MOSS: Multi-Modal Representation Learning on Sequential Circuits

Mingjun Wang[1,2,3,4], Bin Sun[1,3], Jianan Mu*[1,3], Feng Gu[1,2,3,4], Boyu Han[5], Tianmeng Yang[6], Xinyu Zhang[1,3], Silin Liu[1,3], Yihan Wen[7], Hui Wang[4], Jun Gao[4], Zhiteng Chao[1,3,4], Husheng Han[1,3], Zizhen Liu[1,3], Shengwen Liang[1,3], Jing Ye[1,3,4], Bei Yu[2], Xiaowei Li[1,3], Huawei Li*[1,3]

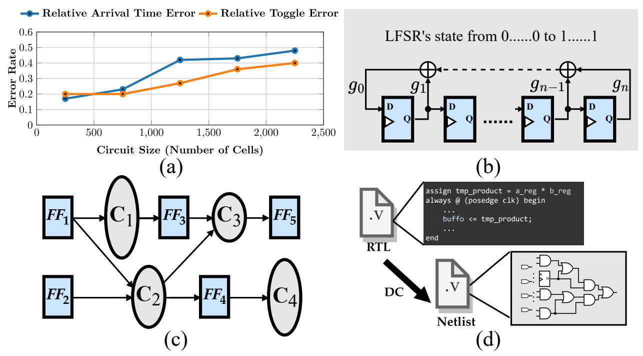
[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences [2]The Chinese University of Hong Kong. [3]University of Chinese Academy of Sciences [4]CASTEST Co., Ltd. [5]Stanford University [6]Peking University [7]Beijing University of Technology

***Abstract*—Deep learning has significantly advanced Electronic Design Automation (EDA), with circuit representation learning emerging as a key area for modeling the relationship between a circuit's structure and functionality. Existing methods primarily use either Large Language Models (LLMs) for Register Transfer Level (RTL) code analysis or Graph Neural Networks (GNNs) for netlist modeling. While LLMs excel at high-level functional understanding, they struggle with detailed netlist behavior. GNNs, however, face challenges when scaling to larger sequential circuits due to long-range information dependencies and insufficient functional supervision, leading to decreased accuracy and limited generalization. To address these challenges, we propose MOSS, a multimodal framework that integrates GNNs with LLMs for sequential circuit modeling. By enhancing D-type Flip-Flop (DFF) node features with embeddings from fine-tuned LLMs on RTL code, we focus the GNN on critical anchor points, reducing reliance on long-range dependencies. The LLM also provides global circuit embeddings, offering efficient supervision for functionality-related tasks. Additionally, MOSS introduces an adaptive aggregation method and a two-phase propagation mechanism in the GNN to better model signal propagation and sequential feedback within the circuit. Experimental results demonstrate that MOSS significantly improves the accuracy of functionality and performance predictions for sequential circuits compared to existing methods, particularly in larger circuits where previous models struggle. Specifically, MOSS achieves a 95.2% accuracy in arrival time prediction.**

I. Introduction

The application of deep learning (DL) techniques in EDA has garnered significant attention [1], [2], leading to notable advancements in areas such as routing [3], [4], synthesis [5], [6], and testing [7], [8]. Among these DL-driven EDA innovations, circuit representation learning [9], [10] has emerged as a promising research paradigm that propels predictions in subsequent EDA tasks. To further advance the development of circuit representation learning, modeling sequential circuits is particularly crucial, as they constitute the most commonly used circuit modules in practical electronic designs.

Existing works on circuit representation learning can mainly be divided into two categories, each modeling different modalities of circuit descriptions. The first category focuses on the register transfer level (RTL) code–a formal language that abstractly describes circuit behavior–utilizing large language models (LLMs) to provide a high-level understanding of functionality [11]. However, this approach faces challenges when it comes to analyzing the specific behavior and connectivity of individual cells in the netlist [12]. The second category employs graph neural networks (GNNs) for end-to-end learning, modeling the relationship between circuit functionality and structure [13]. Compared to the first approach, GNNs are closer to circuit structures, allowing for a more accurate representation of circuit details. Correspondingly, GNN-based models have achieved approximately 80% prediction accuracy for the toggle rates of sequential circuits within 1,000 gates. Such fine-grained tasks appear to be challenging for language models. Therefore, it seems to be a promising approach to use GNNs to model circuit structures for circuit representation.



```
assign tmp_product = a_reg * b_reg
always @ (posedge clk) begin
    ...
    buffo <= tmp_product;
    ...
end
```

Fig. 1 Challenges and motivations for representation learning on sequential circuits. (a) Challenge 1: Performance degradation of existing methods in complex circuit prediction tasks. Both metrics are computed by summing the error ratio to the true value for each node and then averaging over all nodes. (b) Challenge 2: Sequential circuits can have almost infinite truth tables. (c) Motivation 1: DFFs divide sequential circuits. (d) Motivation 2: Different modalities provide different levels of abstraction in structure and functionality.

Unfortunately, when using GNNs to model larger sequential circuits for predicting their functionality and performance, we encounter significant challenges. The first challenge is the difficulty in handling long-range information dependencies, which leads to decreased accuracy on larger circuits. We implemented a GNN model, adopting asynchronous updates and other techniques from [14], and conducted tests on circuits of varying sizes to predict toggle rates and arrival times. The results, shown in Fig. 1(a), indicate that existing methods exhibit a substantial increase in error rates when applied to larger sequential circuits. For example, in a circuit with 2,000 gates, the prediction error rate exceeds 40%. One primary reason for this decline is that modeling larger circuits requires learning information from more distant nodes compared to smaller circuits. Despite employing asynchronous update strategies to propagate long-range information layer by layer, these advanced GNN designs remain susceptible to the loss of distant information, leading to decreased accuracy.

