine $\{\mathbf{h}_i^g\}$ with global positional embeddings $\{\mathbf{p}_i\}$ (Rampášek et al., 2022). This step is crucial because when the node embeddings are flattened into a sequence for the Transformer, the graph's connectivity information is lost. Without the global positional embeddings, the model would struggle to distinguish between similar nodes located in distinct regions of the graph, leading to training collapse.

To compute the global positional embeddings $\{\mathbf{p}_i\}$, we first calculate the symmetric normalized Laplacian matrix for the directed graph (Chung, 1997):

$$L = I - D_{\text{in}}^{-1/2}\left(\frac{A + A^{\text{T}}}{2}\right)D_{\text{out}}^{-1/2} \tag{2}$$

where A denotes the adjacency matrix, and D_{in} and D_{out} are the in-degree and out-degree matrices, respectively. The eigenvalues and corresponding eigenvectors of L are then computed by solving the following equation:

$$L\mathbf{x} = \lambda\mathbf{x} \tag{3}$$

where $\{\lambda_i\}$ are the eigenvalues and $\{\mathbf{x}_i\}$ are corresponding normalized eigenvectors. To construct the global positional embeddings, we select the k eigenvectors corresponding to the smallest k eigenvalues, where $k = 16$ in this work. If the number of nodes is smaller than 16, we pad the embeddings with $\mathbf{0}$. We then project $\{\mathbf{p}_i\}$ using a linear projection layer to ensure it has the same dimensionality as $\{\mathbf{h}_i^g\}$. Additionally, since eigenvectors $\{\mathbf{x}_i\}$ are inherently sign-insensitive (*i.e.*, both $\mathbf{x}_i$ and

Algorithm 1 Stratified Masking

Input: Labels $L \in \mathbb{Z}^N$, mask ratio r, minimum per class m
Output: Mask $M \in \{0,1\}^N$

$M \leftarrow \text{RandomBoolMask}(N, r)$ {Each entry is 1 with probability r}
for all class $c \in \text{Unique}(L)$ **do**
  $I_c \leftarrow \{i \mid L_i = c\}$
  $k \leftarrow \sum_{i \in I_c} M_i$ {Num. of masked nodes in class c}
  if $k < m$ **then**
    $E \leftarrow \text{RandSample}(I_c, \min(m, |I_c|))$
    for all $i \in E$ **do**
      $M_i \leftarrow 1$
    end for
  end if
end for
return M

Algorithm 2 Class-Balanced Focal Loss

Input: Samples per class S, output logits Z, labels L
Parameters: $\beta = 0.9999, \gamma = 2.0$
Output: Loss $\mathcal{L}_{cb\_focal}$

for each class c **do**
  $\text{effective\_num}_c \leftarrow 1.0 - \beta^{S_c}$
  $w_c \leftarrow \frac{1-\beta}{\text{effective\_num}_c + 1e{-}8}$
end for
$w_c \leftarrow \frac{w_c}{\sum_c w_c} \times \text{len}(S)$
for each sample i in batch **do**
  $ce_i \leftarrow \text{CrossEntropy}(Z_i, L_i)$
  $p_t \leftarrow \exp(-ce_i)$
  $\text{focal}_i \leftarrow (1 - p_t)^\gamma$
  $\text{loss}_i \leftarrow w_{c_i} \times \text{focal}_i \times ce_i$
end for
$\mathcal{L}_{cb\_focal} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \text{loss}_i$

$-\mathbf{x}_i$ are valid eigenvectors for λ_i), we randomly flip the signs of the eigenvectors during training. This technique helps reduce the model's sensitivity to the sign of the eigenvectors and improves its generalization ability. Finally, the combined embeddings are passed into the Transformer encoder, where they are transformed into final embeddings $\{\mathbf{h}_i^t\}$ ready for downstream quality estimation via structure-aware masked node modeling and edge prediction.

3.3 Pretraining Tasks

In this work, we adopt two self-supervised pretraining tasks, structure-aware masked node modeling and edge prediction, to obtain structure-informed representations for RTL design quality estimation. Unlike prior methods that directly masks raw node features or edges in the original graph (Hou et al., 2022; Li et al., 2023), we apply masking at the level of post-GNN context-aware embeddings. This design choice is particularly important for computational graphs like CDFGs, where each node carries strict functional semantics. Masking raw nodes or edges in such graphs can introduce ambiguity, as multiple valid replacements exist. By instead masking the post-GNN embeddings, which already encode surrounding semantics, the model can utilize rich context information from unmasked regions to reconstruct the masked content. This preserves the structural and computational integrity of the original graph, enabling more faithful reconstruction.

Structure-Aware Masked Node Modeling. In this task, we randomly mask 20% of the post-GNN node embeddings by replacing them with a special learnable `[MASK]` embedding and use the Transformer encoder to recover the masked nodes by predicting their original node types, which is formulated as a 32-class classification problem. A complete list of node types is provided in Appendix A.2. One key challenge in this task is the severe class imbalance. For example, operator nodes are significantly underrepresented compared to storage elements such as wires and registers. To mitigate this, we adopt two strategies. First, we apply stratified masking, which ensures that at least m nodes from each class are included in the masked set during each training iteration. This helps the model to learn meaningful representations even for infrequent node types. The full procedure is described in Algorithm 1. Second, instead of using the classic cross entropy loss, we adopt the class-balanced focal loss (Cui et al., 2019), which adjusts the loss based on the effective number of samples for each class, and puts more focus on hard-to-classify examples. The details of the class-balanced focal loss are presented in Algorithm 2. The loss function for this task is denoted as $\mathcal{L}_{mnm} = \mathcal{L}_{cb\_focal}$.

