DynamicRTL: RTL Representation Learning for Dynamic Circuit Behavior

Ruiyang Ma[*1], **Yunhao Zhou**[*3], **Yipeng Wang**[4], **Yi Liu**[2], **Zhengyuan Shi**[2], **Ziyang Zheng**[2], **Kexin Chen**[4], **Zhiqiang He**[4], **Lingwei Yan**[4], **Gang Chen**[4], **Qiang Xu**[2], **Guojie Luo**[1, ✉]

[1]The National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University
[2]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[3]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University
[4]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
ruiyang@stu.pku.edu.cn, yunhaoz.cs@gmail.com, qxu@cse.cuhk.edu.hk, gluo@pku.edu.cn

Abstract

There is a growing body of work on using Graph Neural Networks (GNNs) to learn representations of circuits, focusing primarily on their static characteristics. However, these models fail to capture circuit runtime behavior, which is crucial for tasks like circuit verification and optimization. To address this limitation, we introduce DR-GNN (DynamicRTL-GNN), a novel approach that learns RTL circuit representations by incorporating both static structures and multi-cycle execution behaviors. DR-GNN leverages an operator-level Control Data Flow Graph (CDFG) to represent Register Transfer Level (RTL) circuits, enabling the model to capture dynamic dependencies and runtime execution. To train and evaluate DR-GNN, we build the first comprehensive dynamic circuit dataset, comprising over 6,300 Verilog designs and 63,000 simulation traces. Our results demonstrate that DR-GNN outperforms existing models in branch hit prediction and toggle rate prediction. Furthermore, its learned representations transfer effectively to related dynamic circuit tasks, achieving strong performance in power estimation and assertion prediction.

Code — https://github.com/magicyang1573/DynamicRTL

1 Introduction

As circuit complexity increases, traditional algorithms for circuit design and optimization are increasingly challenged to meet the pressing demands for time and cost efficiency. New models, representations, and methodologies for circuit understanding are urgently needed to advance hardware design and electronic design automation (EDA) research, enabling engineers to create efficient hardware solutions.

Recent works have begun to solve many canonical tasks in hardware development with deep learning methods (Chen et al. 2024; Ma et al. 2020). By training on extensive circuit data, these models have demonstrated the potential to understand the static feature of circuits and outperform traditional methods on some prediction tasks, including circuit quality (performance, power, area) estimation (Sengupta et al. 2022; Fang et al. 2023; Lopera et al. 2021), combinational functionality representation (Wang et al. 2022; Li et al. 2022; Shi et al. 2023) and so on.

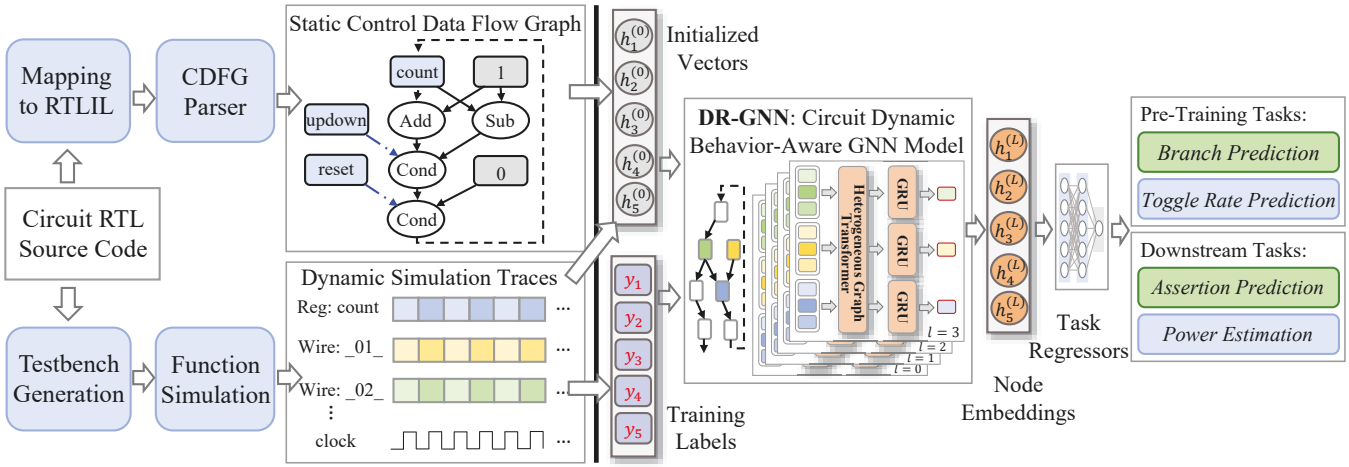
Despite these notable achievements, it has been observed that these models struggle with tasks that require in-depth analysis of circuit designs, especially those involving the dynamic behavior of circuits such as hardware verification, dynamic power estimation and so on (Khan et al. 2024; Vasudevan et al. 2021). The main reason is that current models exclusively depend on the static information of circuits (*i.e.,* hardware source code, netlist) for training. As a result, these models are limited to learning only structural or semantic information about the circuits.

However, the *dynamic behavior* of circuits is equally important for understanding circuits and facilitating dynamic-related downstream tasks. These behaviors can reveal complex dependencies and interactions that are unapparent in static representation, thereby significantly enhancing the quality of circuit representation. In this study, we diverge from traditional approaches that focus on learning static representations of circuits. In an innovative endeavor, we train the model to learn circuit representations based on their dynamic behaviors. The dynamic behavior refers to the circuit's multi-cycle execution status under specific inputs. We aim to predict various circuit behavior features under specific input sequences, such as branch and assertion hits, variable toggle rate, dynamic power and so on.

