EDA-ML: Graph Representation Learning Framework for Digital IC Design Automation

Pratik Shrestha and Ioannis Savidis
Department of Electrical and Computer Engineering, Drexel University
Philadelphia, Pennslyvania 19104
ps937@drexel.edu, isavidis@coe.drexel.edu

2024 25th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-0927-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISQED60706.2024.10528675

***Abstract*—The increase in design complexity of very large-scale integrated (VLSI) circuits due to CMOS technology scaling has generated a growing interest in the integration of machine learning (ML) algorithms with traditional electronic design automation (EDA) methodologies. Graph representation learning (GRL) techniques have gained significant attention in recent years owing to the ability to capture complex relationships in graph-structured data. This paper introduces a task-agnostic graph representation learning framework for integrated circuit (IC) design automation, which effectively addresses the challenges presented by diverse data representations and metrics generated during the standard design flow. The framework converts circuit designs and performance metrics extracted from the EDA tools at different design stages into standardized graph representations, capturing structural relationships among design elements and encoding essential characteristics through graph convolutions for meaningful vectorized embeddings. The vectorized embeddings are utilized within a machine-learning flow to predict downstream metrics of an EDA design tool. The effectiveness of the framework is demonstrated by implementing and analyzing two distinct prediction tasks; specifically post-floorplan to post-routing arrival time prediction and post-placement to post-routing interconnect parasitic impedance prediction. The evaluation is performed on a dataset generated from the IWLS'05 benchmark circuits. The analysis of results indicates an improvement in the mean absolute error of 28.71% and an improvement in the mean absolute percentage error of 39.2% for arrival time prediction. For prediction of the parasitic impedance, results indicate an improvement in the mean absolute error of 22.88% and an improvement in the mean absolute percentage error of 19.64%.**

***Index Terms*—IC design, machine learning, graph convolutional networks**

I. Introduction

Electronic design automation (EDA) plays a crucial role in the design and development of modern integrated circuits (ICs), enabling engineers to produce complex and high-performing electronic systems. With the continuous scaling of CMOS technology, the design complexity of ICs has increased exponentially, posing significant challenges for traditional EDA methodologies. To address such challenges, researchers have turned to machine learning (ML) techniques, leveraging the ability to extract meaningful patterns and make informed predictions from large-scale datasets [1]. However, the success of ML algorithms in EDA heavily relies on the quality of the data representations, as different representations either potentially reveal or obscure the underlying factors that produce variation within the data. In recent years, representation learning [2] utilizing graph neural networks (GNN) [3] has emerged as a powerful framework for the application of ML algorithms, with the objective of learning effective representations directly from the data. Due to the ability to accurately capture intricate relationships and interactions between entities, graphs serve as effective representations of circuits that closely depict the hierarchical structure and interconnections among active and passive electronic components.

A considerable amount of time in modern electronic design automation (EDA) is dedicated to the iterative execution of the design flow, with the objective of meeting circuit specifications based on different performance metrics. By accurately estimating performance metrics at various stages of the physical design flow, proactive adjustments to the circuit are possible, which results in significant reductions in design time and effort. The challenges of downstream prediction are more efficiently addressed through a generalized framework for tasks that share a common problem formulation. The same sets of relevant features are, therefore, applicable across multiple design problems.

The paper introduces EDA-ML, a graph representation learning framework that aims to improve the efficiency and effectiveness of current IC design algorithms. The framework employs a design flow that predicts downstream metrics by representing the circuit in an early design phase X and predicting metric M in a subsequent design phase Y. A task-agnostic graph structure and a feature set based on benchmark circuits are proposed as components of the framework. The contributions of the paper include

- A standardized set of graph structures and feature sets representing a digital circuit and corresponding circuit subcomponents,
- A generalized ML flow to predict downstream performance metrics given the graph representation of the circuit in the current design phase, and
- Two case studies: Specifically, post-floorplan to post-routing arrival time prediction and post-placement to post-routing interconnect parasitic impedance prediction are evaluated to demonstrate the effectiveness of the framework in solving two separate prediction problems.

The paper is organized as follows. Background on circuit design automation and the data generated at different stages of the physical design flow is provided in Section II. An overview of graph machine learning is also provided. The graph representation of a circuit and the machine learning flow are described in Section III. The application of the framework to solve two case studies is discussed in Section IV. Some concluding remarks are provided in Section V.