Edge Prediction. When the post-GNN embeddings are input into the Transformer encoder, the graph's connectivity is lost, which can be viewed as if all edges are masked. For each training iteration, we randomly select 20% of the actual edges as positive samples and an equal number of non-existing edges as negative samples, and formulate the edge prediction task as a binary classi-

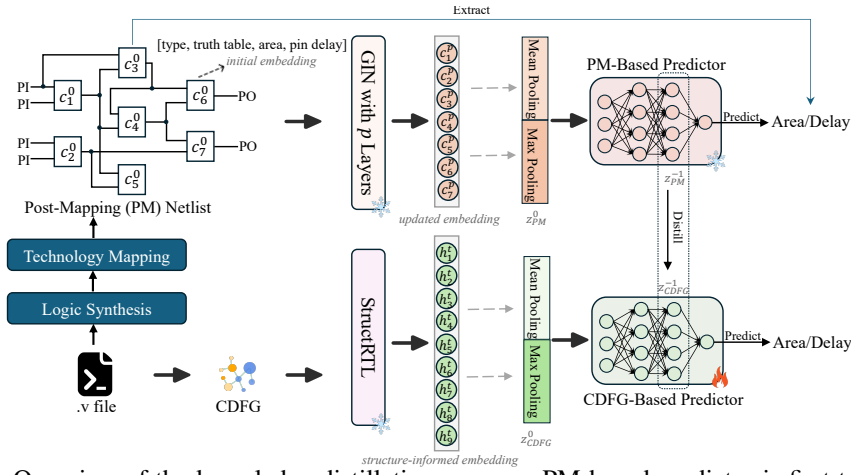


Figure 3: Overview of the knowledge distillation process. PM-based predictor is first trained and frozen. It then guides the learning of the CDFG-based predictor, which performs quality estimation while aligning its final-layer activations with those of the PM-based predictor.

fication problem. Specifically, we concatenate the final embeddings of the source and target nodes and use a 3-layer multi-layer perceptron (MLP) to predict whether an edge exists between them. We employ cross entropy loss for this task, and denote the loss as $\mathcal{L}_{ep}$.

The total loss for pretraining is:

$$\mathcal{L}_{pre} = \alpha \mathcal{L}_{mnm} + (1 - \alpha)\, \mathcal{L}_{ep} \tag{4}$$

where α balances these two pretraining tasks. In this work, we set $\alpha = 0.5$. After pretraining, the validation accuracies for structure-aware masked node modeling and edge prediction are 82.27% and 95.77%, respectively.

3.4 Quality Estimation

After pretraining, StructRTL outputs structure-informed embeddings ready for RTL design quality estimation. This task is formulated as a regression problem, where node-level embeddings are first aggregated into a graph-level representation using joint mean and max pooling. The resulting graph embedding is then passed through a 3-layer MLP to predict quality metrics such as area and delay. Given that these metric values have large magnitudes and exhibit significant variance across designs, we apply a logarithm transformation to these values to make the target distribution more suitable for model learning. This transformation does not affect the practical performance of the model, as we are more concerned with the relative quality of different designs. For this task, we use the log-cosh loss (Saleh & Saleh, 2022), which is robust to outliers, and denote the loss function as $\mathcal{L}_{qe}$.

For evaluation, we use four standard regression metrics: mean absolute error (MAE), mean absolute percentage error (MAPE), coefficient of determination (R^2), and root relative squared error (RRSE). Given predicted values $\hat{y}_i$ and ground truth values y_i for $i \in [1, N]$, these metrics are defined as:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{5}$$

$$\text{MAPE} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{\hat{y}_i - y_i}{y_i} \right| \tag{6}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2} \tag{7}$$

$$\text{RRSE} = \sqrt{\frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}} \tag{8}$$

where $\bar{y}$ denotes the mean of the ground truth values.

Table 1: Performance comparison of various methods for post-synthesis area and delay prediction without incorporating the knowledge distillation strategy. Notably, StructRTL significantly outperforms both graph-based and LLM-based baselines.

w/o KD	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
Graph-Based Baselines								
Graph-XGBoost	0.9267	0.19	0.3987	0.7754	0.6384	0.12	0.3362	0.8147
Graph-GNN	0.5497	0.09	0.5857	0.6437	0.7327	0.13	0.6639	0.5797
LLM-Based Baselines								
CodeV-DS-6.7B	0.8967	0.17	0.4862	0.6973	0.6403	0.12	0.3905	0.7807
CodeV-CL-7B	0.7982	0.15	0.5755	0.6515	0.5620	0.10	0.5174	0.6947
CodeV-QW-7B	0.7229	0.13	0.6353	0.6039	0.5340	0.09	0.5277	0.6872
Ours								
StructRTL (w/o $\mathcal{L}_{mnm}$)	0.3900	0.07	0.7249	0.5245	0.5730	0.11	0.7473	0.5027
StructRTL (w/o $\mathcal{L}_{ep}$)	0.4035	0.07	0.7018	0.5460	0.5902	0.11	0.7368	0.5130
StructRTL	0.3649	0.06	0.7463	0.5037	0.5414	0.10	0.7630	0.4868

3.5 Knowledge Distillation

Directly estimating quality metrics from the RTL stage can be challenging, as there remains a significant gap between RTL designs and the PM netlists from which area and delay are actually measured. To bridge this gap, we incorporate a knowledge distillation (KD) strategy that transfers low-level insights from PM netlists into the CDFG predictor. As illustrated in Figure 3, we synthesize and map RTL designs into PM netlists using Yosys (Wolf et al., 2013) and ABC (Brayton & Mishchenko, 2010) with the SkyWater 130nm technology library (Edwards, 2020). We then extract area and delay metrics from the resulting netlists. To model the PM netlist, we initialize each cell's embedding with a concatenation of its one-hot cell type encoding, logic truth table, and associated area and pin delay information. These embeddings are processed using a GIN, followed by joint mean and max pooling to obtain a graph-level representation, which is then passed to a 3-layer MLP for quality estimation. Since the area and delay values are derived directly from these netlists, the quality prediction at this stage is naturally more accurate than at the RTL level. To improve RTL-stage predictions, we perform KD by aligning the final-layer activations of the CDFG-based predictor z^{-1}_{CDFG} with those of the PM-based predictor z^{-1}_{PM}. The KD loss is defined as:

$$\mathcal{L}_{kd} = \tau \cdot \mathcal{L}_{cos}(z^{-1}_{\text{CDFG}}, z^{-1}_{\text{PM}}) + (1-\tau) \cdot \mathcal{L}_{mse}(z^{-1}_{\text{CDFG}}, z^{-1}_{\text{PM}}) \tag{9}$$

where $\mathcal{L}_{cos}$ is the cosine similarity loss, $\mathcal{L}_{mse}$ is the mean squared error (MSE) loss, and τ balances the contribution of the two terms. We set $\tau = 0.7$ in our experiments. This alignment encourages the CDFG-based predictor to internalize fine-grained low-level insights learned from the gate-level view, thereby improving its quality estimation performance.