The second challenge is the difficulty in forming effective training supervision for the functionality of sequential circuits, resulting in weak capability in functionality-related tasks. Sequential circuits exhibit complex functional behaviors due to their stateful nature, making it challenging to capture functionality effectively with GNNs. The latest work, DeepSeq2 [14], artificially compresses the truth tables of individual nodes to form training supervision. However, this method has several drawbacks: Inefficiency for Large-Scale Circuits: As shown in Fig. 1(b), sequential circuits can have almost infinite truth tables because of the myriad possible states and input sequences over time. This makes the approach of compressing truth tables inefficient. Inadequate Description of Overall Circuit Functionality:

Manually extracting features from truth tables on each cell for training may aid in comparing individual nodes, but it is challenging to form a complete functional description of large-scale sequential circuits. This approach limits the ability to predict the complete circuit behavior and restricts the model's generalization capability. Consequently, this constraint limits performance on downstream tasks that require understanding the circuit's global functionality.

Interestingly, the structural characteristics of sequential circuits can be exploited to reduce the long-range dependencies encountered when modeling larger circuits. As shown in Fig. 1(c), sequential circuits inherently contain certain "anchor points", specifically D-type Flip-Flops (DFFs). These anchor points aggregate upstream input information and propagate it downstream, thereby driving the circuit's functional state. Furthermore, the interconnections between DFFs determine performance metrics such as arrival times. Therefore, by progressively and accurately modeling each DFF and its immediate connectivity network, we can localize the critical information within shorter-range dependencies. This approach reduces the reliance on long-range information propagation. However, existing methodologies have not yet taken advantage of this aspect for targeted modeling.

On the other hand, compared to the vast number of cell-level nodes present in the netlist, the RTL code offers a more macroscopic description of circuit connections and the functionality they form. As illustrated in Fig. 1(d), complex connectivity structures in a netlist can be succinctly represented by a single line of RTL code, providing a clear overview of the circuit's functionality. Therefore, if we can simultaneously learn from both RTL code and circuit netlist structures, integrating local and global information collaboratively, it will significantly enhance the efficiency of circuit representation learning. Moreover, by utilizing the formal language of RTL code-which provides an effective abstraction of functionalityand an LLM fine-tuned on circuit data to interpret the RTL, we can generate more accurate functional descriptions.

In this paper, we propose MOSS, a multimodal framework that integrates GNN with LLMs. We fine-tune an LLM on RTL code to generate embeddings that capture global circuit functionality. These embeddings enhance the features of DFF cells in the netlist, enabling the GNN to focus on critical anchor points and reduce reliance on long-range dependencies. The LLM also provides a global functionality embedding, offering efficient supervision for functionality-related tasks. Additionally, we introduce an adaptive aggregation method and a two-stage propagation mechanism to better model signal propagation and sequential feedback within the circuit. By combining structural insights from GNNs with functional understanding from LLMs, MOSS improves the accuracy of functionality and performance predictions for sequential circuits, particularly in larger circuits where existing methods struggle.

The contributions of this paper are summarized as follows:

- We propose MOSS, a multimodal framework that combines LLMs with GNNs for sequential circuit modeling, enhancing the collaboration between local node embeddings and global representations for circuit functionality and structure.
- We introduce an adaptive aggregation method and a two-phase propagation mechanism in the GNN to better model signal propagation and sequential feedback within circuits.
- MOSS significantly improves the accuracy of functionality and performance predictions for sequential circuits, outperforming existing methods on larger circuits. For example, it achieves a 95.2% accuracy in arrival time prediction.

II. Background and Related Work

A. Graph Neural Networks for EDA Tasks

Graph neural networks (GNNs) are a class of deep learning models specifically designed to process graph-structured data [15]. By operating directly on graphs, they capture complex dependencies and interactions between nodes. GNNs iteratively aggregate information from neighboring nodes to update node representations [16], enabling them to capture both local and global structural features [17]. In the field of EDA, GNNs are widely used for tasks such as circuit representation learning, performance prediction, and fault detection due to their high adaptability to circuit structures [18], [19].

For instance, FGNN2 [20] and DeepGate2 [21] focused on combinational circuits, which lack the timing dependencies inherent in sequential circuits. The DeepSeq series [14], [22] sought to incorporate temporal information into GNNs by training on logic and flip probabilities to model timing behavior. DeepSeq2 advanced this approach by introducing disentangled structure, function, and timing representations. However, as in Fig. 1, DeepSeq series struggles with long-range information dependencies. This shortcoming leads to a substantial decline in accuracy when applied to larger-scale circuits. Additionally, the DeepSeq series performs learning on And-Inverter Graphs (AIGs), which, compared to real netlists composed of standard cells, have nodes with simpler and uniform functions. In EDA scenarios, crucial labels such as timing are annotated at the standard cell level rather than the AIG level. Thus, AIG-based models cannot capture these metrics directly, limiting industrial applicability. In contrast, our work models netlists composed of standard cells.

B. Large Language Models and Fine-Tuning for EDA Tasks

Large language models (LLMs) have shown promise in various EDA tasks, such as design flow automation, error detection, and hardware generation [23]. Pre-trained on vast amounts of textual data, LLMs excel at understanding RTL code and descriptions [24]. Fine-tuning techniques like low-rank adaptation (LoRA) enable efficient adaptation of LLMs to EDA-specific tasks [25]. However, while LLMs can capture the functional semantics from RTL code, their ability to model the complex structural dependencies inherent in circuits is limited, as they were initially designed for processing textual data [26]. Therefore, LLMs alone are insufficient for circuit design tasks that require both functional understanding and detailed structural representation.

C. Multimodal Alignment and Contrastive Learning

Multimodal alignment and contrastive learning have been successfully employed to integrate diverse data types [27], such as text and images [28], by aligning their representations in a shared embedding space [29]. Prominent models like CLIP [30] demonstrate the potential of contrastive learning for cross-modal tasks, including retrieval and classification. These approaches leverage different modalities' strengths to enhance the performance of representation learning [31].