II. Background

Background on digital IC design flow is provided in Section II-A. The use of graph neural network layers for graph representation learning and graph model explanation is described in Section II-B.

A. Digital IC Design Flow

Electronic design automation (EDA) is a comprehensive, multi-stage process that facilitates the efficient and accurate design of an integrated circuit. The overall EDA design flow, as illustrated in Fig. 1, is divided into five critical stages: logical synthesis, floorplanning, placement, clock network synthesis, and routing. The stages collectively enable the design and implementation of robust and high-performing digital circuits, while optimizing for power, performance, and area. Each stage of the IC design flow takes as input budgets and requirements defined as design constraints, as well as the output generated by the previous design stage. For example, during physical design, the placed circuit serves as a starting point for clock tree synthesis (CTS). Constraints are specific to the given design stage, further impacting the design process. Throughout the EDA flow, various data formats are utilized to capture and represent different aspects of the design of a circuit. The files generated at the end of each design stage are generally of standardized format. The completion of each stage results in the generation of the following files:

1) **Logical Netlist:** Technology-specific verilog or VHDL files that use standard cells defined in the Process Design Kit (PDK) of a given technology node and are logical representations of the circuit.
2) **Physical Netlist:** Library Exchange Format (LEF) and Design Exchange Format (DEF) files that are technology-specific netlist representations with spatial information of input/output pins, standard cell placements, and interconnect routing.
3) **Timing reports:** Reports that are generated using a Static Timing Analysis (STA) tool and include information on timing paths, arrival time, and required time. Multiple timing paths are extracted from a single circuit.
4) **Interconnect Parasitics:** Standard Parasitic Extraction Format (SPEF) files [4] containing parasitic impedance information, specifically the resistance, inductance, and capacitance of a circuit. Interconnects are segmented into sections with reported RLC values, enabling accurate analysis and estimates of signal timing and power consumption. While estimated parasitics are obtained post-placement, the true parasitic impedance is determined after the completion of the routing process.

B. Graph Representation Learning

Graph convolutional network (GCN) layers [5] transform graphs into low dimensional vector embeddings by leveraging message passing and graph convolution operations to capture and combine information from neighboring nodes. The embeddings encode structural and relational information, facilitating tasks that include node classification, regression,

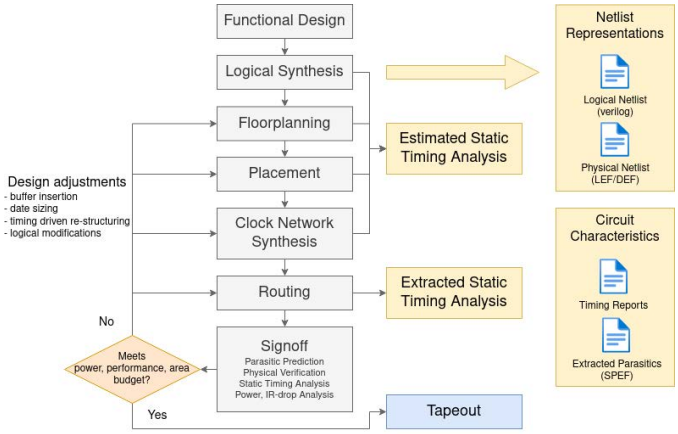

Fig. 1: Physical design automation flow.

and graph clustering. A graph convolution layer is defined by an aggregation function and a multi-layer perceptron and operates on a directed graph $G = (V, E)$, where $v \in V$ represents a node and $e \in E$ represents an edge in the graph. The aggregation function is defined as

$$\bar{h}_i = \sum_{(v_i, v_j) \in E_i} u_{ij} h_j, \tag{1}$$

where E_i is the set of edges connecting node v_i to all neighboring nodes represented by a set V_j such that node $v_j \in V_j$, h_j denotes node features applied as input to a convolutional layer, and u_{ij} is a trainable weight for aggregation of node v_i for features of node v_j. The representation of (1) in matrix form is given by $\bar{H} = UH^T$, where H denotes the matrix of node features such that $\bar{H} = [\bar{h}_1, ..., \bar{h}_n]$, and U denotes a trainable weight matrix.

