MTLSO: A Multi-Task Learning Approach for Logic Synthesis Optimization

Faezeh Faez[1] Raika Karimi[1] Yingxue Zhang[1]
Xing Li[2] Lei Chen[2] Mingxuan Yuan[2] Mahdi Biparva[1]

[1]Huawei Noah's Ark Lab, Toronto, Canada
[2]Huawei Noah's Ark Lab, Hong Kong, China

Outline

1. Motivation
2. Related Work
3. Methodology
4. Experiments
5. Summary

Motivation (I)

Challenges in Logic Synthesis Optimization (LSO):

- **Complexity of Modern ICs:**
  - Billions of transistors in modern ICs make manual design infeasible.
  - Traditional heuristic-based methods face limitations in achieving optimal results.
- **Machine Learning as a Solution:**
  - ML enhances LSO by enabling faster and more accurate predictions [6].
- **Data Scarcity:**
  - Limited availability of large, labeled datasets hampers machine learning models.
  - Overfitting challenges reduce the generalization and reliability of predictions.
- **Inefficiencies in Graph Encoding:**
  - Large AIGs with numerous nodes pose challenges for plain GNNs.
  - Treating all nodes with equal importance leads to suboptimal representations.

Motivation (II)

Purpose of MTLSO:

- **Addressing Data Scarcity:**
  - Multi-task learning (MTL) enables the model to leverage shared supervision from related tasks.
  - Introducing an auxiliary task (binary multi-label graph classification) enhances model robustness.
- **Improving Graph Representation:**
  - Hierarchical graph representation learning captures multi-level abstractions of AIGs.
  - Combines GNNs with graph downsampling for better scalability and expressiveness.

Related Work

Key Areas in Related Research:

- **Logic Synthesis Optimization (LSO):**
  - A growing trend toward employing ML techniques for EDA tasks, moving away from traditional hand-engineered approaches.
  - Techniques use GNNs, LSTMs, or Transformers for graph and recipe representation learning.
  - Challenges:
    - Inefficiencies in encoding large AIGs due to treating all graph nodes with equal importance.
    - Overfitting due to data scarcity.
- **Multi-task Learning (MTL):**
  - Enhances model generalization by leveraging shared data across multiple related tasks.
  - Demonstrated success in NLP, vision, and speech for mitigating data scarcity challenges.
  - Despite its potential to address the data scarcity challenge, MTL remains underutilized in LSO.

Methodology

- Problem Formulation
- Graph Encoder
- Recipe Encoder
- Multi-Task Learning

Methodology

- **Problem Formulation**
- Graph Encoder
- Recipe Encoder
- Multi-Task Learning

Objective

Predict QoR value for each pair (G, r_i) where G is an AIG and r_i is a synthesis recipe.

$$f : \mathcal{G} \times \mathcal{R} \rightarrow \mathbb{R}$$

- $\mathcal{G}$: Set of AIGs
- $\mathcal{R}$: Set of synthesis recipes

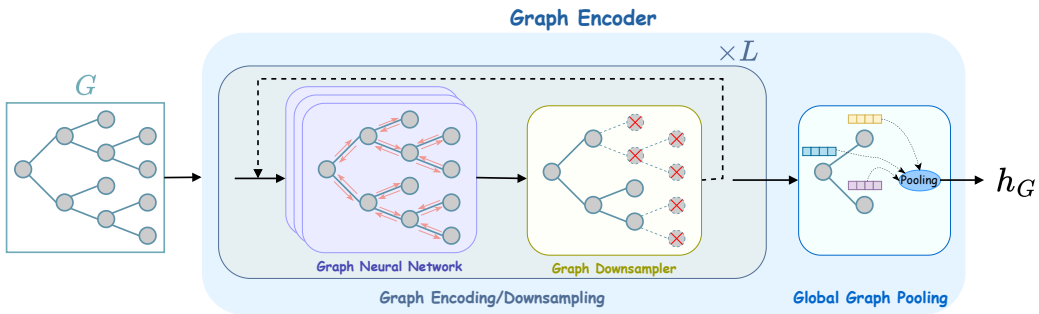
Methodology

- Problem Formulation
- **Graph Encoder**
- Recipe Encoder
- Multi-Task Learning

Methodology → Graph Encoder (Overview)

Goal: Learn a meaningful graph-level representation h_G of a large AIG G using GNNs.

- **Setup:**
  - Input graph G with node feature matrix X.
  - Output is a vector representation $h_G \in \mathbb{R}^F$ for the entire graph.
- **Challenge:** Large AIGs can consist of thousands of nodes with varying importance for QoR prediction, rendering conventional plain GNNs less efficient.
- **Solution:**
  - A **Hierarchical Graph Representation Learning (HGRL)** approach.
  - Iteratively remove unimportant nodes to focus on critical subgraphs.