III. Problem Formulation

Sequential circuits, modeled as directed graphs $G = (V, E)$, where V represents circuit components (e.g., logic gates, DFFs) and E represents component connections, pose unique challenges due to the temporal dependencies introduced by state elements. The key problem is to learn node embeddings $\mathbf{H} \in \mathbb{R}^{|V| \times d}$ that encode both structural and temporal features. These embeddings should support downstream tasks such as toggle rate prediction, arrival time prediction, and circuit classification:

$$\mathbf{H} = f(G, \mathbf{X}, \mathbf{T}) = f(V, E, \mathbf{X}, \mathbf{T}), \quad (1)$$

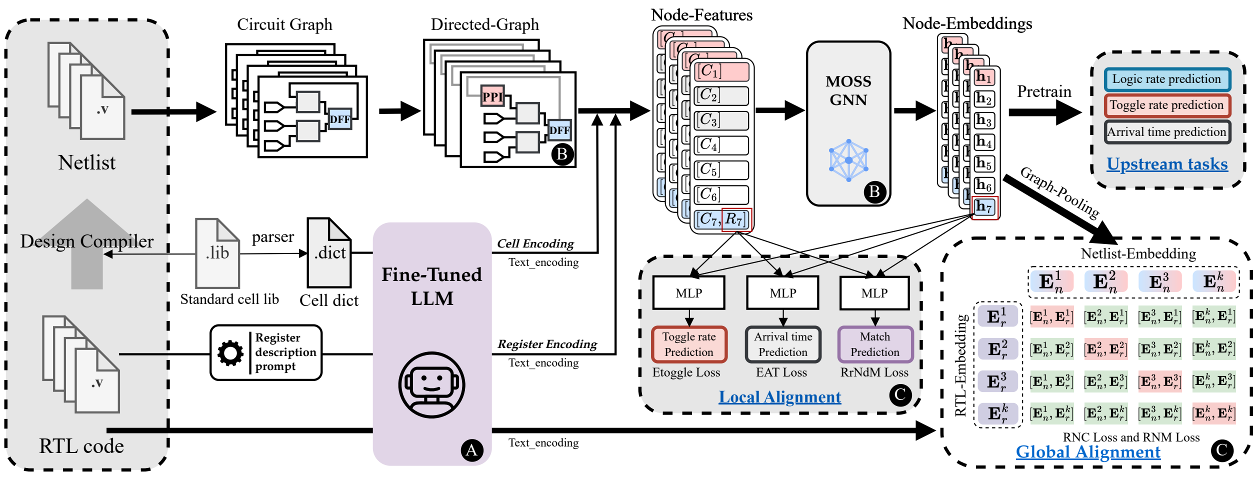

Netlist
.v
Design Compiler
RTL code
Circuit Graph
DFF
Directed-Graph
PPI
B
.lib
parser
.dict
Standard cell lib
Cell dict
Register description prompt
Fine-Tuned LLM
A
Cell Encoding
Text_encoding
Register Encoding
Text_encoding
Text_encoding
Node-Features
C_1 C_2 C_3 C_4 C_5 C_6 C_7 R_7
MOSS GNN
B
Node-Embeddings
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$ $\mathbf{h}_5$ $\mathbf{h}_6$ $\mathbf{h}_7$
Pretrain
Logic rate prediction
Toggle rate prediction
Arrival time prediction
Upstream tasks
Graph-Pooling
MLP
MLP
MLP
Toggle rate Prediction
Arrival time Prediction
Match Prediction
Etoggle Loss
EAT Loss
RrNdM Loss
Local Alignment
C
Netlist-Embedding
$\mathbf{E}_n^1$ $\mathbf{E}_n^2$ $\mathbf{E}_n^3$ $\mathbf{E}_n^k$
RTL-Embedding
$\mathbf{E}_r^1$ $\mathbf{E}_r^2$ $\mathbf{E}_r^3$ $\mathbf{E}_r^k$
$[\mathbf{E}_n^1, \mathbf{E}_r^1]$ $[\mathbf{E}_n^2, \mathbf{E}_r^1]$ $[\mathbf{E}_n^3, \mathbf{E}_r^1]$ $[\mathbf{E}_n^k, \mathbf{E}_r^1]$
$[\mathbf{E}_n^1, \mathbf{E}_r^2]$ $[\mathbf{E}_n^2, \mathbf{E}_r^2]$ $[\mathbf{E}_n^3, \mathbf{E}_r^2]$ $[\mathbf{E}_n^k, \mathbf{E}_r^2]$
$[\mathbf{E}_n^1, \mathbf{E}_r^3]$ $[\mathbf{E}_n^2, \mathbf{E}_r^3]$ $[\mathbf{E}_n^3, \mathbf{E}_r^3]$ $[\mathbf{E}_n^k, \mathbf{E}_r^3]$
$[\mathbf{E}_n^1, \mathbf{E}_r^k]$ $[\mathbf{E}_n^2, \mathbf{E}_r^k]$ $[\mathbf{E}_n^3, \mathbf{E}_r^k]$ $[\mathbf{E}_n^k, \mathbf{E}_r^k]$
RNC Loss and RNM Loss
Global Alignment
C

Fig. 2 Overview of the MOSS framework.

where $\mathbf{X} \in \mathbb{R}^{|V| \times d_x}$ represents structural features, and $\mathbf{T} \in \mathbb{R}^{|V| \times d_t}$ represents temporal features.

The main challenge of circuit representation learning is capturing long-range dependencies and temporal dynamics in sequential circuits, while most current GNNs struggle with over-smoothing and fail to retain critical temporal information. Integrating multimodal data, such as netlists and RTL descriptions, is crucial for comprehensively understanding functional and structural aspects.