During training, we first train the PM-based predictor and freeze its parameters. It is then used to guide learning of CDFG-based predictor. The total loss for training the CDFG-based predictor is:

$$\mathcal{L}_{total} = \mu \mathcal{L}_{qe} + (1-\mu)\mathcal{L}_{kd} \tag{10}$$

where we set $\mu = 0.5$ in our experiments. Note that the PM-based predictor is only used during training for supervision; during validation, only the CDFG-based predictor is retained.

4 Experiments

In this section, we conduct extensive experiments to show the effectiveness of our proposed framework, StructRTL. We begin by introducing the baselines, experimental settings, and dataset construction process. We then report comparative results on post-synthesis area and delay prediction tasks. Additionally, we perform an ablation study to demonstrate the importance of the two pretraining tasks in learning structure-informed representations.

Table 2: Performance comparison of various methods for post-synthesis area and delay prediction with the incorporation of the knowledge distillation strategy. While knowledge distillation enhances the performance of all methods, StructRTL consistently achieves the best results, significantly outperforming the baselines, especially in delay prediction.

w/ KD	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
Teacher								
PM-based Predictor	0.2982	0.05	0.9334	0.2581	0.1688	0.03	0.9484	0.2272
Graph-Based Baselines								
Graph-GNN	0.4689	0.09	0.7954	0.4523	0.2926	0.05	0.8113	0.4344
LLM-Based Baselines								
CodeV-DS-6.7B	0.4896	0.09	0.7928	0.4552	0.3787	0.07	0.7235	0.5258
CodeV-CL-7B	0.4192	0.08	0.8225	0.4213	0.3208	0.06	0.7696	0.4800
CodeV-QW-7B	0.4397	0.08	0.8174	0.4273	0.3284	0.06	0.7687	0.4809
Ours								
StructRTL (w/o $\mathcal{L}_{mnm}$)	0.4015	0.07	0.8557	0.3799	0.2446	0.04	0.8796	0.3470
StructRTL (w/o $\mathcal{L}_{ep}$)	0.4071	0.07	0.8480	0.3899	0.2568	0.04	0.8654	0.3669
StructRTL	0.3856	0.07	0.8676	0.3639	0.2381	0.04	0.8872	0.3359

4.1 Baselines

In this work, we primarily compare StructRTL with VeriDistill (Moravej et al., 2025), a recent state-of-the-art method that leverages LLMs specifically trained for Verilog generation (Pei et al., 2024; Zhao et al., 2025; Liu et al., 2025b) to derive embeddings from RTL code. VeriDistill has shown strong performance on RTL design quality estimation tasks over a large-scale dataset of more than 10,000 designs. Following their setup, we adopt the open-source Verilog LLM CodeV (Zhao et al., 2025), which includes three variants: CodeV-DS-6.7B, CodeV-CL-7B, and CodeV-QW-7B, finetuned from DeepSeek-Coder (Guo et al., 2024), CodeLlama (Roziere et al., 2023), and Code-Qwen (Bai et al., 2023), respectively. These models process the RTL code as raw token sequences, generate token-level embeddings, and apply mean pooling followed by a 3-layer MLP to produce final quality predictions. In addition, we implement two graph-based baselines following prior work (Sengupta et al., 2022). First, we implement Graph-XGBoost, where we extract hand-crafted statistical features from the CDFG, including the total bits per node type, the frequency of each node type, the average wire width, and the length of the longest combinational logic path. These features are concatenated and fed into an XGBoost (Chen & Guestrin, 2016) regressor, following the configuration described in their original work. Second, we implement Graph-GNN, which uses a GNN to directly process the CDFG without any self-supervised pretraining. The resulting node embeddings are aggregated via joint mean and max pooling and passed through a 3-layer MLP for quality estimation. This setup mirrors the architecture used in our PM-based predictor, with further details about the model presented in the following subsection.

4.2 Experimental Setup

In this subsection, we detail the model configurations and training parameters. StructRTL uses an 8-layer GIN, followed by an 8-layer Transformer encoder, each layer containing 4 attention heads. The model is trained for 2,000 epochs with a batch size of 16, using the AdamW (Loshchilov & Hutter, 2017) optimizer with a learning rate of 2e-5 and weight decay of 1e-4. The PM-based predictor employs a 20-layer GIN with residual connections, trained for 1,000 epochs with a batch size of 256. It is optimized with the Adam (Kinga et al., 2015) optimizer at a learning rate of 1e-4 and weight decay of 1e-5. The 3-layer MLP quality estimators are trained for 600 epochs with a batch size of 256, using the same optimizer settings as the PM-based predictor. Under these configurations, all models have been trained until full convergence. The experiments are conducted on a cluster of 5 L40 GPUs, each with 46GB of memory. Further details on the model training complexity analysis can be found in Appendix A.3.

4.3 Dataset Construction

For dataset construction, we begin by collecting some existing Verilog datasets, including VeriGen (Thakur et al., 2024), RTLCoder (Liu et al., 2024), MGVerilog (Zhang et al., 2024), and DeepCircuitX (Li et al., 2025), which include designs obtained from GitHub repositories, textbook examples, and designs generated by LLMs. We filter out designs that cannot be converted into CDFGs or synthesized into PM netlists. Additionally, we use Verilator (Snyder, 2004) to simulate these designs with randomly generated input patterns, retaining only those with meaningful outputs. This ensures that the post-synthesis area and delay values are well-defined. The resulting dataset consists of 13,200 designs, which is split into training and validation sets with a ratio of 0.8:0.2. For further details on the dataset statistics and label distribution, please refer to Appendix A.4.