We design DR-GNN (**D**ynamic**R**TL-GNN) to learn the dynamic behavior of circuits. DR-GNN is distinguished by two aspects. First, we construct the Graph Neural Network (GNN) based on the operator-level Control Data Flow Graph (CDFG) of Register Transfer Level (RTL) circuit design. The CDFG provides a high-level and concise representation of the dynamic functionality of the circuit, making it an ideal foundation for learning dynamic behaviors. Second, we develop a circuit dynamic behavior-aware GNN propagation mechanism that integrates specific circuit features into the model. We design semi-decoupled aggregators tailored for numerous RTL operators and introduce a position-aware operator messaging mechanism in heterogeneous graph transformer (Hu et al. 2020). This enables DR-GNN to capture both the structural and dynamic characteristics of circuits.

We collect a comprehensive Verilog dataset and employ branch execution and variable toggle information as training supervision. DR-GNN achieves outstanding performance in these pre-training tasks, significantly surpassing models

[*]These authors contributed equally.



(a) Circuit Data Preparation **(b) Dynamic Tasks Prediction with DR-GNN**

Figure 1: The overview of DynamicRTL. The framework prepares circuit training data, including CDFGs and simulation traces, and employs pre-training tasks to train the model. The learned representations are then used for downstream tasks.

based solely on circuit structural and semantic learning. Moreover, DR-GNN demonstrates effective transfer learning on downstream tasks including power estimation and assertion prediction, which highlights its ability to generate useful circuit representations for tasks related to circuit dynamic behaviors. To our knowledge, DR-GNN is the first unified model that captures the dynamic behavior of hardware designs, setting the stage for teaching neural network models to better understand how circuits execute.

In summary, this paper makes the following contributions:

- We introduce the problem of learning circuit dynamic representation. We create the first dynamic circuit dataset, comprising over 6,300 diverse Verilog designs, along with their CDFGs and 63,000 simulation traces.
- We design the DR-GNN model which is built on the circuit operator-level CDFG and captures the dynamic functionality of the circuits.
- We obtain the first unified representation for circuit dynamic behaviors and achieve the state-of-the-art performance with an average accuracy of 94.80% in branch hit prediction and 94.25% in toggle rate prediction.
- We show that the pre-trained DR-GNN representations are useful for transfer learning, achieving an average accuracy of 79.60% in power estimation and 87.03% in assertion prediction.

2 Preliminary and DynamicRTL

Figure 1 presents an overview of DynamicRTL. In this section, we discuss key concepts in our workflow. We introduce our method for representing the RTL circuit source code as a control data flow graph. Then, we explain the concept of dynamic behavior in circuits. Finally, we outline the tools used to extract these information from circuit designs.

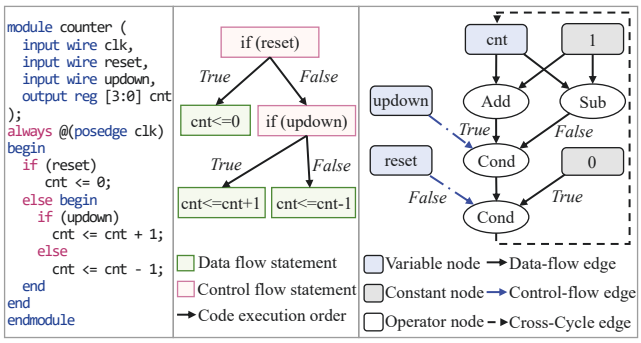
RTL Code. *Register transfer level* (RTL) is an abstraction level in digital circuit designs. A digital circuit is composed of combinational logic (computing operators, branch controls) and sequential logic (registers). The assignment in RTL represents a physical connection between wires or registers. A wire is a conductive path for signal transmission, while a register is a small storage element that holds data temporarily during the execution of a digital circuit.

Control Data Flow Graph. A common way to understand the function of RTL is *control data flow graph* (CDFG). Figure 2 shows an example of RTL code and its CDFG. Previous work, Design2Vec (Vasudevan et al. 2021), uses a statement-level CDFG to learn semantic representation of circuits, where each node represents an RTL statement. However, this CDFG only reflects the execution of circuit code from a software perspective, which fails to reflect the actual data flow within circuits.

To better represent the dynamic behavior of circuits, we propose the operator-level CDFG as the circuit graph structure. In the operator-level CDFG, directed edges indicate data or control flows between nodes. In sequential circuits, registers store values to the next clock cycle, therefore the in-edges of registers signify data flow across clock cycles.

The CDFG comprises three node types: variable nodes, constant nodes and operator nodes. *Variable nodes* represent variables that have dynamic values, such as wires and registers. Input wires are special variable nodes whose values depend on the external drive. *Constant nodes* represent unchanging values throughout clock cycles. *Operator nodes* represent operations on variables in data flow. The `condition` node is a unique operator node, which acts as a multiplexer and uses select signals to control data flow from different channels. The nodes in CDFG can be multi-bit, enabling a higher abstraction level and a more effective way to represent RTL behaviors compared with single-bit circuit graphs (Fang et al. 2023; Li et al. 2022). For more details on circuit CDFG, refer to Appendix B in supplementary material.