IV. MOSS Framework

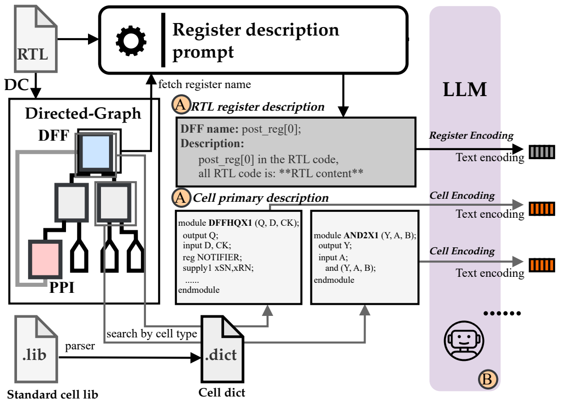
Fig. 2 describes MOSS's framework. We integrate LLM and GNN to capture both functional semantics from RTL code and structural dependencies from netlists. Our key approaches are listed below:

- LLM-enhanced node features (Fig. 2 A): The LLM is fine-tuned on RTL data to generate contextual embeddings, enriching the GNN's node features, particularly for register nodes (DFFs). Details are described in Section IV-A.
- Graph construction and modeling (Fig. 2 B): The netlist is represented as a directed graph. MOSS uses multiple aggregators and Pseudo Primary Input (PPI)/DFF nodes to model temporal dependencies in sequential circuits. Details are described in Section IV-B.
- Local and global alignment (Fig. 2 C): MOSS aligns GNN embeddings with LLM-generated RTL embeddings. Locally, tasks like toggle rate and arrival time prediction are optimized using Etoggle Loss (enhanced toggle loss), EAT Loss (enhanced arrival time loss) and RrNdM (RTL-register to Netlist-DFF matching loss). Globally, RNC Loss (RTL-Netlist contrastive loss) and RNM Loss (RTL-Netlist matching loss) align the netlist and RTL embeddings. Details are described in Section IV-C.

A. LLM Fine-Tuning and Node Feature Enhancement

To address the challenges of modeling sequential circuits, we enhance the GNN's node features by leveraging a fine-tuned LLM to generate contextual embeddings for DFF nodes and embeddings for each logic cell. Additionally, to provide more effective training supervision for circuit functionality tasks, we use the LLM to generate embeddings for the complete RTL code. To this end, we specifically fine-tuned an LLM to enhance its understanding of RTL code.

LLM Fine-Tuning. We fine-tuned Yi-Coder-9B-Chat [32], an open-source code generation model, on 31,701 RTL descriptions to extract contextual embeddings for registers and logic cells.



```
DFF name: post_reg[0];
Description:
    post_reg[0] in the RTL code,
    all RTL code is: **RTL content**
```

```
module DFFHQX1 (Q, D, CK);
 output Q;
 input D, CK;
 reg NOTIFIER;
 supply1 xSN,xRN;
  ......
endmodule
```

```
module AND2X1 (Y, A, B);
 output Y;
 input A;
   and (Y, A, B);
endmodule
```

Fig. 3 LLM feature extraction process for GNN nodes.

Data Extraction and RTL Descriptions. To enhance the features of circuit nodes, we identify each node in the circuit and extract its description as shown in Fig. 3 A . We first use Design Compiler (DC) [33] to synthesize the netlist, resulting in a directed graph structure. We then identify each elements of DFFs and logic cells. Further, for each node in the netlist, we extract its cell description, which typically includes structural and functional information. Specifically for DFFs, we locate the corresponding registers in the RTL code and generate Register Description Prompts. These prompts enable the LLM to describe the context and functionality of each DFF, capturing both local and global functional relationships.

LLM-GNN Feature Integration and Alignment. Using the extracted node information and descriptions, we generate embeddings with the LLM as illustrated in Fig. 3 B . We use mean pooling to aggregate token embeddings from the LLM, providing a comprehensive representation of the input sequence. LLM-generated node embeddings are concatenated with each node's structural features in the GNN. For DFFs, the produced embeddings combine the RTL description and the DFF cell description, capturing both structural and functional information. For logic cells, the embeddings corresponds to each cell's cell description.

B. Graph Construction and Temporal Dependency Modeling

In our GNN design, we first utilize the multimodal characteristics of MOSS to generate node embeddings. Next, we design an adaptive aggregator to better accommodate standard cell-level netlists. Finally,

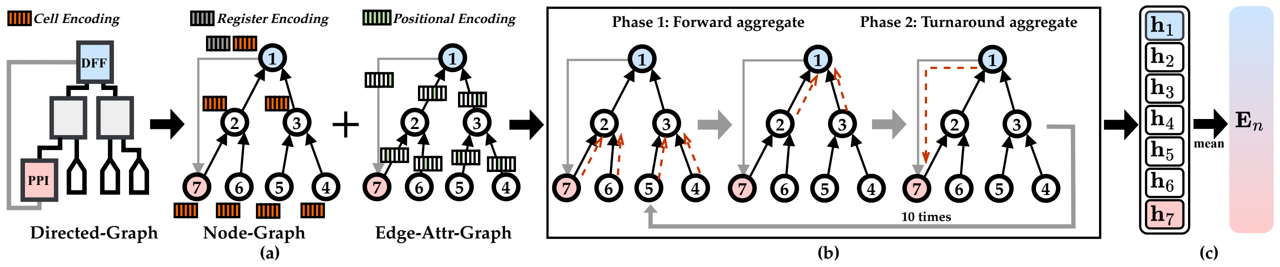


Fig. 4 (a) Graph structure in MOSS; (b) Two-phase Asynchronous temporal propagation with multi-encoding; (c) Mean-pooling strategy for graph representation.

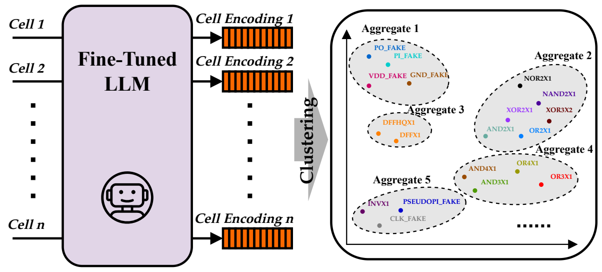


Fig. 5 Adaptive-aggregator design with adaptive clustering.

we employ a two-phase asynchronous temporal propagation mechanism to more accurately model signal propagation in circuits.