4.4 Experimental Results

In this subsection, we present a comparative analysis of post-synthesis area and delay prediction between StructRTL and baseline methods. Among all evaluation metrics, we primarily focus on the R^2 score, as it provides insight into the relative quality of two designs, which is crucial for guiding optimization efforts. As shown in Table 1, StructRTL significantly outperforms both graph-based and LLM-based baselines without incorporating the knowledge distillation strategy, achieving a 0.1 increase in the R^2 score over the previous state-of-the-art methods for both area and delay prediction. Notably, Graph-XGBoost performs the worst among all methods, highlighting the limited expressiveness of hand-crafted features. Compared to Graph-GNN, which performs quality estimation in an end-to-end manner, StructRTL demonstrates superior performance, underscoring the effectiveness of structure-aware graph self-supervised learning. This approach helps the model learn structure-informed representations that are suitable and generalizable for downstream quality estimation tasks. Besides, StructRTL shows a significant advantage over LLM-based baselines, especially in the delay prediction task. This is primarily due to the fact that, unlike the token-based view, the CDFG exposes the design's structural semantics more explicitly, providing richer cues for learning complex patterns that impact the design's quality. This is particularly important for delay prediction, which is closely tied to the design's structure. Table 2 presents the results when the knowledge distillation strategy is incorporated. As shown, knowledge distillation improves the performance of all methods, highlighting the importance of cross-stage supervision. However, StructRTL consistently achieves the best results, further validating the effectiveness of structural learning.

Ablation Study. We conduct an ablation study to assess the impact of the two pretraining tasks in StructRTL. Specifically, we remove the structure-aware masked node modeling and edge prediction, resulting in two variants: StructRTL (w/o $\mathcal{L}_{mnm}$) and StructRTL (w/o $\mathcal{L}_{ep}$), respectively. As shown in Tables 1 and 2, removing either task leads to performance degradation, demonstrating the importance of both tasks in learning the structure-aware representations necessary for accurate RTL design quality estimation. We provide additional performance comparisons of StructRTL on combinational and sequential circuits in Appendix A.5. Experiments evaluating the generalization ability of StructRTL across functionality and design sizes are presented in Appendices A.6 and A.7, respectively. The performance of LLM-based methods using CDFG representations is reported in Appendix A.8. Qualitative differences between StructRTL and LLM-based approaches are discussed in Appendix A.9. Finally, the limitations of StructRTL are summarized in Appendix A.10.

5 Conclusion

In this work, we introduce StructRTL, a novel structure-aware graph self-supervised learning framework to improve RTL design quality estimation. By leveraging CDFG, StructRTL incorporates two tailored pretraining tasks, structure-aware masked node modeling and edge prediction, to learn structure-informed representations for downstream quality estimation. StructRTL outperforms prior methods that either rely on shallow hand-crafted features or fail to capture the structural semantics of RTL designs. To further boost performance, we integrate a knowledge distillation strategy that transfers low-level insights from PM netlists into the CDFG-based predictor, enabling StructRTL to achieve state-of-the-art results in area and delay prediction. Our findings highlight the effectiveness of combining structural representation learning with cross-stage supervision and open new directions for advancing RTL design quality estimation in EDA workflows.

Ethics Statement

We have read the ICLR Code of Ethics[1] and are committed to full compliance. All datasets used in this work are obtained from open-source repositories under appropriate licenses, and their processing is carried out exclusively with open-source tools.

Reproducibility Statement

To ensure reproducibility, we provide comprehensive documentation and resources. Section 3.1 outlines the CDFG construction process, while Section 4.3 describes the dataset sources and validation procedure. Detailed experimental configurations are provided in Section 4.2. In addition, we release the source code of StructRTL along with the training and evaluation datasets through an anonymous GitHub repository: `https://anonymous.4open.science/r/StructRTL-CB09/`.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pp. 24–40. Springer, 2010.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Xiangli Chen, Yuehua Meng, and Gang Chen. Incremental verilog parser. In *2023 International Symposium of Electronics Design Automation (ISEDA)*, pp. 236–240. IEEE, 2023.

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9268–9277, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pp. 4171–4186, 2019.

R Timothy Edwards. Google/skywater and the promise of the open pdk. In *Workshop on Open-Source EDA Technology*, 2020.

Ege Erdil and Tamay Besiroglu. Algorithmic progress in computer vision. *arXiv preprint arXiv:2212.05153*, 2022.

Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Masterrtl: A pre-synthesis ppa estimation framework for any rtl design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[1] `https://iclr.cc/public/CodeOfEthics`

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, pp. 4116–4126. PMLR, 2020.

Anson Ho, Tamay Besiroglu, Ege Erdil, David Owen, Robi Rahman, Zifan C Guo, David Atkinson, Neil Thompson, and Jaime Sevilla. Algorithmic progress in language models. *Advances in Neural Information Processing Systems*, 37:58245–58283, 2024.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 594–604, 2022.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Diederik Kinga, Jimmy Ba Adam, et al. A method for stochastic optimization. In *International conference on learning representations (ICLR)*, volume 5. California;, 2015.

Jintang Li, Ruofan Wu, Wangbin Sun, Liang Chen, Sheng Tian, Liang Zhu, Changhua Meng, Zibin Zheng, and Weiqiang Wang. What's behind the mask: Understanding masked graph modeling for graph autoencoders. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1268–1279, 2023.

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, et al. Deepcircuitx: A comprehensive repository-level dataset for rtl code understanding, generation, and ppa analysis. *arXiv preprint arXiv:2502.18297*, 2025.

Mingjie Liu, Yun-Da Tsai, Wenfei Zhou, and Haoxing Ren. Craftrtl: High-quality synthetic data generation for verilog code models with correct-by-construction non-textual representations and targeted code repair. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

Yi Liu, XU Changran, Yunhao Zhou, Zeju Li, and Qiang Xu. Deeprtl: Bridging verilog understanding and generation with a unified representation model. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Yi Liu, Hongji Zhang, Yunhao Zhou, Zhengyuan Shi, Changran Xu, and Qiang Xu. Deeprtl2: A versatile model for rtl-related tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025c.

Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *IEEE transactions on knowledge and data engineering*, 35(6): 5879–5900, 2022.

Daniela Sánchez Lopera, Lorenzo Servadei, Vishwa Priyanka Kasi, Sebastian Prebeck, and Wolfgang Ecker. Rtl delay prediction using neural networks. In *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pp. 1–7. IEEE, 2021.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Reza Moravej, Saurabh Bodhe, Zhanguang Zhang, Didier Chetelat, Dimitrios Tsaras, Yingxue Zhang, Hui-Ling Zhen, Jianye Hao, and Mingxuan Yuan. The graph's apprentice: Teaching an llm low level knowledge for circuit quality estimation. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence*, 2025.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: controlled verilog generation with discriminative guidance. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 40145–40153, 2024.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Resve A Saleh and AK Saleh. Statistical properties of the log-cosh loss function used in machine learning. *arXiv preprint arXiv:2208.04564*, 2022.

Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. How good is your verilog rtl code? a quick answer from machine learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

Kristina P Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. *IEEE access*, 8:80716–80727, 2020.

Wilson Snyder. Verilator and systemperl. In *North American SystemC Users' Group, Design Automation Conference*, volume 79, pp. 122–148, 2004.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *The Seventh International Conference on Learning Representations*, 2019.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33: 5812–5823, 2020.

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *International conference on machine learning*, pp. 12121–12132. PMLR, 2021.

Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan Celine Lin. Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5. IEEE, 2024.

Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Muxin Song, Yinan Xu, Ziyuan Nan, Mingju Gao, Tianyun Ma, Lei Qi, et al. Codev: Empowering llms with hdl generation through multi-level summarization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. Primal: Power inference using machine learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

A Appendix

A.1 The Use of Large Language Models (LLMs)

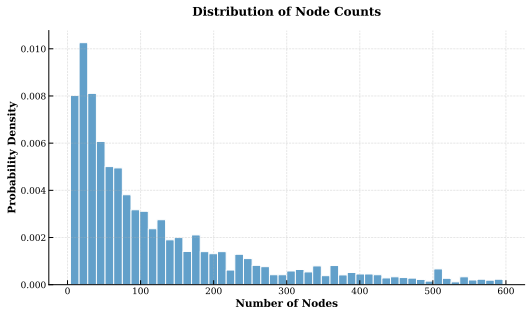
We acknowledge the use of large language models (LLMs) in this work. In particular, the LLM-powered programming tool Cursor[2] assisted with implementation, while GPT-5[3] was used during manuscript preparation to correct grammar and improve phrasing.

A.2 CDFG Details

In Section 3.1, we describe the process of constructing the CDFG. In this section, we provide additional details, including a list of all node types along with their respective counts, ratios, and descriptions in our dataset, as shown in Table 3. The CDFG contains a total of 32 distinct node types, with significant variation in their frequencies. For instance, `Wire` nodes are the most common, comprising 40.05% of the dataset, while Unary Operator nodes, such as `LNot` and `Not`, make up a small portion of the dataset, with frequencies as low as 0.00% to 1.18%. For the structure-aware masked node modeling pretraining task, we formulate it as a 32-class classification problem. From Table 3, we can observe a severe class imbalance, particularly with the operation node types, which are underrepresented compared to storage elements such as `Wire` and `Reg` nodes. To mitigate this issue, we propose two strategies: stratified masking and class-balanced focal loss, which are described in the Section 3.3.

Range	Count
$[0, 600)$	10645
$[600, 5000)$	1148
$[5000, 10000)$	542
$[10000, 15000)$	497
$[15000, 20000)$	368

(a) Coarse-grained distribution of node counts for all designs in the dataset.



(b) Fine-grained distribution of node counts for designs with fewer than 600 nodes in the CDFG.

Figure 4: Node count distributions in the dataset.

A.3 Model Training Complexity Analysis

As in Section 4.2, our experiments are conducted on a cluster with five NVIDIA L40 GPUs, each equipped with 46GB of memory. Using the training setting described in Section 4.2, StructRTL can be trained to convergence on a single GPU in roughly 40 hours, with a memory footprint of about 42GB. Importantly, the batch size can be reduced to accommodate GPUs with smaller memory capacity, making the model practical for a wide range of academic and industrial users.

We also note that simpler GNN-based models, which omit the Transformer encoder and therefore have lower computational overhead, can be trained with even fewer resources. However, our experiments show that such simplified architectures experience substantial performance degradation. This highlights a fundamental trade-off: while simpler GNN-only pipelines are more lightweight, their predictive accuracy is significantly lower. StructRTL, despite its multi-stage architecture, remains a modestly sized model at only 30M parameters (114 MB in fp32), and thus is still affordable for typical research and industrial settings while delivering substantially stronger performance.

Finally, compared to LLM-based approaches, which typically require around 7B parameters, StructRTL introduces far fewer deployment barriers. Although LLM-based pipelines may appear conceptually simpler, their enormous model sizes and resource requirements make them considerably more challenging to adopt in practice than our method.

[2] `https://cursor.com`
[3] `https://chatgpt.com`

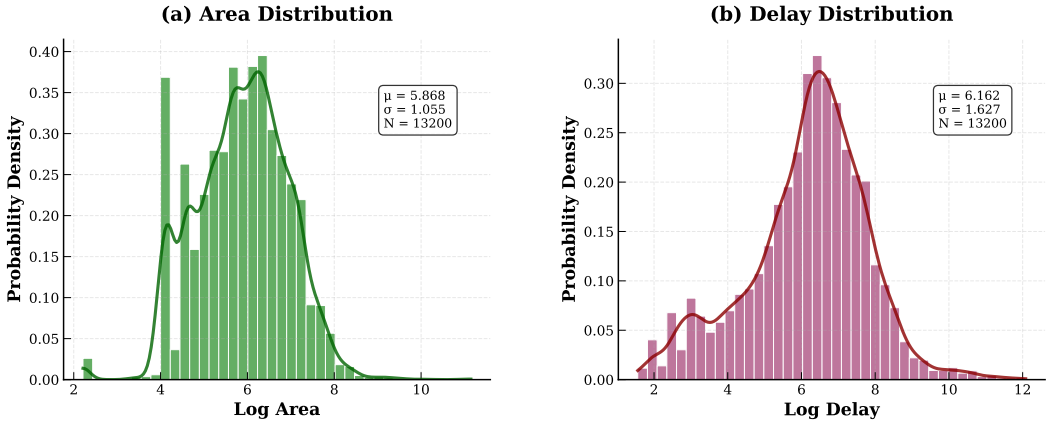


Figure 5: The distribution of area and delay values in the dataset, with logarithmic transformation applied to both.