Spatial graphs represent data with nodes and edges that include spatial properties, which enable analysis and modeling of relationships within a physical or spatial context. Spatial graph convolution networks (SGCN) [6] enhance traditional GCNs by incorporating node positions within the convolution operation, which enables models that utilize spatial features and learn from graphs that possess inherent spatial arrangements. SGCN extends the aggregation function as given by

$$\bar{h}_i(U, b) = \sum_{(v_i, v_j) \in E_i} ReLU(U^T(p_i - p_j) + b) \odot h_j, \tag{2}$$

where p_i and p_j are the matrix representation of the coordinate location of nodes v_i and v_j, respectively, b is a bias vector, ReLU is a rectified linear activation function, and $\odot$ represents element-wise multiplication. The extended aggregation function is utilized by the multi-layer perceptron layer, which provides an output given by

$$MLP(\bar{H}; W) = ReLU(W^T \bar{H} + b), \tag{3}$$

where W is the trainable weight matrix.

III. Graph Learning Framework

The objective of developing the graph learning framework is to efficiently and effectively address multiple classes of EDA tasks with techniques that leverage graph representation learning. The key components of the framework, which include the graph structure and feature engineering, are discussed in Section III-A. The generalized machine learning architecture used for downstream prediction is described in Section III-B.

A. Graph Structure and Feature Engineering

Circuits are represented as graphs, where nodes represent electronic components (e.g., cells, interconnects/wires, inputs, and output), and edges represent the connections between components. The graph is either directed or undirected, depending on the target problem and application. Each node and edge in the graph is associated with relevant features that capture essential information of the electronic components and the corresponding connections of each component.

1) Netlist Graph Representation: Circuit information is extracted from the Verilog and DEF files that includes the logical functionality of the gates, the input/output pin locations, the placement position/coordinates of the gates, and the interconnect routing paths. After parsing, the information is structured as a netlist graph denoted by $NG = (V, E)$, where nodes $v \in V$ correspond to inputs, outputs, and gates of a netlist, while edges $e \in E$ represent the connections between two node components. A netlist graph for a simple circuit is depicted in Fig. 2a.

2) Timing Path Graphs: Static Timing Analysis (STA) is an important step of the physical design flow that ensures timing constraints are met by first identifying and then computing the arrival time and required time of critical signal paths (timing paths). Analysis of the timing paths is essential to optimize the speed and reliability of the circuit. Subgraphs representing timing paths, denoted as timing path graphs $TPG = (V, E)$, are derived from the netlist graph NG, with an example TPG shown in Fig. 2b. Each node of a timing path subgraph is populated with a carefully selected feature set, which is listed in Table I. Detailed information on each standard cell is extracted from a technology specific LEF file. Structural features are computed from the netlist graphs, while timing features are extracted from the STA timing reports generated by the EDA tools and mapped to the netlist of the circuit. In addition, parasitic impedance features, represented as net capacitances and resistances, are extracted from the SPEF files generated by IC Compiler.

3) Interconnect Graph: By extracting detailed information from the predicted placement of interconnect segments and the corresponding spatial relationships from the netlist graph, an interconnect graph is constructed, which enables a more comprehensive analysis of the physical characteristics of the circuit interconnects. An estimate of the physical characteristics (i.e. path) of the circuit interconnects is extracted from the SPEF file generated after placement, with an example of an interconnect path shown in Fig. 3a. The spatial information is parsed and mapped onto an interconnect graph $IG = (V, E)$,

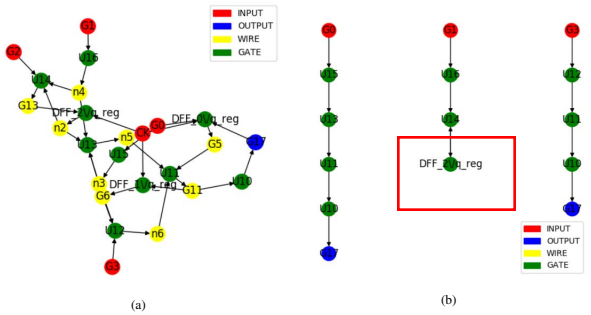

Fig. 2: Sample benchmark representation of a (a) netlist graph NG and (b) a subset of timing path graphs TPG.

TABLE I: Node features of the netlist and timing path graphs.