Graph Construction. In constructing the graph, we diverge from prior works that rely on manually embedding labels derived from the circuit graph's structural information. Instead, we leverage the multimodal capabilities of MOSS by utilizing the LLM to process descriptions of the RTL code and cell nodes. This approach generates embeddings for the graph nodes, as illustrated in Fig. 4(a).

First, for each cell node, we utilize the LLM to generate embeddings based on the cell's functional and structural descriptions in the standard cell library. Specifically, for the "anchor points" in sequential circuit structures, the DFFs, we further enhance their embeddings. We overlay the LLM embeddings of the contextual structure and functional descriptions of the corresponding register nodes from the RTL code onto the DFF nodes. This enhancement helps the GNN to capture a wider range of information more accurately at DFF nodes.

Finally, since different edges of cells have varying propertiesfor example, in certain logic gates, different inputs have different influences on the outputwe perform positional encoding on the edges.

Adaptive-Aggregator Design. Because we operate at the standard cell level rather than at the AIG level used in existing designs, our netlists contain cells with diverse functions and structures, and different cell libraries may include varying cell types. Therefore, we propose a clustering-based adaptive aggregator for standard cell-level netlists to adapt to different kinds of cells automatically.

As shown in Fig. 5, MOSS first uses Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and hierarchical clustering to dynamically group nodes based on their LLM-derived embeddings. DBSCAN clusters nodes based on functional similarity, detecting clusters of varying density without specifying the number of clusters in advance. Hierarchical clustering further refines these clusters by considering both functional similarities and structural dependencies such as fan-in and fan-out.

Then, each group is processed by a dedicated aggregator that learns to aggregate information based on the characteristics of the nodes. Notably, we employ graph attention mechanisms for aggregation, ultimately forming different attention-based aggregators for each category. This adaptive aggregator approach enables MOSS to flexibly adapt to different node types and interactions, capturing unique behaviors essential for modeling complex circuits.

Two-Phase Asynchronous Temporal Propagation Mechanism. To accurately model temporal dependencies in sequential circuits, MOSS adopts a two-phase asynchronous temporal propagation mechanism as in [14]. The propagation mechanism, illustrated in Fig. 4(b), includes forward and backward phases to handle the feedback behavior of DFFs. It is worth noting that both phases use asynchronous updates to more closely resemble signal propagation in circuits.

In the first phase, Forward Propagation, signals are propagated from primary inputs (PIs) to the inputs of DFFs through combinational logic gates. This phase captures the forward signal flow in the circuit, simulating how input signals affect downstream components.

In the second phase, referred to as Turnaround Propagation, the outputs from the DFFs are fed back into the circuit, simulating the characteristic feedback loops of sequential elements. This feedback is crucial for modeling the timing dependencies and state transitions intrinsic to sequential circuits, allowing the GNN to capture how past states influence future behavior. To capture both short-range and long-range dependencies, the two-phase propagation process is repeated multiple times (e.g., 10 iterations).

Afterward, as shown in Fig. 4(c), the final node representations are aggregated using mean-pooling, generating comprehensive embeddings that effectively summarize the temporal and structural information in the circuit.

C. Local and Global Alignment Strategy

Our local and global alignment strategy ensures that the GNN-generated netlist embeddings remain consistent with the LLM-derived RTL embeddings, capturing both local and global circuit characteristics. This section details the loss functions used to achieve alignment, focusing on key tasks such as toggle rate prediction, arrival time prediction, and register-DFF matching.

Local alignment aims to align nodes through task-specific losses, optimizing the GNN's ability to represent local circuit behavior. The local alignment contains several prediction tasks, and their loss functions are Etoggle loss, EAT loss, and RrNdM loss. Etoggle Loss minimizes the difference between predicted and actual toggle rates, capturing logic gates' dynamic behavior. EAT Loss focuses on predicting signal arrival times, ensuring the GNN accurately models timing dependencies. RrNdM Loss ensures proper alignment between RTL registers and netlist DFFs, minimizing mismatches in register-to-DFF mapping. We implement these loss functions as smooth L1 loss functions.

Global alignment ensures consistency between the overall structure and functionality of netlist and RTL embeddings through graph

```
# rtl_encoder - Text Transformer
# netlist_encoder - GNN
# R[n, l] - minibatch of aligned RTL codes
# N[n, c, m] - minibatch of aligned netlists
# W_n[d_n, d_r] - projection of embedding
# MLP[2*d_r, 1] - MLP for binary classification
# t - temperature parameter
# extract embedding of each modality
R_f = rtl_encoder(R)      #[n, d_r]
N_f = netlist_encoder(N) #[n, d_n]
# joint multimodal embedding [n, d_r]
R_e = l2_normalize(R_f, axis=1)
N_e = l2_normalize(np.dot(N_f, W_n), axis=1)
# RTL-Netlist Contrastive (RNC) Loss
RNCLogits = np.dot(R_e, N_e.T) * np.exp(t)
RNCLabels = np.arange(n)
loss_r = cross_entropy_loss(logits, labels, axis=0)
loss_n = cross_entropy_loss(logits, labels, axis=1)
RNCLoss = (loss_r + loss_n) / 2
# RTL-Netlist Matching (RNM) Loss
RNMLogits = [MLP(np.concatenate(R_e[i], N_e[j]))
             for i in range(n) for j in range(n)]
RNMLabels = np.eye(n)
RNMLoss = SmoothL1Loss(RNMLogits, RNMLabels)
```

Fig. 6 Numpy-like pseudocode for loss function of global alignment.

pooling and alignment losses. While RNC loss aligns the global structure, RNM loss enforces functional consistency between netlist and RTL. RNC and RNM calculation method is shown in the pseudocode in Fig. 6.