A.4 Dataset Statistics

In this section, we provide additional details on the dataset statistics and label distributions. Our dataset consists of 13,200 designs, with CDFG node counts ranging from 4 to 19,169. As described in Section 4.3, the raw RTL data is sourced from four distinct Verilog datasets: VeriGen (Thakur et al., 2024), RTLCoder (Liu et al., 2024), MGVerilog (Zhang et al., 2024), and DeepCircuitX (Li et al., 2025). These datasets include designs collected from GitHub repositories, textbook examples, and designs generated by LLMs. The resulting RTL designs cover a broad spectrum of digital design functionalities that are foundational across various sectors of hardware design, from basic data processing and logical operations to complex systems involving memory management, communication interfaces, state control, multimedia applications, fault-tolerant designs, *etc.* This diversity ensures that our dataset provides a comprehensive and fair evaluation of models' quality estimation capabilities. Table 4a presents the coarse-grained distribution of node counts across the entire dataset, revealing that the majority of designs contain fewer than 600 nodes. To provide a closer look at this majority subset, Figure 4b shows a more fine-grained distribution of node counts specifically for designs with fewer than 600 nodes in the CDFG.

Figure 5 illustrates the distributions of post-synthesis area and delay values across all designs in the dataset. Given the large magnitudes and significant variance of these metrics, we apply a logarithmic transformation to normalize the target distributions and facilitate more effective model training. This transformation does not impact practical model performance, as our primary focus lies in capturing the relative quality of different designs rather than their absolute values.

A.5 Performance Comparison of Combinational and Sequential Circuits

In this section, we evaluate the performance difference of StructRTL on combinational and sequential circuits, which provides deeper insight into its behavior. Following the original experimental setting, we partition the validation set into combinational and sequential subsets and evaluate the model on each category separately. As shown in Table 4, StructRTL generally performs better on combinational circuits than on sequential circuits across all quality estimation tasks and evaluation metrics, both with and without the incorporation of knowledge distillation. The performance gap is more pronounced for delay prediction than for area prediction. One plausible explanation is that combinational circuits typically exhibit simpler structural characteristics, as they do not involve inter-cycle data dependencies. Their CDFGs are therefore simpler and more direct, making it easier for the model to extract meaningful structural cues. In contrast, sequential circuits incorporate inter-cycle data flows, which complicate the underlying timing behavior and make the identification of critical paths more challenging. This added complexity likely contributes to the larger performance difference observed in delay prediction.

A.6 Generalization Across Functionality

In this section, we evaluate the functional generalization capability of our method. Instead of using the random train/validation split adopted in the original experimental setting, we partition the dataset

Table 3: Node types in the CDFG, along with their respective counts, ratios, and descriptions.

	Node Type	Count	Ratio	Description
Unary Operator	LNot	33521	1.18%	Logical NOT
	Not	24579	0.86%	Bitwise NOT
	URxor	119	0.00%	Unary reduction XOR
	URand	739	0.03%	Unary reduction AND
	UOr	5285	0.19%	Unary reduction OR
Binary Operator	Lt	2852	0.10%	Less than
	Le	2285	0.08%	Less than or equal
	Gt	1701	0.06%	Greater than
	Ge	3139	0.11%	Greater than or equal
	Add	30179	1.06%	Addition
	Sub	6177	0.22%	Subtraction
	Mul	3325	0.12%	Multiplication
	Div	198	0.01%	Division
	Mod	142	0.00%	Modulo operation
	ShiftLeft	261	0.01%	Logical left shift
	ShiftRight	487	0.02%	Logical right shift
	And	29679	1.04%	Logical AND
	Or	8303	0.29%	Logical OR
	Eq	189418	8.17%	Equality
	Neq	1350	0.05%	Inequality
	BitAnd	51656	1.82%	Bitwise AND
	BitOr	23330	0.82%	Bitwise OR
	BitXor	101147	3.56%	Bitwise XOR
	BitNXor	67	0.00%	Bitwise XNOR
N-ary Operator	PartSelect	124144	4.36%	Bit-range extraction
	Concat	52727	1.85%	Bitwise concatenation
Cond	Cond	571656	20.09%	Ternary conditional operator
Wire	Wire	1139415	40.05%	Wire declaration or reference
Const	Const	232450	8.17%	Constant literal value
I/O	Input	73853	2.60%	Input port
	Output	64047	2.25%	Output port
Reg	Reg	66680	2.34%	Register declaration

based on the underlying functionality of the designs. To achieve this, we first generate embeddings for each RTL design using text-embedding-3-large (Neelakantan et al., 2022) and then perform k-means clustering (Sinaga & Yang, 2020) with $k = 50$. This procedure groups functionally similar designs into the same cluster while separating those with distinct behaviors into different clusters. We then divide the dataset into training and validation sets at a ratio of 0.8:0.2 using *disjoint* clusters, thereby creating an explicit functional shift between the two sets.

As shown in Tables 5 and 6, StructRTL experiences only a slight performance drop compared with the original setting. Despite this functional shift, StructRTL maintains a strong predictive capability and continues to outperform prior state-of-the-art LLM-based baselines by a substantial margin. Furthermore, the incorporation of knowledge distillation yields additional improvements, highlighting the effectiveness of cross-stage supervision.