Circuit Dynamic Behavior. The dynamic behavior of a circuit refers to the changes in wire and register values dur-



(a) RTL Source Code (b) Statement-level CDFG (c) Operator-level CDFG

Figure 2: Comparison between statement-level CDFG and operator-level CDFG.

ing execution. Input signals drive these changes based on the combinational logic. Some values are stored in registers to be used in the next clock cycle. Given different input sequences, the circuit may exhibit significantly different behaviors, which has not been adequately addressed in prior studies on circuit representation. In our work, we consider both sequential and combinational features of the circuit. We take the values of all wires and registers into account when assessing dynamic behavior. These values can be easily obtained through hardware simulation.

Tools. To construct the CDFG, we first convert the source RTL code into RTL intermediate language (RTLIL) using EDA tool Yosys (Wolf, Glaser, and Kepler 2013). This conversion results in a functionally equivalent description, consisting only of assign statements and register transferring. Next, we use the Stagira Verilog parser (Chen, Meng, and Chen 2023) to parse the RTLIL and generate an *abstract syntax tree* (AST). Finally, we traverse the AST to create circuit CDFG. To collect the dynamic traces of circuits, we use a commercial hardware simulator and capture the value of wires and registers in each clock cycle.

3 DR-GNN: Circuit Dynamic Behavior-Aware GNN

3.1 Overview

DR-GNN takes as input the CDFG and input sequence I. The circuit has branch set $\mathbb{B}$ and variable set $\mathbb{V}$. We use branch hit prediction and variable toggle rate prediction as the pre-training tasks. The objective of DR-GNN is to predict the branch hit probability $is\_hit(I, \mathbb{B})$ that input I hit branches, and the toggle rate $toggle\_rate(I, \mathbb{V})$ that input I trigger variable transitions.

Figure 3 presents the architecture of DR-GNN. The model is constructed based on the circuit CDFG. The embedding of each node represents its dynamic behavior. DR-GNN operates in three stages: Firstly, the initial dynamic embedding for each node is encoded based on its attribute and functionality (Section 3.2). Secondly, DR-GNN updates the node embeddings with dynamic behavior-aware propagation (Section 3.3). Thirdly, the final embeddings are used for prediction tasks related to circuit dynamic behaviors (Section 3.4).

3.2 CDFG Node Initialization

In DR-GNN model, node embeddings represent the dynamic behaviors. Before the GNN propagating, we encode some known dynamic behaviors as the initial embedding of CDFG nodes. The nodes are initialized based on three categories.

Input Sequence Encoder. We encode the input sequence as the initial embedding of input nodes. Considering many circuit operators function at binary level, we design a binary input encoder, which involves two steps: embedding the value in each clock cycle, and integrating these embeddings into a sequence embedding. We treat the value in each clock cycle as a binary vector and embed it with linear projection, similar to prior work on numerical representation (Yan et al. 2020). We define a learnable vector for each bit position, and sum these vectors element-wise, modulated by the value of corresponding bit. For the learnable vector $\boldsymbol{e}_i$, for each bit and the n bit value $v = \{b_1, b_2, ..., b_n\}$, we compute $\boldsymbol{v}_{emb} = \sum_{i=1}^{n} b_i \boldsymbol{e}_i$. Then we use gated recurrent unit (GRU) to embed values into a sequence embedding and use that as the initial embedding of input nodes. We train the input sequence encoder independently before training the whole model. We use a sequence reconstruction task to simultaneously train an encoder and a decoder. After training, we freeze the encoder weight and use it as our input sequence encoder in the model.

Constant Value Encoder. The constant node can be seen as an input node that keeps the same value in each clock cycle. In order to ensure the embedding consistency of each node and reduce the complexity of GNN learning, we employ the same method as input encoder to encode the constant value.

Initialization of Operator Nodes. The embedding of operator nodes represents the dynamic behavior of their calculated results in each clock cycle. Since we cannot predict these data before simulation, we initialize these embeddings to all zeros, indicating the unknown.

3.3 Circuit Dynamic Behavior-Aware Propagation

Circuit signals are processed through hardware components, which correspond to the operator nodes in CDFG. We use the propagation in GNN to represent the dynamic behaviors. Since the CDFG is a directed heterogeneous graph, we design the propagation mechanism referring to the heterogeneous graph transformer (HGT) (Hu et al. 2020). However, there are challenges arising from the unique characteristics of circuit CDFGs that require tailored approaches for current HGT.

Semi-Decoupled Operator Aggregation. Circuit CDFGs contain diverse operators. Previous works on circuit representation focus mainly on netlist with limited operator types, which usually allocate a separate aggregator for each gate type to represent circuit functionalities (Li et al. 2022; Wang et al. 2022). These approaches are insufficient for representing RTL-level CDFG, which includes over 30 operators. In this case, using separate aggregators for each operator is impractical, as it would make the model overly complex and difficult to train. A potential solution is to use a shared aggregator combined with an attention-based messaging mechanism to differentiate operator functions. However, it is challenging