Feature Type	Feature	Source
Standard cell features	Width	LEF file
	Height	
	No. of input pins	
	No. of output pins	
	Is sequential gate	
	Is inverter gate	
	Is buffer gate	
	Drive strength	
Structural features	Logic level	Calculated from netlist graph representation
	Number of fan-in gates	
	Number of fan-out gates	
	Distance from closest input	
	Distance to closest output	
	Distance from closest flip-flop	
	Distance to closest flip-flop	
Spatial features	Gate co-ordinate (x, y)	DEF file
Current phase timing report features	Gate delay	STA timing report
	Arrival time	
Current phase parasitic features	Total interconnect resistance	SPEF file
	Total interconnect capacitance	

where nodes $v \in V$ represent estimated interconnect segments and edges $e \in E$ represent connections between two or more interconnect segments. The interconnect graph captures the spatial layout and connections of the circuit. An example of an interconnect graph is shown in Fig. 3b. Each node of an interconnect graph is populated with the feature set listed in Table II. The congestion features of each node of the interconnect graph are computed using the data on each interconnect segment obtained from the SPEF file. The circuit is divided into a grid of cells, each with dimensions of 10×10 μm^2, and the pin and node density are calculated for each cell. The congestion feature for the graph node representing an interconnect segment is determined by the number of pins, nets, and cells per unit area in the grid region the interconnect segment passes through. The entire interconnect graph is assigned the graph-level feature attributes listed in Table III.

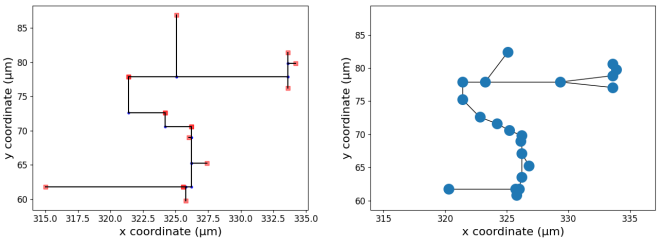

Fig. 3: Geometrical representation of the (a) predicted routing path extracted from the post placement SPEF file and (b) interconnect spatial graph based on the extracted routing path.

TABLE II: Node features of the interconnect graph.

Feature Type	Feature	Source
Structural features	Interconnect segment level	Calculated from netlist graph representation
	Interconnect segment length	
Spatial features	Interconnect segment midpoint (x, y)	Calculated from current phase netlist (DEF file) and SPEF file
Congestion features	Pin density	
	Net density	
	Cell density	
Current phase parasitic features	Interconnect segment resistance	SPEF file
	Interconnect segment capacitance	

TABLE III: Graph features of the interconnect graph.

Feature Type	Feature	Source
Structural features	Number of inputs	Calculated from netlist graph representation
	Number of outputs	
Spatial features	Half perimeter width length	Calculated from current phase netlist (DEF file) and SPEF file
Congestion features	Pin density	
	Net density	
	Cell density	
Current phase parasitic features	Total interconnect resistance	SPEF file
	Total interconnect capacitance	

B. Generalized ML flow

The proposed graph learning framework follows a generalized ML flow, which allows for the investigation of various downstream prediction tasks. The flow includes several key steps described as follows.

1) Problem Formulation: The definition of the problem (P) includes identifying the following:

- The metric (M) to be predicted.
- The initial phase (X) (e.g., post-floorplan, post-placement) and final phase (Y) (e.g., post-routing) of the design flow between which the prediction is made.
- An appropriate graph representation (G), as described in Section III-A.
- Whether the prediction is performed at the graph level (predicting a single value for the entire circuit) or node level (predicting values for individual components of the circuit).
- A baseline metric (M_b) that serves as a reference for the evaluation of the effectiveness and improvement provided by the model.

2) Network Architecture: For problem P and the selected graph representation G, a general deep neural network model includes graph convolutional layers to encode the graph representation, followed by linear layers. The output of the neural network for the objective metric M is to be modeled. To improve the learning performed by the neural network, additional inputs including the baseline reports from the initial design stage and numeric features are provided to the first linear layer.

For node-level predictions, no additional pooling layer is required. However, for graph-level predictions, a pooling layer is added between the GCN layers and linear layers. The pooling layer aggregates the node-level features obtained through convolution, which results in graph-level features that represent the entire circuit.

Graph-level attribute features are also included as inputs to the linear layers in addition to the numeric features. The inclusion of graph-level attributes provides valuable information on the overall characteristics of the circuit, augmenting the learning process and improving the predictive capabilities of the model. The overall structure of the network architecture is depicted in Fig. 4.