A.7 Generalization Across Design Sizes (Scalability)

In this section, we conduct an experiment to evaluate the scalability of our method. Rather than using the random train/validation splits as in the original experimental setting, we instead partition the dataset by node count. Specifically, we use the smallest 80% of the designs as the training set,

Table 4: Performance comparison of StructRTL on the combinational and sequential validation subsets, with and without the incorporation of knowledge distillation. StructRTL generally achieves better performance on area prediction than on delay prediction.

	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
w/o KD								
Combinational	0.3579	0.06	0.7503	0.4997	0.5262	0.10	0.7923	0.4557
Sequential	0.3724	0.06	0.7388	0.5111	0.5562	0.10	0.7573	0.4926
w/ KD								
Combinational	0.3753	0.07	0.8740	0.3550	0.2277	0.04	0.9002	0.3159
Sequential	0.3952	0.07	0.8534	0.3829	0.2478	0.04	0.8681	0.3632

Table 5: Comparison of the functional generalization performance of different methods for post-synthesis area and delay prediction without knowledge distillation. StructRTL consistently outperforms the LLM-based baselines.

w/o KD (Split by Function)	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
LLM-Based Baselines								
CodeV-DS-6.7B	0.9636	0.20	0.4493	0.7421	0.6659	0.12	0.3407	0.8120
CodeV-CL-7B	0.8364	0.17	0.5144	0.6969	0.5854	0.11	0.4394	0.7487
CodeV-QW-7B	0.7840	0.15	0.5722	0.6541	0.5691	0.11	0.4505	0.7413
Ours								
StructRTL	0.3921	0.07	0.7237	0.5256	0.5688	0.11	0.7510	0.4990

and the remaining 20% containing larger designs as the validation set. Under this configuration, the training set is composed primarily of designs with fewer than 600 nodes, whereas the validation set includes many designs with more than 10,000 nodes.

As shown in Tables 7 and 8, StructRTL exhibits a degree of performance degradation compared with the original experimental setting. Nonetheless, it still demonstrates strong generalization from small to large designs for both area and delay prediction. Its performance remains competitive and superior to prior state-of-the-art LLM-based approaches, which generally struggle to generalize beyond the scale of their training data. In addition, knowledge distillation provides further improvements, which underscores the effectiveness of cross-stage supervision.

Additionally, we observe that delay prediction generalizes better than area prediction when the model is trained on small circuits but evaluated on substantially larger ones. One plausible explanation is that area typically grows roughly proportionally with circuit size, meaning that the area values associated with very large circuits can fall far outside the distribution of values seen in smaller circuits during training, which affects generalization. In contrast, the delay of a large circuit is often dominated by the critical path within one or a few subcircuits, particularly for sequential designs. Consequently, structural patterns relevant to critical path information can be learned from small circuits and transferred effectively to larger ones. From this perspective, a potential direction for further improving scalability to even larger designs is to partition circuits at register boundaries to obtain subcircuits and perform area and delay prediction at the subcircuit level. The overall area could then be obtained by *summation*, and the overall delay by *taking the maximum* among the subcircuits. We leave this exploration to future work.

A.8 LLM-Based Methods on CDFG Representation

In this section, we conduct an experiment where LLMs are used to directly process CDFG representations to further demonstrate the superiority of our method over the LLM-based ones. Importantly,

Table 6: Comparison of the functional generalization performance of different methods for post-synthesis area and delay prediction with knowledge distillation. StructRTL consistently outperforms the LLM-based baselines.

w/ KD (Split by Function)	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
Teacher								
PM-based Predictor	0.3081	0.05	0.9290	0.2665	0.1928	0.03	0.9400	0.2450
LLM-Based Baselines								
CodeV-DS-6.7B	0.5600	0.10	0.7670	0.4827	0.4067	0.07	0.6931	0.5540
CodeV-CL-7B	0.4374	0.08	0.7813	0.4677	0.3590	0.06	0.7200	0.5292
CodeV-QW-7B	0.4514	0.08	0.7769	0.4723	0.3690	0.06	0.7141	0.5347
Ours								
StructRTL	0.3866	0.07	0.8259	0.4173	0.2768	0.05	0.8575	0.3775

Table 7: Comparison of the generalization capability of various methods across design sizes for post-synthesis area and delay prediction without knowledge distillation. StructRTL consistently achieves superior performance relative to LLM-based baselines.

w/o KD (Split by Size)	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
LLM-Based Baselines								
CodeV-DS-6.7B	1.0621	0.22	0.3276	0.8200	0.6059	0.11	0.1783	0.9065
CodeV-CL-7B	0.9340	0.19	0.3453	0.8091	0.7133	0.13	0.1146	0.9410
CodeV-QW-7B	0.8724	0.16	0.4098	0.7682	0.5942	0.10	0.3267	0.8205
Ours								
StructRTL	0.5316	0.09	0.5567	0.6658	0.5745	0.10	0.6753	0.5698

the CDFG used in StructRTL is a graph-based representation that cannot be directly fed to LLMs. Therefore, for LLM-based baselines, we use the corresponding `.dot`-format CDFG, which provides a textual serialisation of the same underlying graph. This allows us to evaluate whether LLMs, when given the textual CDFG representation, can serve as competitive predictors of area and delay.

Using these `.dot`-form CDFGs as input, we generate LLM-derived CDFG embeddings and evaluate their predictive performance under the original experimental setting. The results are summarized in Tables 9 and 10, which show that, even when replacing RTL code with CDFG inputs, StructRTL continues to outperform all LLM-based baselines by a substantial margin, both with and without knowledge distillation, on area and delay prediction.

We further observe that replacing RTL code with `.dot`-format CDFGs degrades the performance of VeriDistill. This is likely because CDFG-like representations are largely absent from the corpora used to pre-train these LLMs, leaving the models with limited prior exposure or inductive bias for such structures. As a result, their ability to extract meaningful information from textualized CDFGs is diminished, leading to weaker downstream predictive performance.

A.9 Qualitative Differences Between StructRTL and LLM-Based Methods

In this section, we explain why each pretraining component of StructRTL contributes to the observed performance improvements and how the resulting structural representations differ qualitatively from token-based embeddings used in LLM-based methods. StructRTL incorporates two pretraining tasks, structure-aware masked node modeling and edge prediction, to encourage the model to learn structure-informed representations from CDFGs for RTL quality estimation. Conceptually, both tasks compel the model to decipher the underlying relational patterns within the graph.