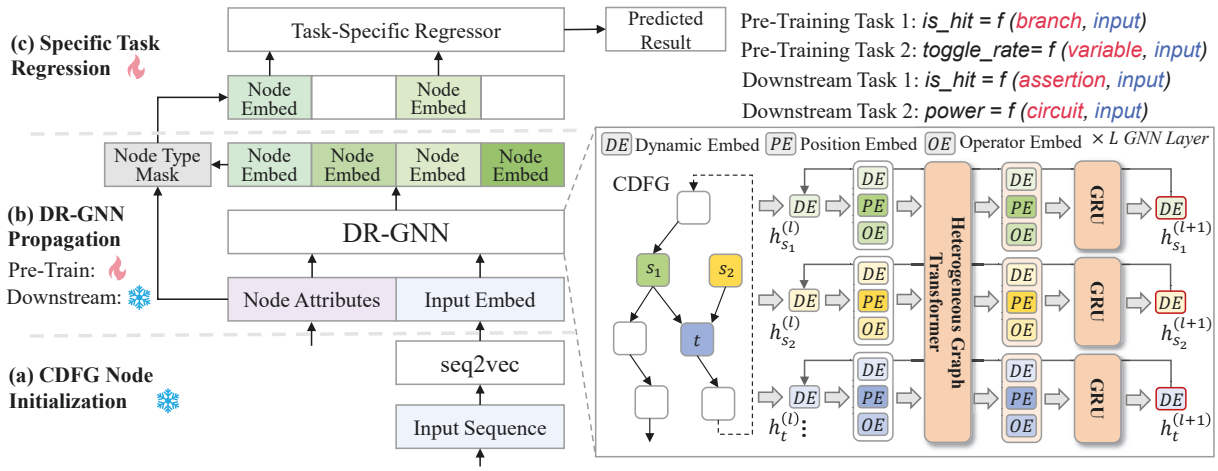


Figure 3: The overview of circuit dynamic behavior-aware DR-GNN model.

Group	Num	Description
Unary	7	Operators with one operand
Binary	20	Operators with two operands
Multary	2	Operators with more than two operands
Cond.	1	Condition operator for branch selection
Reg.	1	Register variable treated as operator

Table 1: Semi-decoupled aggregator groups for operators.

to rely solely on a shared aggregator to accurately represent such a diverse array of operators.

Therefore, we propose semi-decoupled aggregation mechanism to represent operators in circuit CDFG. Operators are categorized into five groups based on their characteristics and importance, as shown in Table 1. Operators within each group share aggregation weights and use attention-based message mechanism to represent different functions.

Position-Aware Operator Messaging. To represent different functions of operators within each aggregation group, we use attention mechanism in message passing. It is crucial to consider necessary information for the attention to effectively capture circuit dynamic behaviors. The first necessary information is the operator type, which directly affects the computing results. Additionally, some operators are non-commutative. For instance, the subtraction operator yields different results for `a-b` and `b-a`. Therefore, it is also essential to consider the relative position of operand nodes to the operator when aggregating embeddings. Traditional position encodings in graph transformer are insufficient to represent such operand-to-operator positional relationships (Rampášek et al. 2022). To address this, we incorporate operand position encoding and design a position-aware operator messaging mechanism in HGT to represent circuit dynamic behaviors.

Consider an operator node t with dynamic embedding $\boldsymbol{h}_t^{(l)}$ in the l-th GNN layer. Node t has multiple source nodes $s_1, s_2, ..., s_m$, each with dynamic embeddings $\boldsymbol{h}_{s_1}^{(l)}, \boldsymbol{h}_{s_2}^{(l)}, ..., \boldsymbol{h}_{s_m}^{(l)}$. The type of the operator node is denoted as $op[t]$. The position of each source node s relative to t is denoted as $pos[s,t]$, indicating the order of operand s in the expression of operator t. We use rotary position embedding to encode these positions. The input to the heterogeneous graph transformer is then constructed by concatenating the source node position embedding, the operator type embedding, and the dynamic embedding of the source node.

$$\boldsymbol{x}_{s,t}^{(l)} = \text{Concat}(\boldsymbol{h}_s^{(l)}, \text{Embed}(pos[s,t]), \text{Embed}(op[t])) \quad (1)$$

In the heterogeneous graph transformer, we map the embedding of source node s and target node t into Query and Key vectors. The concatenation of the Query and Key vectors serves as the input to the attention mechanism. The attention weight from node s to t is calculated by:

$$w_{s,t}^{(l)} = \text{MLP}(\text{Concat}(\text{Q-linear}(\boldsymbol{x}_{s,t}^{(l)}), \text{K-linear}(\boldsymbol{h}_t^{(l)}))) \quad (2)$$

For each source node s connected to node t, the normalized attention weights are computed using the softmax function:

$$\{a_{s_1,t}^{(l)}, ..., a_{s_m,t}^{(l)}\} = \text{Softmax}(\{w_{s_1,t}^{(l)}, ..., w_{s_m,t}^{(l)}\}) \quad (3)$$

The message from s to t is represented by a Value vector:

$$message_{s,t}^{(l)} = \text{V-linear}(\boldsymbol{x}_{s,t}^{(l)}) \quad (4)$$

Finally, the messages from all source nodes are aggregated using the attention weights to obtain the aggregated information for node t:

$$aggr_t^{(l)} = \sum_{i=1}^{m} message_{s_i,t}^{(l)} \cdot a_{s_i,t}^{(l)} \quad (5)$$

Node Embedding Update. We use GRU to update the embedding of target node t, where $aggr_t^{(l)}$ is the aggregation information as the GRU input and $\boldsymbol{h}_t^{(l)}$ is the past state of GRU. The output $\boldsymbol{h}_t^{(l+1)}$ serve as the dynamic embedding for the subsequent $(l+1)$-th GNN layer.

$$\boldsymbol{h}_t^{(l+1)} = \text{GRU}(aggr_t^{(l)}, \boldsymbol{h}_t^{(l)}) \quad (6)$$

By stacking L layers, we obtain the node representations $\boldsymbol{h}^{(L)}$ for the entire graph. These representations can be used for prediction of the pre-training and downstream tasks.