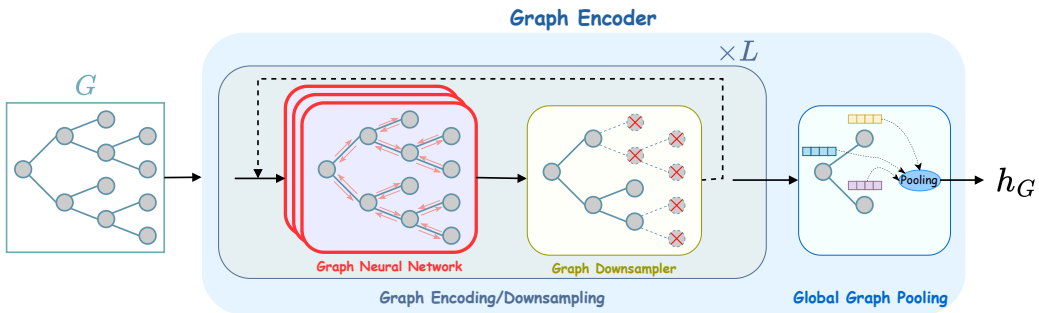


Methodology → Graph Encoder → Hierarchical Graph Representation Learning (I)

Layer-by-Layer GNN Encoding:

- HGRL stacks L GNN layers sequentially to process the graph.
- At layer l, the graph G^l is defined by:
  - Node feature matrix $X^l \in \mathbb{R}^{N^l \times F^l}$
  - Adjacency matrix $A^l \in \{0,1\}^{N^l \times N^l}$
- Node representations are updated via message passing:

$$H^{l+1} = \text{GNN}(X^l, A^l), \quad H^{l+1} \in \mathbb{R}^{N^l \times F^{l+1}}$$



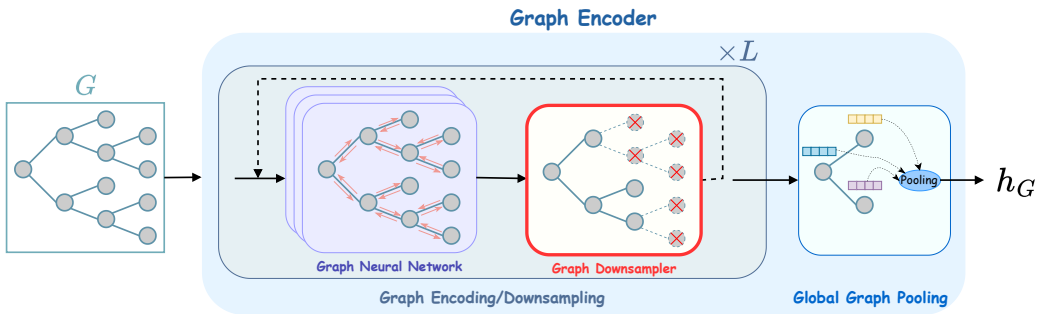
Methodology → Graph Encoder → Hierarchical Graph Representation Learning (II)

Node Downsampling:

- Select the top $\lceil \alpha N^l \rceil$ nodes by computing scores as the projection of H^{l+1} onto a learnable vector, capturing node importance.
- Remove nodes with lower scores, reducing graph size for the next layer:

$$A^{l+1}, X^{l+1} = \textsc{GraphDownsample}(A^l, H^{l+1})$$

- Outputs:
  - A^{l+1}: Pruned adjacency matrix.
  - X^{l+1}: Representation matrix for the remaining nodes.



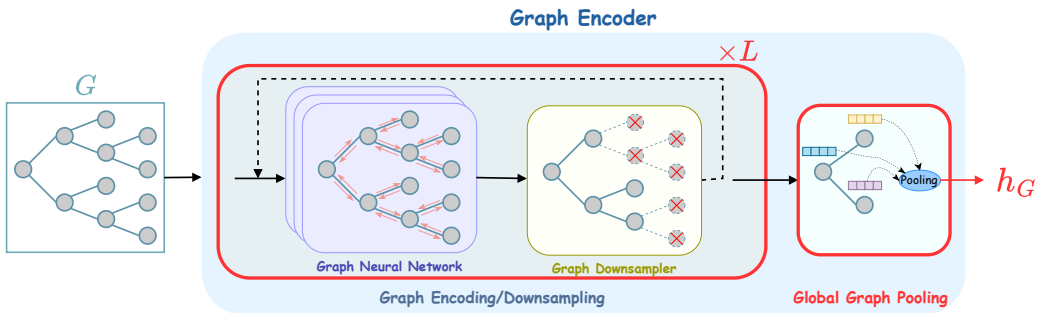
Methodology → Graph Encoder → Hierarchical Graph Representation Learning (III)

Final Pooling:

- L consecutive encoding and downsampling steps yield A^L and X^L as the pruned adjacency matrix and final node embeddings.
- A global pooling module aggregates node embeddings into a single vector:

$$h_G = \textsc{GraphPool}(X^L).$$

- The vector $h_G \in \mathbb{R}^F$ serves as the graph representation.