MOSS employs a multi-task learning strategy to generalize across multiple circuit-related tasks. By optimizing a weighted combination of task-specific loss functions, MOSS captures local node-level behaviors and global circuit-wide patterns. The overall loss is formulated as:

$$L_{\text{total}} = \sum_{i=1}^{n} \lambda_i L_{\text{task}i}, \tag{2}$$

where λ_i dynamically adjusts to balance the contribution of each task, ensuring that no single task dominates the learning process. This approach enhances the model's ability to generalize effectively across diverse sequential circuit tasks.

MOSS employs a dual alignment strategy that integrates local task-specific losses (Etoggle, EAT, and RrNdM) with global consistency losses (RNC and RNM). This approach captures both fine-grained circuit dynamics and global structural behavior, enhancing the model's accuracy and generalizability, particularly for complex sequential circuits.

V. Experimental Results

A. Experimental Settings

Datasets. We collected 31,701 RTL designs and synthesized them using Synopsys Design Compiler (DC), applying multiple rounds of optimization. For each RTL, we generated several distinct circuits, with sizes ranging from 100 to 5000 cells, all based on real-world designs. From this collection, we further selected a subset of RTLs with diverse functionalities and significant variations as our experimental dataset.

Model Variants. We evaluate several MOSS variants to understand the impact of critical components: "MOSS" is the full model; "MOSS w/o A" is the full model withouth local-global alignment strategy; "MOSS w/o AA" includes LLM features but omits adaptive-aggregator and alignment; "MOSS w/o FAA" is the model without all the LLM feature enhancement, the adaptive-aggregator design, and the alignment strategy. Besides, we implement DeepSeq2 [14], the state-of-the-art GNN-based model, for the comparison.

Evaluation Metrics. We assess all models using four metrics: the accuracy of arrival time prediction (ATP) for each DFF, the accuracy of toggle rate prediction (TRP) for each cell, the accuracy of power prediction (PP) for the circuit, and the accuracy of functional equivalence prediction (FEP). Note here the prediction accuracy is measured by

$$\text{accuracy} = 1 - \text{mean relative error}. \tag{3}$$

FEP measures the rate of correctly identifying functionally equivalent RTL-netlist pairs. Higher values indicate better performance. Ground truth data is collected using commercial EDA tools. Arrival Time (AT) is obtained via timing analysis on DFF nodes using PrimePower [34] and Synopsys DC [33]. Toggle Rate is derived from VCS [35] simulations over 60,000 cycles with random inputs, and power is reported by PrimePower based on their toggle rates. These ground truths ensure accurate benchmarking of MOSS's predictions.

Platform Configuration. Experiments were performed on an 8-GPU system with NVIDIA A100 GPUs. We implemented MOSS in PyTorch and trained using the Adam optimizer, with a learning rate of 6×10^{-4} and a batch size of 32. Each experiment ran for 45 epochs, with early stopping based on validation performance to prevent overfitting.

B. Experimental Results and Analysis

In this section, we evaluate the performance of MOSS across various tasks, comparing it with its variants and the baseline model DeepSeq2. The experiments focus on critical tasks such as arrival time prediction, toggle rate prediction, power prediction, and functional equivalence checking. We list results in TABLE I and TABLE II, and analyze the training loss curves shown in Fig. 7 and Fig. 8.

Arrival Time Prediction (ATP). For arrival time prediction (ATP), MOSS achieves an average accuracy of 95.2%, significantly outperforming DeepSeq2 (79.1%). In larger circuits, such as *mult_16x32_to_48*, MOSS maintains high accuracy (94.3%), while DeepSeq2 drops to 57.6%. This highlights MOSS's ability to handle long-range dependencies, a known challenge in larger circuits. The alignment strategy has a limited impact on this task, as demonstrated by the performance of MOSS w/o A (94.9%), which remains high but is slightly lower than the full model.

Toggle Rate Prediction (TRP). For toggle rate prediction (TRP), MOSS achieves 87.5% accuracy, outperforming DeepSeq2 (76.4%). In circuits such as *pipeline_reg*, MOSS reaches 92.4%, whereas MOSS w/o FAA drops to 63.6%, underscoring the importance of the adaptive aggregation and LLM enhancements, which are key for capturing dynamic behaviors in sequential circuits.

Power Prediction (PP). For power prediction (PP), MOSS leads with 96.3% accuracy, compared to DeepSeq2's 88.4%. The LLM-enhanced node features allow MOSS to model power consumption patterns more effectively, particularly in circuits like *wb_data_mux*, where MOSS achieves 96.2%, significantly higher than MOSS w/o FAA (82.6%). The smaller performance gap between MOSS and MOSS w/o A (95.1%) indicates that alignment is less critical for power estimation compared to feature enhancement.

Functional Equivalence Prediction (FEP). We selected several datasets with 512 pairs of RTL and netlist in each set and calculated the average correctness of their predictions. In functional equivalence checking, MOSS excels with an average accuracy of 93.7%, far