3.4 Pre-training Tasks

Task 1 predicts the branch hit probability. Hardware branches, represented by conditional statements like `if` or `case` in RTL code, capture abundant dynamic circuit behaviors and are key coverage metrics in hardware verification. We identify nodes serving as select signals for the `condition` node to form the branch set $\mathbb{B}$. For each branch node b, we read out its embedding $h_b^{(L)}$ and pass it through a multi-layer perceptron to predict the hit probability.

$$\hat{P}_b = \mathrm{MLP}_{\mathrm{branch}}(\boldsymbol{h}_b^{(L)}), \quad b \in \mathbb{B} \tag{7}$$

Task 2 predicts the variable toggle rate. In synchronous sequential circuits, a toggle occurs when the value of a variable changes, typically on the edge of the clock signal. The toggle rate of a variable refers to the number of toggles divided by the total number of clock cycles. This metric indicates how frequently a variable changes and is crucial for downstream tasks like power estimation. We select nodes representing variables to form the variable set $\mathbb{V}$, and use each node's embedding $h_v^{(L)}$ to predict the toggle rate.

$$\hat{R}_v = \mathrm{MLP}_{\mathrm{toggle}}(\boldsymbol{h}_v^{(L)}), \quad v \in \mathbb{V} \tag{8}$$

We use multiple supervisions in pre-training to learn a comprehensive circuit dynamic representation, which captures diverse behaviors and improves generalization. In Section 5.1 and 5.2, we will show that abundant supervisions enhance performance in each downstream task.

4 Experiments

4.1 Dataset Preparation

Circuit Designs Collection. We construct the first dynamic circuit dataset, which consists of around 6,300 different circuit designs and 63,000 circuit simulation traces. We collect existing open-source Verilog datasets (Zhang et al. 2024; Thakur et al. 2024). To further expand our collection, we search GitHub using keywords such as "Verilog", "RTL", and "circuit" and scrape Verilog files from relevant repositories. During the preprocessing stage, we filter out designs with syntax errors, synthesis or simulation failures. We retain only sequential circuits in the dataset.

The collected designs range in size from 10 to over 500 CDFG nodes. The scale of these circuits is comparable to prior circuit representation works (Wang et al. 2022; Shi et al. 2023; Fang et al. 2023). A detailed introduction to the dataset is provided in Appendix C in supplementary material. Additionally, we also evaluate the generalizability of DR-GNN on larger-scale circuits ranging from 1k to 10k CDFG nodes in downstream tasks, which are detailed in Section 5.1.

Simulation Traces Collection. We use commercial simulator to collect the traces. Random input patterns are generated, and each circuit is simulated with 10 traces. During simulation, all internal variable values are recorded at each clock cycle. When generating testbench, we automatically identify special signals such as the reset signal. The reset signal is active only at the beginning of simulation and remains inactive thereafter, which ensures the simulation traces capture a comprehensive range of circuit behaviors.

4.2 Experimental Settings

We split our dataset by designs, using 80% of the designs for training, 10% for validation and 10% for testing. This ensures the designs used for testing are never seen by the model during training and validation, providing a reliable measure of its generalization ability. Experiment is conducted on an A800 GPU. A detailed introduction to the training process and model hyperparameters is provided in Appendix A in supplementary material.

4.3 Pre-training Tasks

Baseline Comparison. We evaluate DR-GNN model on two pre-training tasks using input sequences of varying lengths. Since these tasks focus on branches and variables at RTL level, which are challenging to map to synthesized netlist circuits, we compare DR-GNN with two RTL-level circuit representation frameworks. HGVC (Sengupta et al. 2022) extracts CDFG features (e.g., node type, bit width) as initial node embeddings and applies GNNs for prediction tasks. We apply GCN (Kipf and Welling 2016), GAT (Veličković et al. 2017), Gated GCN (Bresson and Laurent 2017) and GATv2 (Brody, Alon, and Yahav 2021) as baselines in the experiments. For a fair comparison, we incorporate dynamic information into HGVC and use the same initialization for input node embeddings as in DR-GNN. Design2Vec (Vasudevan et al. 2021) constructs the GNN based on statement-level CDFG and learns semantic embeddings. Because its nodes represent statements rather than variables, we evaluate it only on branch hit prediction, consistent with its original study.

Table 2 presents the experiment results. All models are trained for 5 times and the mean and standard variance of test performance are reported. The DR-GNN model achieves accuracy of about 94% in both branch hit prediction and toggle rate prediction[1] across different input sequence lengths, outperforming all baselines.

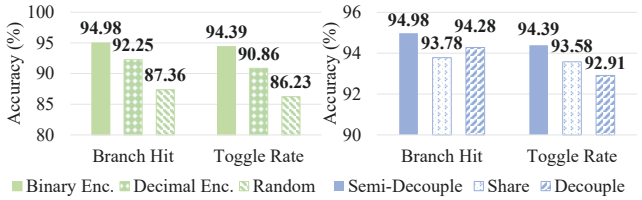
Ablation Study of Input Encoding. The dynamic behavior of circuits is determined by the input sequence, which we encode in DR-GNN using a binary input encoder. We compare DR-GNN with two ablation variants, as shown in Figure 4a. In the random encoding, we ignore the input sequence and provide random value as the model's input, excluding dynamic information from the model. In the decimal encoding, we treat the input sequence as decimal values rather than a binary vectors. Experiments demonstrate that encoding the input sequence as a binary vector sequence allows the DR-GNN model to achieve the best performance.