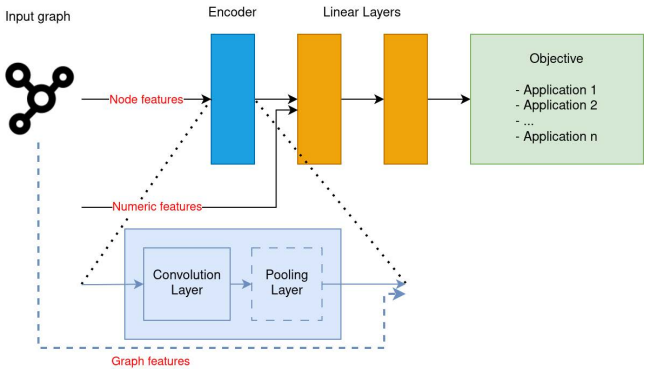

Fig. 4: Generalized template of the network architecture. The graph pooling layer and graph features are represented with dotted lines and are only used for graph level predictions.

3) Training and Evaluation: In the training and evaluation phase of the generalized ML flow, the downstream metric prediction problem P is solved utilizing a deep neural network for regression (DNNR) [7]. Primary options for DNNR include the selection of the appropriate loss function (L), optimizer (O), learning rate (LR), and batch size (B). The performance of the model is evaluated using Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) as given by, respectively,

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|M_i - M_{b,i}|, \text{ and} \tag{4}$$

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}|1 - \frac{M_{b,i}}{M_i}| \tag{5}$$

for metric M with baseline M_b, where n is the number of total samples in the overall dataset.

In addition, to account for the heavily skewed data that commonly results from the execution of various physical design algorithms [8]–[10], the worse 1% MAE and MAPE are used to characterize the performance of the model on the tail data of a standard normal distribution. The performance metrics can be expanded to include a broader range of evaluation in addition to MAPE and MAE [11].

4) Feature Importance Ranking Using Sensitivity Analysis: Sensitivity analysis evaluates feature importance in a machine learning model by leaving one feature out of each trained model. Primary features are identified that significantly influence the accuracy of the model, while also analyzing potential noise in the model, irrelevant features, and highly correlated features. Removing ineffective or redundant features based on sensitivity analysis enhances model performance by reducing overfitting and improving interpretability. The technique offers valuable insights to fine-tune and optimize the configuration of a model for better predictive performance.

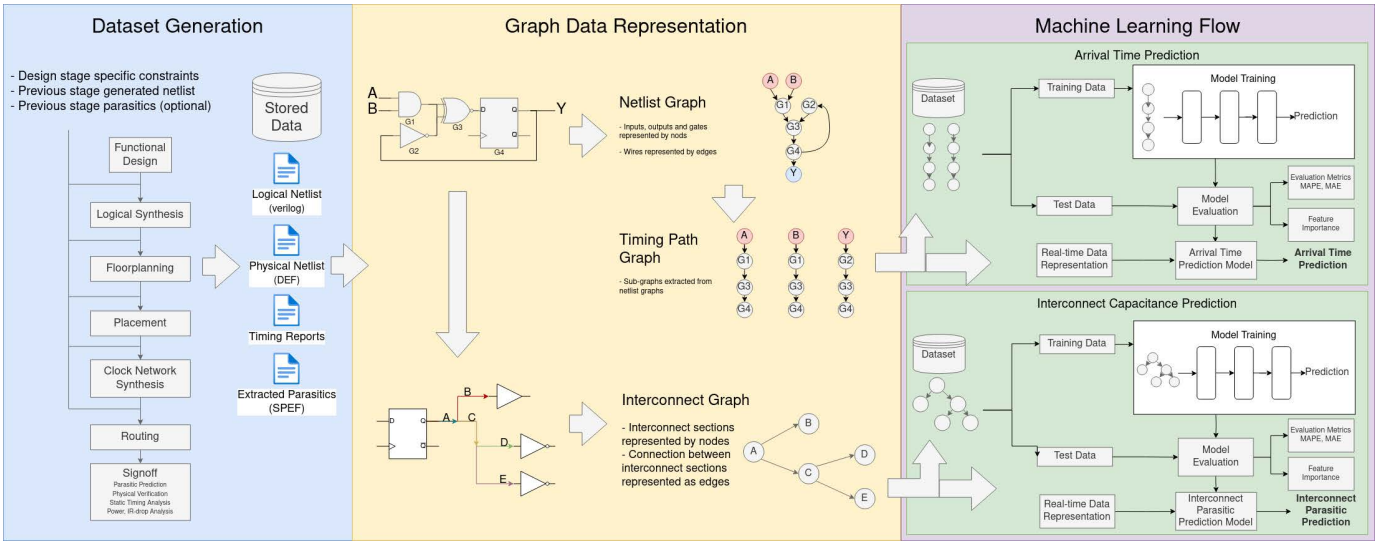

Fig. 5: Block representation of overall framework including training and test flow.