TABLE I Performance Comparison of MOSS Framework Variants

Circuit	#Cells	DeepSeq2 [14]			MOSS w/o FAA			MOSS w/o AA			MOSS w/o A			MOSS		
		ATP	TRP	PP	ATP	TRP	PP	ATP	TRP	PP	ATP	TRP	PP	ATP	TRP	PP
max_selector	278	81.4	78.7	94.6	47.0	75.8	88.6	82.3	85.2	94.5	95.4	89.4	99.9	**95.6**	**90.5**	**99.9**
pipeline_reg	610	77.6	83.6	91.4	52.2	63.6	63.4	80.5	88.3	90.2	94.2	92.1	94.1	**94.5**	**92.4**	**94.6**
prbs_generator	643	87.8	76.5	71.7	57.2	72.7	81.7	78.6	82.5	90.8	92.0	**87.4**	94.5	**93.0**	85.4	**95.1**
shift_reg_24	731	86.9	80.9	90.4	58.2	63.9	75.4	82.4	85.6	92.5	**96.2**	**90.2**	**97.6**	95.8	89.0	97.5
error_logger	812	79.5	83.2	94.3	58.5	59.0	81.3	81.2	80.2	95.3	94.5	85.4	99.5	**95.0**	**86.3**	**99.7**
signed_mac	1306	66.4	77.3	**95.6**	26.9	56.1	73.6	76.5	78.4	88.6	93.8	83.8	92.3	**94.5**	**85.3**	94.1
wb_data_mux	1364	95.7	64.3	88.6	45.3	25.5	82.6	85.4	75.6	88.5	98.8	82.9	91.2	**99.1**	**83.3**	**96.2**
mult_16x32_to_48	4144	57.6	66.6	80.1	19.3	40.1	54.1	75.2	72.3	85.4	93.9	84.8	91.5	**94.3**	**87.9**	**93.5**
Average	-	79.1	76.4	88.4	45.6	57.1	75.1	80.3	81.0	90.7	94.9	87.0	95.1	**95.2**	**87.5**	**96.3**

ATP: Arrival Time Prediction accuracy (%). **TRP**: Toggle Rate Prediction accuracy (%). **PP**: Power Prediction accuracy (%).
MOSS w/o FAA: MOSS without Feature Enhancement, Adaptive-Aggregator and Alignment.
MOSS w/o AA: MOSS without Aligment and Adaptive-Aggregator. **MOSS w/o A**: MOSS without Alignment.

TABLE II RTL-netlist functional equivalence prediction accuracy (FEP) on different circuit sources

Circuit	MOSS w/o FAA	MOSS w/o AA	MOSS w/o A	MOSS
github_0	4.8	19.4	24.1	**91.4**
github_1	5.3	20.3	33.6	**95.0**
github_2	10.0	23.7	32.0	**94.3**
huggingface_0	7.9	16.4	19.5	**94.1**
huggingface_1	8.7	18.3	22.9	**93.6**
huggingface_2	14.1	21.1	27.5	**93.5**
Average	8.5	19.9	26.6	**93.7**

Bold numbers indicate the best performance for each circuit
All circuits are from public repositories (GitHub and HuggingFace)

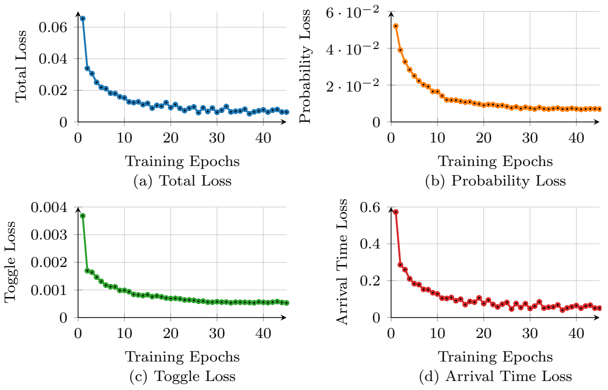


Fig. 7 Losses in the pre-training section.

exceeding MOSS w/o FAA (8.5%) , MOSS w/o AA (19.9%) and MOSS w/o A (26.6%). This significant drop in variants highlights the necessity of both feature enhancement and alignment to maintain functional consistency between RTL and netlist representations. For example, in the *github_0* circuit sets, MOSS achieves 91.4%, while MOSS w/o A only reaches 24.1%, confirming the critical role of multimodal alignment. This experiment shows that by leveraging multimodal learning, MOSS achieves superior circuit functionality modeling, enabling functional equivalence checking.

Loss Analysis. As illustrated in Fig. 7, in the pre-training phase, all loss components steadily decrease, with the total loss (see Fig. 7(a)) showing continuous improvement. Probability loss (see Fig. 7(b)), toggle loss (see Fig. 7(c)) and arrival time loss (see Fig. 7(d)) decrease steadily, indicating that MOSS learns to capture both timing and dynamic behaviors effectively.

Fig. 8 shows the global loss in multimodal alignment. The total

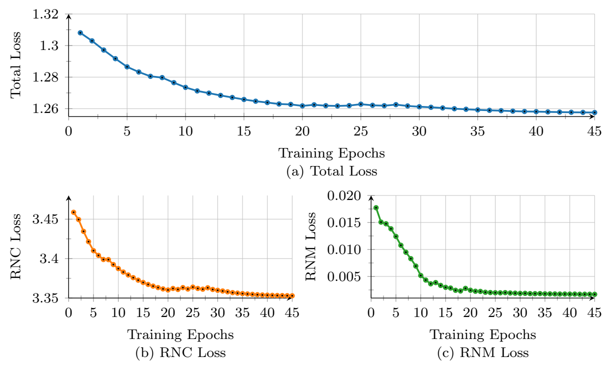


Fig. 8 Global Losses in the multimodal alignment section.

loss stabilizes around 1.26 after 45 epochs, indicating effective convergence. The RNC loss (see Fig. 8(b)) and RNM loss (see Fig. 8(c)) also converge smoothly, confirming the success of the alignment strategy. The RNM loss reaches as low as 0.002, indicating that the RTL-netlist matching is highly accurate.

The experimental results demonstrate that MOSS effectively addresses the challenges of circuit representation learning, particularly in handling larger circuits with long-range dependencies. By integrating LLM-enhanced features and GNN-based structural learning, MOSS significantly improves the accuracy of arrival time, toggle rate, and power predictions, while ensuring functional consistency through multimodal alignment.