Ablation Study of Aggregator. To handle the diversity of operator types in CDFG while avoiding overly large models, we design a semi-decoupled aggregation mechanism. We evaluate two ablation variants, as shown in Figure 4b. In the decoupled aggregation mechanism, each operation type is assigned an independent aggregator. In the shared aggregation mechanism, a single aggregator is used for all operators, with their different functions represented through an attention mechanism. The results demonstrate that the semi-decoupled aggregation achieves the best performance.

[1]We define toggle rate accuracy as $1 - |\hat{R}_v - R_v|$, consistent with the approach in prior study (Khan et al. 2024).

Pretraining task	Branch Hit Prediction (%)				Toggle Rate Prediction (%)			
Seq. length	10	20	30	50	10	20	30	50
GCN	81.75±0.58	82.24±0.52	82.68±0.78	82.97±0.59	81.88±0.51	81.15±0.35	80.99±0.58	80.93±0.47
GAT	82.68±0.42	83.57±0.72	82.70±0.63	83.81±0.41	82.38±0.53	81.96±0.68	81.49±0.33	81.79±0.63
Gated GCN	83.52±0.64	83.96±0.40	84.15±0.57	84.34±0.68	83.07±0.30	82.84±0.54	82.46±0.35	82.62±0.37
GATv2	85.23±0.71	85.60±0.76	85.72±0.56	86.16±0.78	85.25±0.37	85.11±0.46	84.37±0.38	84.55±0.51
Design2Vec	82.03±0.88	86.78±0.65	72.81±1.13	69.33±0.91	-	-	-	-
DR-GNN	**94.69**±0.75	**94.75**±0.64	**94.98**±0.59	**94.78**±0.69	**93.74**±0.41	**93.98**±0.47	**94.39**±0.34	**94.90**±0.36

Table 2: Comparison of accuracy of different models for branch hit prediction and variable toggle rate prediction tasks.



(a) Influence of input encoders. (b) Influence of aggregators.

Figure 4: Ablation Study. Branch hit and toggle rate prediction on different input encoding and aggregation mechanisms.

Ablation Study of Messaging. We propose a position-aware operator messaging mechanism in DR-GNN. Ablation studies compare the performance with variants that eliminate the operator type (w/o Operator) and the operand positions (w/o Position) from the attention mechanism. Additionally, we compare our operand position embedding with two existing position encoding methods in graph transformer: global position encoding using Laplacian eigenvectors (Dwivedi and Bresson 2021) and relative position encoding using distance embedding (Li et al. 2020). These position encodings are incorporated into the initial node embeddings. The results in Figure 5 show that the position-aware messaging achieves the best performance.

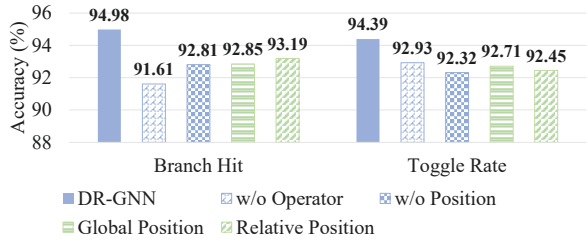
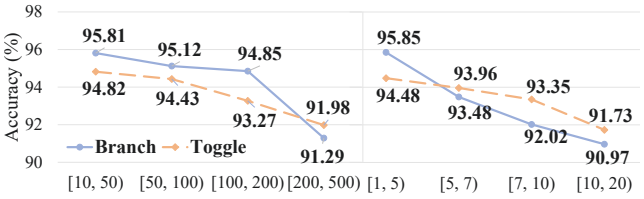


Figure 5: Ablation Study. Branch hit and toggle rate prediction accuracy on different messaging mechanisms.

Circuit Scale Variation Analysis. We evaluate the performance of DR-GNN on circuit designs of different sizes and layer depths, as presented in Figure 6. The circuit size is defined by the node count of the CDFG, and the layer depth is defined by the longest path from input nodes to output nodes in CDFG. For designs with fewer than 50 nodes and 5 layers, our model achieves over 95% branch hit prediction accuracy and more than 94% toggle rate prediction accuracy. As design size increases, the complexity of understanding rises, while our model maintains good performance.



(a) Influence of CDFG nodes (b) Influence of CDFG depth

Figure 6: Branch hit and toggle rate prediction on different circuit CDFG size and depth.

5 Downstream Tasks

This section evaluates the generalizability of the pre-trained DR-GNN model on two downstream tasks. We use the learned dynamic circuit representation as the circuit node feature and use that in the model for downstream tasks. These downstream tasks figure out whether DR-GNN effectively capture the dynamic behaviors of circuit and enhance the performance of dynamic-related circuit tasks.

5.1 Evaluation on Power Estimation

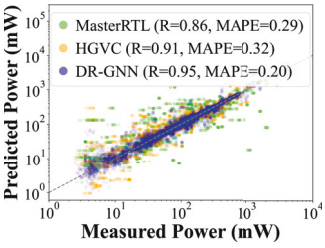
In circuit development, power estimation is time-consuming and challenging due to its strong dependence on circuit dynamic behaviors. We evaluate DR-GNN on RTL-level dynamic power estimation task. In our study, we use the learned dynamic representation, concatenated with node attributes including node type and width, as node embeddings for estimation. With DR-GNN weights frozen, we introduce an additional GAT based on CDFG to estimate power.