IV. Application to IC Design Automation

The effectiveness of the proposed graph learning framework is evaluated through two case studies, which demonstrate the capability of solving downstream prediction tasks. The overall execution of the framework and training process is shown in Fig. 5. A dataset of physical designs and timing profiles (i.e. STA) for the IWLS benchmark circuits is generated, as described in Section IV-A. The dataset is utilized as input for the two case studies, which are described in Section IV-B.

A. Dataset Generation

A dataset of layouts is constructed for six IWLS'05 sequential benchmark circuits [12], which are listed in Table IV. The dataset is generated on a 28 nm technology node. The benchmark circuits are initially synthesized into technology specific gate-level netlists using Synopsys Design Compiler. The gate-level netlists are then placed and routed into final layouts using Synopsys IC Compiler. Once physical design is complete, static timing analysis is performed using Synopsys PrimeTime. The circuits are designed to achieve an operating frequency of 1 GHz, while constrained to an aspect ratio of 0.5, and an area utilization of 70%.

TABLE IV: Benchmark circuits utilized to characterize the proposed ML framework.

Circuit	aes_core	des3_perf	ethernet	fpu	vga_lcd	wb_conmax
No. of nets	10448	49944	20398	29004	25662	20591
No. of timing paths	741	7878	1776	422	121	758

B. Case Studies

The dataset described in Section IV-A is utilized to analyze two distinct downstream metric prediction problems: post-floorplan to post-routing arrival time prediction and post-placement to post-routing prediction of interconnect parasitic impedance. Extending prior work [8], [9], the case studies include a larger feature set and utilize a dataset generated on a 28 nm technology node.

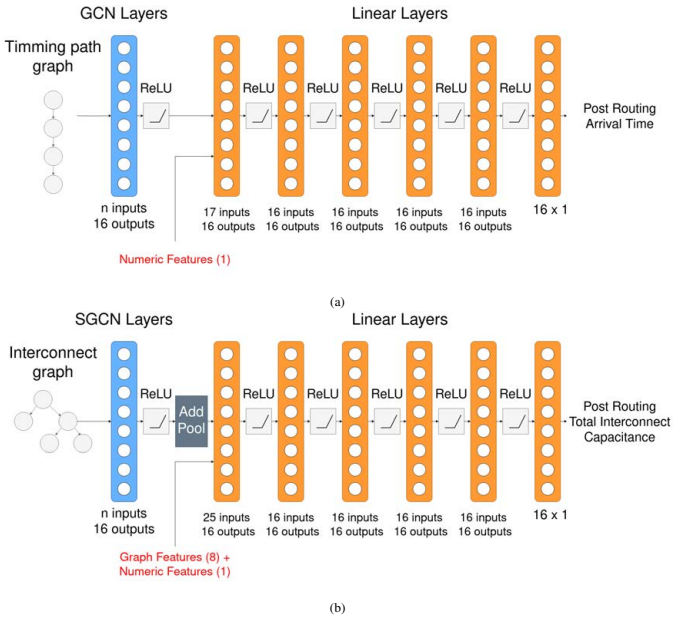

Fig. 6: Architecture of the learning network for (a) arrival time prediction and (b) interconnect parasitic impedance prediction.

The network architecture for arrival time prediction is depicted in Fig. 6a, while the network architecture utilized for prediction of the interconnect parasitic impedance is shown in Fig. 6b. Arrival time prediction requires timing path graphs with node-level aggregation and no pooling. Interconnect parasitic impedance prediction utilizes interconnect graphs and requires pooling to predict the total capacitance of a net. Arrival time prediction utilizes GCNs for encoding due to a lack of spatial information in the early stages of physical design, while interconnect parasitic prediction uses SGCNs. Both case studies utilize six linear layers of size 16, with

TABLE V: Problem formulation and model analysis details for each case study.