VI. Conclusion

We propose MOSS, a multimodal framework combining GNNs and LLMs to enhance sequential circuit representation. MOSS addresses long-range information loss by leveraging LLMs for global semantic understanding and GNNs for local structural features. It also introduces adaptive aggregation and two-stage propagation to improve signal modeling. Experimental results show that MOSS significantly boosts accuracy for tasks like toggle rate and arrival time prediction, achieving 95.2% accuracy in arrival time prediction for large circuits, outperforming existing methods. We hope that MOSS can not just offer an effective solution for circuit functionality and performance prediction, but also stimulate more future research.

Acknowledgement

This paper is supported in part by the Chinese Academy of Sciences under grant No. XDB0660102, and in part by the National Natural Science Foundation of China (NSFC) under grant No. (62090024). The corresponding authors of this paper are Jianan Mu and Huawei Li.

References

[1] L. Dai, B. Wang, X. Cheng, Q. Wang, and X. Ni, "The application of deep learning technology in integrated circuit design," *Energy Informatics*, vol. 7, no. 1, pp. 1–20, 2024.

[2] C. Tian, L. Chen, Y. WANG, S. Wang, J. Zhou, Y. Pang, and Z. Du, "A survey on fpga electronic design automation technology based on machine learning," *Journal of Electronics & Information Technology*, vol. 45, no. 1, pp. 1–13, 2023.

[3] J. Yan, X. Lyu, R. Cheng, and Y. Lin, "Towards machine learning for placement and routing in chip design: a methodological overview," *arXiv preprint arXiv:2202.13564*, 2022.

[4] A. Al-Hyari, H. Szentimrey, A. Shamli, T. Martin, G. Grewal, and S. Areibi, "A deep learning framework to predict routability for FPGA circuit placement," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 14, no. 3, pp. 1–28, 2021.

[5] R. Roy and S. Godil, "Machine Learning for Logic Synthesis," in *Machine Learning Applications in Electronic Design Automation*. Springer, 2022, pp. 183–204.

[6] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. Süsstrunk, and G. De Micheli, "Deep learning for logic optimization algorithms," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–4.

[7] A. Kumar, S. L. Tripathi, and K. S. Rao, *Machine Learning Techniques for VLSI Chip Design*. John Wiley & Sons, 2023.

[8] A. Choubey and S. B. Choubey, "Machine Learning for Testing of VLSI Circuit," in *VLSI and Hardware Implementations Using Modern Machine Learning Methods*. CRC Press, 2021, pp. 23–40.

[9] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 61–66.

[10] P. Shrestha, S. Phatharodom, and I. Savidis, "Graph representation learning for gate arrival time prediction," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2022, pp. 127–133.

[11] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[12] C.-T. Ho and H. Ren, "Large Language Model (LLM) for Standard Cell Layout Design Optimization," *arXiv preprint arXiv:2406.06549*, 2024.

[13] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "CktGNN: Circuit graph neural network for electronic design automation," *arXiv preprint arXiv:2308.16406*, 2023.

[14] S. Khan, Z. Shi, Z. Zheng, M. Li, and Q. Xu, "DeepSeq2: Enhanced Sequential Circuit Learning with Disentangled Representations," *arXiv preprint arXiv:2411.00530*, 2024.

[15] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[16] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 793–803.

[17] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[18] E. Chien, M. Li, A. Aportela, K. Ding, S. Jia, S. Maji, Z. Zhao, J. Duarte, V. Fung, C. Hao *et al.*, "Opportunities and challenges of graph neural networks in electrical engineering," *Nature Reviews Electrical Engineering*, vol. 1, no. 8, pp. 529–546, 2024.

[19] A. Said, M. Shabbir, B. Broll, W. Abbas, P. Völgyesi, and X. Koutsoukos, "Circuit design completion using graph neural networks," *Neural Computing and Applications*, vol. 35, no. 16, pp. 12 145–12 157, 2023.

[20] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, Y. Huang, and B. Yu, "FGNN2: A Powerful Pre-Training Framework for Learning the Logic Functionality of Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

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

[22] S. Khan, Z. Shi, M. Li, and Q. Xu, "DeepSeq: Deep Sequential Circuit Learning," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2024, pp. 1–2.

[23] Z. He and B. Yu, "Large Language Models for EDA: Future or Mirage?" in *Proceedings of the 2024 International Symposium on Physical Design*, 2024, pp. 65–66.

[24] A. Korvala, "Analysis of LLM-models in optimizing and designing VHDL code," 2023.

[25] U. Sharma, B.-Y. Wu, S. R. D. Kankipati, V. A. Chhabria, and A. Rovinski, "OpenROAD-Assistant: An Open-Source Large Language Model for Physical Design Tasks," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2024, pp. 1–7.

[26] M. Abdollahi, S. F. Yeganli, M. A. Baharloo, and A. Baniasadi, "Hardware Design and Verification with Large Language Models: A Literature Survey, Challenges, and Open Issues," 2024.

[27] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 2, pp. 423–443, 2018.

[28] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International conference on machine learning*. PMLR, 2022, pp. 12 888–12 900.

[29] A. Rahate, R. Walambe, S. Ramanna, and K. Kotecha, "Multimodal co-learning: Challenges, applications with datasets, recent advances and future directions," *Information Fusion*, vol. 81, pp. 203–239, 2022.

[30] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning (ICML)*, 2021, pp. 8748–8763.

[31] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *Ieee Access*, vol. 8, pp. 193 907–193 934, 2020.

[32] A. Young, B. Chen, C. Li, C. Huang, G. Zhang, G. Zhang, H. Li, J. Zhu, J. Chen, J. Chang *et al.*, "Yi: Open foundation models by 01. ai," *arXiv preprint arXiv:2403.04652*, 2024.

[33] Synopsys, Inc., *Design Compiler® User Guide*, Synopsys, Inc., Mountain View, CA, USA, Apr. 2023.

[34] Synopsys , Inc., *PrimePower User Guide*, Synopsys, Inc., Mountain View, CA, USA, Mar. 2022.

[35] Synopsys, Inc., *VCS User Guide*, Synopsys, Inc., Mountain View, CA, USA, Sep. 2019.