Experimental Settings. We synthesize the RTL code using commercial synthesis tool with the NanGate 45nm technology library (Stine et al. 2007). The dynamic power of the gate-level netlist under specific input is recorded as the ground-truth label. We compare DR-GNN with two baselines. MasterRTL (Fang et al. 2023) models circuit as bit-level simple operator graph (SOG), which is close to netlist, and extracts toggle rate as node features. It then uses a tree-based model to estimate power. HGVC (Sengupta et al. 2022) models the circuit as CDFG, similar to ours, using node attributes (*e.g.,* type, width) as initial embeddings and a GAT for power prediction. We train these models on our dataset, which is significantly larger than the dataset used in their original studies. We use correlation coefficient (R), mean absolute percentage error (MAPE), and root relative square error (RRSE) to evaluate the accuracy between predicted value $\hat{y}$ and ground-truth

y, as consistent with prior work (Fang et al. 2023).

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i},\ \text{RRSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad (9)$$

Power Estimation Results. Figure 7 presents the results. Using the pre-trained dynamic representation, our DR-GNN model achieves the best accuracy compared to MasterRTL and HGVC. This is because the dynamic embedding is trained on abundant circuit data and supervisions, which contain more information than manually extracted features. Moreover, our model outperforms MasterRTL without the need for synthesis into the SOG structure or running simulations to obtain toggle rates, which demonstrates the effectiveness of our dynamic representations. We also investigate the impact of using different pre-training supervisions on the power estimation performance. Experiment results show that using both types of supervision leads to better performance than using toggle supervision alone (DR-GNN$_{ts}$) or branch supervision alone (DR-GNN$_{bs}$).



(a) Predict vs ground-truth.

Method	R	MAPE	RRSE
MasterRTL	0.86	0.29	0.51
HGVC	0.91	0.32	0.41
DR-GNN$_{bs}$	0.92	0.27	0.42
DR-GNN$_{ts}$	0.93	0.20	0.37
DR-GNN	**0.95**	**0.20**	**0.33**

(b) Accuracy comparison.

Figure 7: Power estimation comparison of MasterRTL, HGVC and DR-GNN with different supervisions.

Power Estimation on Large Scale Circuits. Table 3 presents the accuracy and runtime improvements achieved by DR-GNN on five large-scale circuit designs, compared to a netlist-level power estimation method DeepSeq2 (Khan et al. 2025), and the conventional power estimation flow using EDA tools. For small-scale circuits, the netlist-level model exhibits higher accuracy. This is because the circuit netlist provides a more fine-grained representation of the circuit structure compared to the RTL-level CDFG. However, this comes at the cost of a significantly larger graph structure in the netlist-level And-Inverter Graph (AIG). DeepSeq2 cannot handle the three larger-scale circuits due to memory constraint (illustrated as MO in the table).

To conclude, DR-GNN has stronger applicability to large-scale circuits compared with netlist-level models. Moreover, with the dynamic circuit representation learned from large training datasets, DR-GNN significantly improves the power estimation accuracy at the RTL level compared to MasterRTL and HGVC, which enables more accurate power estimation results for circuits at the early design stage.

Design	CDFG Nodes	AIG Nodes	DeepSeq2 Error	DeepSeq2 Time (s)	DR-GNN Error	DR-GNN Time (s)	EDA Time (s)
pcie_ctrl	0.9k	22k	0.08	14.8	0.15	0.17	540
dma_sched	1.9k	43k	0.09	27.2	0.14	0.29	1232
ysyx_cpu	6.0k	120k	MO	MO	0.19	0.61	2479
csc_wl_dec	7.4k	185k	MO	MO	0.18	0.73	2175
ch_ctrl	8.3k	304k	MO	MO	0.23	0.96	2896

Table 3: Power estimation on 5 large-scale circuit designs.

5.2 Evaluation on Assertion Prediction

The assertion prediction task reveals what dynamic representation DR-GNN has learned in pre-training. With DR-GNN weights frozen, a multi-layer perceptron is trained for each assertion. The model predicts whether a variable satisfies the assertion under given input. We formulate eight assertions. For single-variable assertions, we select all the possible circuit variables as prediction target. For dual-variable assertions, we target possible variable pairs.

Assertion	DR-GNN$_{ts}$	DR-GNN$_{bs}$	**DR-GNN**	Frequency
`v < 4`	82.37	81.76	**86.98**	56.78
`v < 16`	81.74	82.57	**87.35**	61.52
`v ≠ 2`	80.88	79.39	**82.01**	70.69
`v ≠ 4`	80.59	76.38	**81.36**	73.98
`v1 ≠ v2`	88.47	84.55	**91.36**	66.51
`v1 < v2`	81.67	77.18	**88.72**	50.89
`v1&v2==0`	82.85	66.80	**86.80**	60.25
`v1|v2 ≠ 0`	89.21	85.63	**91.67**	68.83

Table 4: Assertion prediction with pre-trained representation.

Table 4 presents the assertions and their prediction accuracies. The results indicate that the learned embeddings effectively predict some simple assertions. DR-GNN accurately assesses approximate value ranges for assertions like `v<4` and `v<16`. This capability is also evident in assertions requiring variable comparisons, such as assertion `v1≠v2` and `v1<v2`. Besides, our bit-wise signal embedding approach enables the representation to support bit-level operations like bit-and (&), bit-or (|). However, we also find limitations that DR-GNN struggles to predict exact values of variables. For assertion like `v≠2` and `v≠4`, the prediction accuracy is lower, which leaves room for improvement.