		Arrival Time Prediction	Interconnect Parasitic Prediction
Problem Formulation	**Metric to predict** (M)	Gate arrival time	Total interconnect capacitance
	Initial phase (X)	Post logical synthesis	Post placement
	Final phase (Y)	Post routing	Post routing
	Graph representation (G)	Timing path graph (TPG)	Interconnect graph (IG)
	Prediction level	Node level	Graph level
	Baseline (M_b)	Post logical synthesis arrival time	Post placement total interconnect capacitance
Network Architecture	**Graph embedding layer**	GCN	SGCN
	Use of graph pooling	No	Yes
Model Training and Evaluation Details	**Loss function** (L)	Mean Squared Error Loss (MSE Loss)	
	Optimizer (O)	Stochastic Gradient Descent (SGD) [13]	
	Learning Rate (LR)	1%	
	Batch Processing (B)	Yes (1024 batches per iteration)	
	Evaluation metrics	MAE, MAPE, worse 1% MAE, worse 1% MAPE	

TABLE VI: Analysis of the mean absolute percentage error (MAPE) and mean absolute error (MAE) of the post-logical synthesis to post-routing arrival time prediction.

		Including all features		Excluding irrelevant features	
Error Metric	**Baseline**	**Proposed Model**	**Improvement**	**Proposed Model**	**Improvement**
MAE (ns)	0.098	0.0657	29.72%	0.0594	48.01%
worse 1% MAE (ns)	0.262	0.1386	53.96%	0.137	53.94%
MAPE	62.28%	33.39%	39.2%	30.78%	50.85%
worse 1% MAPE	138.35%	76.83%	60.73%	79.54%	61.21%

TABLE VII: Analysis of the mean absolute percentage error (MAPE) and mean absolute error (MAE) of the post-placement to post-routing interconnect capacitance prediction.

		Including all features		Excluding irrelevant features	
Error Metric	**Baseline**	**Proposed Model**	**Improvement**	**Proposed Model**	**Improvement**
MAE (pF)	0.2538	0.1972	22.88%	0.1894	25.48%
worse 1% MAE (pF)	6.4925	5.2656	18.63%	5.4319	19.04%
MAPE	18.51%	15.82%	13.7%	13.51%	25.06%
worse 1% MAPE	136.27%	106.99%	19.64%	91.75%	31.15%

each hidden layer incorporating a rectified linear unit (ReLU) [14] as an activation function. Numeric features, including the baseline, are added to both networks.

The dataset from the six circuits listed in Table IV is divided into six instances of the neural regression model, where each model includes one circuit as the test set and the remaining five circuits as the training set. Details regarding both the setup for the training of the model and the evaluation of the model are provided through the parameters listed in Table V. The implementation of the framework is in Python 3, utilizing the PyTorch-geometric deep learning library [15]. The training is performed on a system with 96 GB memory, twenty-four 4 GHz CPU cores, and an NVIDIA GeForce GTX 3090 graphics card.

The trained models are validated against the test datasets. The MAPE, MAE, and the MAPE and MAE of the top 1% of worse case errors averaged across the results produced by each of the trained models when predicting on each circuit are listed in Table VI and Table VII for arrival time prediction and parasitic impedance prediction, respectively. Results from a comprehensive sensitivity analysis of feature importance based on average MAPE are listed in Table VIII.

1) Arrival Time Prediction: The proposed model outperforms the baseline errors provided by PrimeTime, achieving an average improvement of 39.2% in the MAPE score and 29.72% in the MAE score across the six models. In addition, for worse case predictions, the improvement is even more significant, with the proposed model providing an average improvement of 60.73% in the MAPE score and 53.96% in the MAE score over the baseline for the top 1% of arrival times with the largest estimated errors.

TABLE VIII: Selected features from sensitivity analysis of the mean absolute percentage error (MAPE) of the post-logical synthesis to post-routing arrival time prediction and post-placement to post-routing interconnect capacitance prediction. Features providing negative contribution to the model are highlighted in red.

Arrival Time Prediction		Interconnect Parasitic Prediction	
Feature	**MAPE**	**Feature**	**MAPE**
Post logical synthesis arrival time	76.31%	Post placement interconnect capacitance	116.77%
Logic level	37%	Cell density	17.23%
No. of fan-ins	36.79%	Net density	16.06%
Distance to nearest output	36.7%	Pin density	14.64%
Cell is buffer	30.55%	Interconnect length	13.51%

The sensitivity analysis reveals the significant impact of specific features on the performance of the model. Arrival time for the initial phase X emerges as the most impactful feature, followed by the position of the gate in the timing path, which represents a structural feature of the circuit and graph. In addition, removing the buffer feature improves the performance of the model. By eliminating features that introduce noise, lack relevance, or exhibit high correlation, the accuracy of the model is improved, and the risk of overfitting is reduced, which results in greater performance and interpretability. The evaluation of the models after removing the ineffective feature (is_buffer) is performed, with results listed in Table VI. The average MAE and MAPE of the models after removing the buffer feature improves from 65.7 ps to 59.4 ps and 33.39% to 30.78%, respectively.