Table 8: Comparison of the generalization capability of various methods across design sizes for post-synthesis area and delay prediction with knowledge distillation. StructRTL consistently outperforms the LLM-based baselines.

w/ KD (Split by Size)	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
Teacher								
PM-based Predictor	0.3628	0.06	0.8100	0.4359	0.2480	0.04	0.8966	0.3216
LLM-Based Baselines								
CodeV-DS-6.7B	0.5150	0.09	0.6382	0.6015	0.3845	0.06	0.5142	0.6970
CodeV-CL-7B	0.5424	0.10	0.6037	0.6295	0.4260	0.07	0.4029	0.7727
CodeV-QW-7B	0.5024	0.09	0.6397	0.6002	0.3749	0.06	0.5444	0.6750
Ours								
StructRTL	0.4441	0.08	0.6838	0.5623	0.3572	0.05	0.7443	0.5057

Table 9: Performance comparison of different methods for post-synthesis area and delay prediction without knowledge distillation, where LLMs directly process textualized CDFG representations. StructRTL significantly outperforms the LLM-based baselines.

w/o KD (LLM on CDFG)	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
LLM-Based Baselines								
CodeV-DS-6.7B	0.9420	0.19	0.3980	0.7759	0.7260	0.13	0.3257	0.8212
CodeV-CL-7B	0.9039	0.18	0.4410	0.7477	0.6305	0.12	0.3671	0.7956
CodeV-QW-7B	0.7917	0.15	0.5639	0.6604	0.5961	0.11	0.4250	0.7583
Ours								
StructRTL	0.3649	0.06	0.7463	0.5037	0.5414	0.10	0.7630	0.4868

In the structure-aware masked node modeling task, we randomly mask a subset of post-GNN, context-aware node embeddings and train the Transformer to recover their original node types. Correctly predicting a masked operator node (*e.g.*, distinguishing an `Add` from a `Sub` or `Mul`) is only possible if the model has learned how that node participates in the surrounding data and control flow. This task therefore captures not only node-level semantics relevant to area (since different operator types imply differing implementation costs), but also structural characteristics relevant to delay (because operator types affect the architectural depth and critical-path behavior of the design). In essence, this task forces the model to learn both local semantics and the broader structural roles that nodes play within the CDFG.

The edge prediction task complements this by requiring the model to determine whether a connection exists between two nodes based on their learned embeddings. Recovering such connectivity forces the model to capture global structural properties (*e.g.*, reconvergence and pipeline depth) that strongly affect both area and delay. Edge prediction thus complements masked node modeling by explicitly regularizing topological understanding at the graph level.

Together, these two pretraining objectives shape the learned representations to reflect both semantic and structural characteristics of CDFGs. As shown in Tables 1 and 2, removing either pretraining task yields consistent degradation on both area and delay prediction, demonstrating that each task provides distinct yet complementary structural cues.

In contrast, token-based embeddings derived from LLMs are trained predominantly through next-token prediction on source code. This objective is highly effective for capturing syntax and high-level semantics, which accounts for LLMs' strong performance in code generation and general code understanding. However, it does not explicitly align model representations with the underlying

Table 10: Performance comparison of different methods for post-synthesis area and delay prediction with knowledge distillation, where LLMs directly process textualized CDFG representations. StructRTL significantly outperforms the LLM-based baselines.

w/ KD (LLM on CDFG)	Area				Delay			
	MAE↓	MAPE↓	R^2↑	RRSE↓	MAE↓	MAPE↓	R^2↑	RRSE↓
Teacher								
PM-based Predictor	0.2982	0.05	0.9334	0.2581	0.1688	0.03	0.9484	0.2272
LLM-Based Baselines								
CodeV-DS-6.7B	0.6201	0.11	0.7194	0.5297	0.4429	0.08	0.6579	0.5849
CodeV-CL-7B	0.5817	0.10	0.7582	0.4917	0.4140	0.07	0.7173	0.5317
CodeV-QW-7B	0.5826	0.10	0.7609	0.4890	0.4196	0.07	0.7084	0.5400
Ours								
StructRTL	0.3856	0.07	0.8676	0.3639	0.2381	0.04	0.8872	0.3359

control- or data-flow structure. Structural factors central to area and delay, such as operator connectivity, critical-path depth, or reconvergent dataflow, are only implicitly encoded in token sequences and are not directly supervised during pretraining. As a result, token-based embeddings generally capture semantic content more faithfully than structural detail.

Empirically, this distinction is clearly reflected in our results: while token-based representations somewhat narrow the gap for area prediction (which correlates more directly with node-type distributions), their performance on delay prediction remains significantly weaker. Delay is highly sensitive to structural properties that token sequences struggle to represent, whereas CDFG-based representations capture these properties explicitly. This explains why StructRTL exhibits the largest performance gains on delay, even under identical knowledge-distillation settings.

A.10 Limitations of StructRTL

One potential limitation of StructRTL is that model performance may degrade as circuit designs grow increasingly large and complex. Larger designs also incur higher computational cost and memory usage, which can raise the adoption barrier. However, we note that these challenges are inherent to all learning-based quality estimation methods and are not unique to our framework.

As discussed in Section A.7 a practical mitigation strategy is to partition large designs at register boundaries, thereby decomposing the circuit into smaller and more manageable subcircuits. Area and delay can then be estimated at the subcircuit level, with the overall area obtained by summation and the overall delay determined by taking the maximum across subcircuits. This hierarchical strategy helps alleviate computational and memory requirements and may also reduce performance degradation by enabling more efficient, localized prediction.

A notable advantage of the CDFG-based representation is that register boundaries are explicitly and naturally identifiable in the graph, making such partitioning straightforward to implement. In contrast, performing partitioning directly from Verilog source code is significantly more difficult due to syntactic and semantic complexity. From this perspective, the CDFG-based formulation provides a practical benefit over LLM-based approaches that operate solely on textual RTL representations.