Table 4 also presents prediction performance of DR-GNN$_{ts}$ (trained only with toggle supervision) and DR-GNN$_{bs}$ (trained only with branch supervision). The original DR-GNN exhibits best performance, which demonstrates the benefits of both supervisions. It also proves that supplementing more useful supervisions could help learn a more comprehensive representation of circuit dynamic behaviors.

6 Conclusion and Future Work

This paper introduces DR-GNN model which learns dynamic representations for digital circuits. The model is pre-trained on the tasks of branch hit prediction and toggle rate prediction. It then demonstrates transfer learning capabilities on downstream tasks, such as power estimation and assertion prediction, including applications to large-scale circuit designs. For the first time, we showcase the ability of deep learning to capture complex temporal logic in sequential circuits, which facilitates a more abundant understanding of circuits and promotes a more efficient circuit design process.

Acknowledgements

This work was partly supported by the National Natural Science Foundation of China (Grant No. 62090021), the Hong Kong Research Grants Council (RGC) under Grant No. 14212422, 14202824, and C6003-24Y, and in part by Huawei Technolgies Co. Ltd. under grant No. N2-2c-TH2420350.

References

Bresson, X.; and Laurent, T. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*.

Brody, S.; Alon, U.; and Yahav, E. 2021. How Attentive are Graph Attention Networks? In *International Conference on Learning Representations*.

Chen, L.; Chen, Y.; Chu, Z.; Fang, W.; Ho, T.-Y.; Huang, R.; Huang, Y.; Khan, S.; Li, M.; Li, X.; Li, Y.; Liang, Y.; Liu, J.; Liu, Y.; Lin, Y.; Luo, G.; et al. 2024. Large circuit models: opportunities and challenges. *Science China Information Sciences*, 67(10): 200402.

Chen, X.; Meng, Y.; and Chen, G. 2023. Incremental Verilog Parser. In *International Symposium of Electronics Design Automation (ISEDA)*, 236–240.

Dwivedi, V. P.; and Bresson, X. 2021. A generalization of transformer networks to graphs. *arXiv:2012.09699 [cs.LG], AAAI 2021 Workshop on Deep Learning on Graphs: Methods and Applications (DLG-AAAI)*.

Fang, W.; Lu, Y.; Liu, S.; Zhang, Q.; Xu, C.; Wills, L. W.; Zhang, H.; and Xie, Z. 2023. MasterRTL: A Pre-Synthesis PPA Estimation Framework for Any RTL Design. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1–9.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020. Heterogeneous graph transformer. In *Proceedings of the Web Conference (WWW)*, 2704–2710.

Khan, S.; Shi, Z.; Li, M.; and Xu, Q. 2024. DeepSeq: Deep Sequential Circuit Learning. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–2.

Khan, S.; Shi, Z.; Zheng, Z.; Li, M.; and Xu, Q. 2025. Deepseq2: Enhanced sequential circuit learning with disentangled representations. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 498–504.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, M.; Khan, S.; Shi, Z.; Wang, N.; Yu, H.; and Xu, Q. 2022. DeepGate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 667–672.

Li, P.; Wang, Y.; Wang, H.; and Leskovec, J. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 4465–4478.

Lopera, D. S.; Servadei, L.; Kasi, V. P.; Prebeck, S.; and Ecker, W. 2021. RTL delay prediction using neural networks. In *IEEE Nordic Circuits and Systems Conference (NorCAS)*, 1–7.

Ma, Y.; He, Z.; Li, W.; Zhang, L.; and Yu, B. 2020. Understanding graphs in EDA: From shallow to deep learning. In *Proceedings of the International Symposium on Physical Design (ISPD)*, 119–126.

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

Sengupta, P.; Tyagi, A.; Chen, Y.; and Hu, J. 2022. 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 (ICCAD)*, 1–9.

Shi, Z.; Pan, H.; Khan, S.; Li, M.; Liu, Y.; Huang, J.; Zhen, H.-L.; Yuan, M.; Chu, Z.; and Xu, Q. 2023. DeepGate2: Functionality-aware circuit representation learning. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1–9.

Stine, J. E.; Castellanos, I.; Wood, M.; Henson, J.; Love, F.; Davis, W. R.; Franzon, P. D.; Bucher, M.; Basavarajaiah, S.; Oh, J.; and Jenkal, R. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *IEEE International Conference on Microelectronic Systems Education (MSE)*, 173–174. Accessed: 2025-01-20.

Thakur, S.; Ahmad, B.; Pearce, H.; Tan, B.; Dolan-Gavitt, B.; Karri, R.; and Garg, S. 2024. VeriGen: A Large Language Model for Verilog Code Generation. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)*, 29(3).

Vasudevan, S.; Jiang, W. J.; Bieber, D.; Singh, R.; Ho, C. R.; Sutton, C.; et al. 2021. Learning semantic representations to verify hardware designs. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 23491–23504.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, Z.; Bai, C.; He, Z.; Zhang, G.; Xu, Q.; Ho, T.-Y.; Yu, B.; and Huang, Y. 2022. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 61–66.

Wolf, C.; Glaser, J.; and Kepler, J. 2013. Yosys - A free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97.

Yan, Y.; Swersky, K.; Koutra, D.; Ranganathan, P.; and Hashemi, M. 2020. Neural execution engines: Learning to execute subroutines. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 17298–17308.

Zhang, Y.; Yu, Z.; Fu, Y.; Wan, C.; and Lin, Y. C. 2024. MG-Verilog: Multi-grained Dataset Towards Enhanced LLM-assisted Verilog Generation. In *The First IEEE International Workshop on LLM-Aided Design (LAD)*.