2) Prediction of Interconnect Parasitic Impedance: The proposed model outperforms the baseline predictions provided by IC compiler, achieving an average improvement of 13.7% in the MAPE score and 22.87% in the MAE score across the six models. For worse-case predictions, the proposed model provides an average improvement of 19.74% in the MAPE score and 18.63% in the MAE score over the baseline for the top 1% of nets with the largest error in estimated parasitic impedance.

The most significant feature that improves model performance when predicting the parasitic impedance is the post-placement estimate of the interconnect capacitance. Removing the length of an interconnect as a feature also improves the performance of the model as the interconnect parasitics are already correlated with the interconnect length. The evaluation of the models after removing interconnect length is performed, with results as listed in Table VII. The average MAE and MAPE after removing the ineffective feature improves from 197.2 fF to 189.4 fF and 15.82% to 13.51%, respectively.

V. Conclusion

This paper introduces EDA-ML, a task-agnostic graph representation learning framework for IC design automation, which effectively captures complex relationships within graph-structured data and leverages machine learning techniques for downstream metric prediction tasks. The proposed standardized graph representations and feature sets outperform reported baseline errors for two distinct prediction tasks, providing significant improvements in both MAPE and MAE scores. An improvement in MAE of 28.71% and an improvement in MAPE of 39.2% is observed for arrival time prediction, and an improvement in MAE of 22.88% and an improvement in MAPE of 19.64% is observed when predicting for parasitic impedance. In addition, a sensitivity analysis is performed on the feature sets to analyze the contribution of each feature to the performance of the model. Ineffective features are removed from the model as a result of sensitivity analysis. The removal of the ineffective features when predicting arrival time further improves the MAE and MAPE to 48.01% and 50.86%, respectively. Similarly, removal of ineffective features improves the MAE and MAPE of parasitic impedance prediction to 25.48% and 25.06%, respectively. Overall, the proposed ML framework provides a means to improve integrated circuit design, and paves the way for future advancements in design automation methodologies for both research and industry.

References

[1] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K Zhong, *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 26, No. 5, pp. 1–46, Jun. 2021.

[2] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 8, pp. 1798–1828, Aug. 2013.

[3] 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, No. 1, pp. 57–81, Jan. 2020.

[4] "IEEE standard for integrated circuit (IC) open library architecture (OLA)," *IEEE Std 1481-2019 (Revision of IEEE Std 1481-2009)*, pp. 1–641, Mar. 2020.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, pp. 1–14, Sept. 2016.

[6] T. Danel, P. Spurek, J. Tabor, M. Śmieja, Ł. Struski, A. Słowik, and Ł. Maziarka, "Spatial graph convolutional networks," *Proceedings of the International Conference on Neural Information Processing (NIPS)*, pp. 668–675, Nov. 2020.

[7] C.-H. Chen, J.-P. Lai, Y.-M. Chang, C.-J. Lai, and P.-F. Pai, "A study of optimization in deep neural networks for regression," *Electronics*, Vol. 12, No. 14, pp. 3071, Jul. 2023.

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

[9] P. Shrestha and I. Savidis, "Graph representation learning for parasitic impedance prediction of the interconnect," *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May. 2023.

[10] Z. Jiang, M. Liu, Z. Guo, S. Zhang, Y. Lin, and D. Pan, "A tale of EDA's long tail: Long-tailed distribution learning for electronic design automation," *Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD*, pp. 135–141, Sept. 2022.

[11] V. Plevris, G. Solorzano, N. P. Bakas, S. Ben, and E. A. Mohamed, "Investigation of performance metrics in regression analysis and machine learning-based prediction models," *European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2022)*. European Community on Computational Methods in Applied Sciences, pp. 1–25, 2022.

[12] C. Albrecht, "IWLS 2005 benchmarks," *Proceedings of the IEEE International Workshop for Logic Synthesis (IWLS): http://www.iwls.org*, Jun. 2005.

[13] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, pp. 1–14, Sept. 2016.

[14] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," *Proceedings of the IEEE International Conference on Machine Learning*, pp. 807–814, Jun. 2010.

[15] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428*, pp. 1–9, Mar. 2019.