Methodology

- Problem Formulation
- Graph Encoder
- **Recipe Encoder**
- Multi-Task Learning

Methodology → Recipe Encoder (I)

- A synthesis recipe r_i consists of a sequence of n transformations:

$$r_i = [t_{i1}, t_{i2}, \ldots, t_{in}],$$

  where each transformation t_{ij} falls into one of several categories.

- To predict the QoR for a given pair of an AIG and a recipe, it is essential to learn a meaningful representation of the recipe.

Steps for Representation Learning:

1. **Embedding:**
   - Convert transformations t_{ij} into dense, continuous vectors:

$$\mathbf{e}_i = \textsc{RecipeEmbed}(r_i), \quad \mathbf{e}_i \in \mathbb{R}^{n \times p}.$$

   - Captures patterns and dependencies in the recipe.

Steps for Representation Learning:

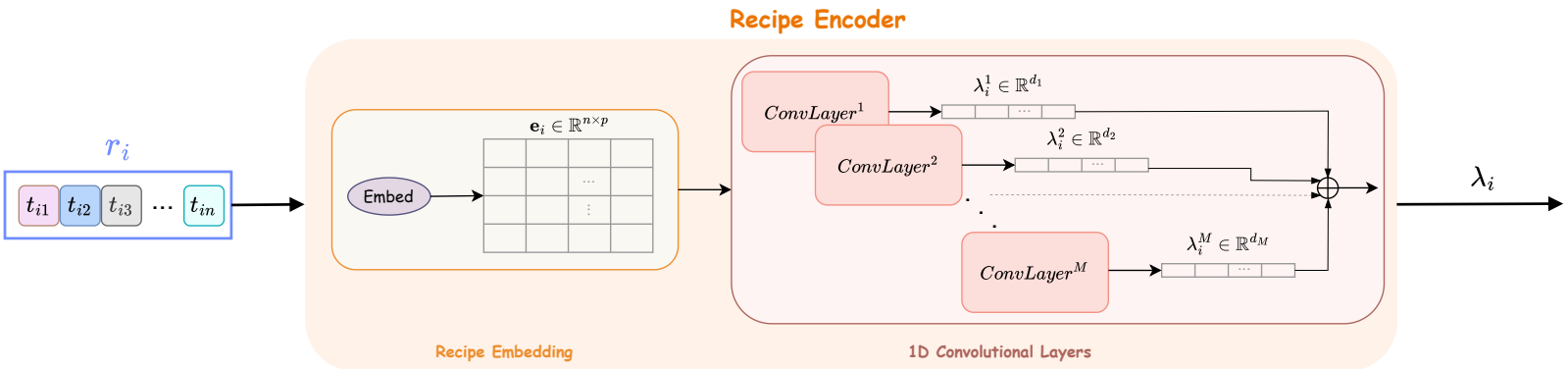
2. **Convolutional Layers:**

- Capturing relationships between adjacent transformations and extracting features with M one-dimensional convolutional layers. The m-th convolutional layer is denoted as:

$$\lambda_i^m = \textsc{ConvLayer}^m(\mathbf{e}_i).$$

- Final representation of recipe r_i:

$$\lambda_i = \text{Concat}(\lambda_i^1, \lambda_i^2, \ldots, \lambda_i^M).$$



Methodology

- Problem Formulation
- Graph Encoder
- Recipe Encoder
- **Multi-Task Learning**

Methodology → Multi-Task Learning

Objective: Address overfitting caused by data scarcity in the Logic Synthesis Optimization using a multi-task learning approach.
Primary Task: QoR value regression for each pair (G, r_i).
Auxiliary Task: Binary multi-label graph classification.

- Aids in training by providing additional signals.
- Helps signify recipe relevance to an AIG during inference.

Total Loss Function:

$$\mathcal{L} = \mathcal{L}_{\text{classification}} + \gamma \mathcal{L}_{\text{regression}}$$

Methodology → Multi-Task Learning → Binary Multi-Label Graph Classification

Objective: Predict whether each of the K recipes performs well for a given AIG G.

Label Construction:

- Select the $\lceil \rho K \rceil$ best recipes with the lowest QoR values.
- The label for each recipe r_i with respect to AIG G is defined as:

$$c_i^G = \begin{cases} 1 & \text{if } r_i \text{ is among the top-performing recipes for } G, \\ 0 & \text{otherwise.} \end{cases}$$

Classifier:

- Input: Graph representation h_G.
- Output: $P^G_{\text{classification}} \in [0,1]^K$ (predicted probabilities for each recipe):

$$P^G_{\text{classification}} = \textsc{GraphClassify}(h_G)$$

- Classification Loss:

$$\mathcal{L}_{\text{classification}} = \textsc{BinaryCrossEntropy}(P^G_{\text{classification}}, C^G)$$

Methodology → Multi-Task Learning → QoR Value Regression

Objective: Predict QoR value y_i^G for a given pair (G, r_i).

Inputs:

- Graph representation h_G.
- Recipe representation λ_i.
- Classification probabilities $P^G_{\text{classification}}$.

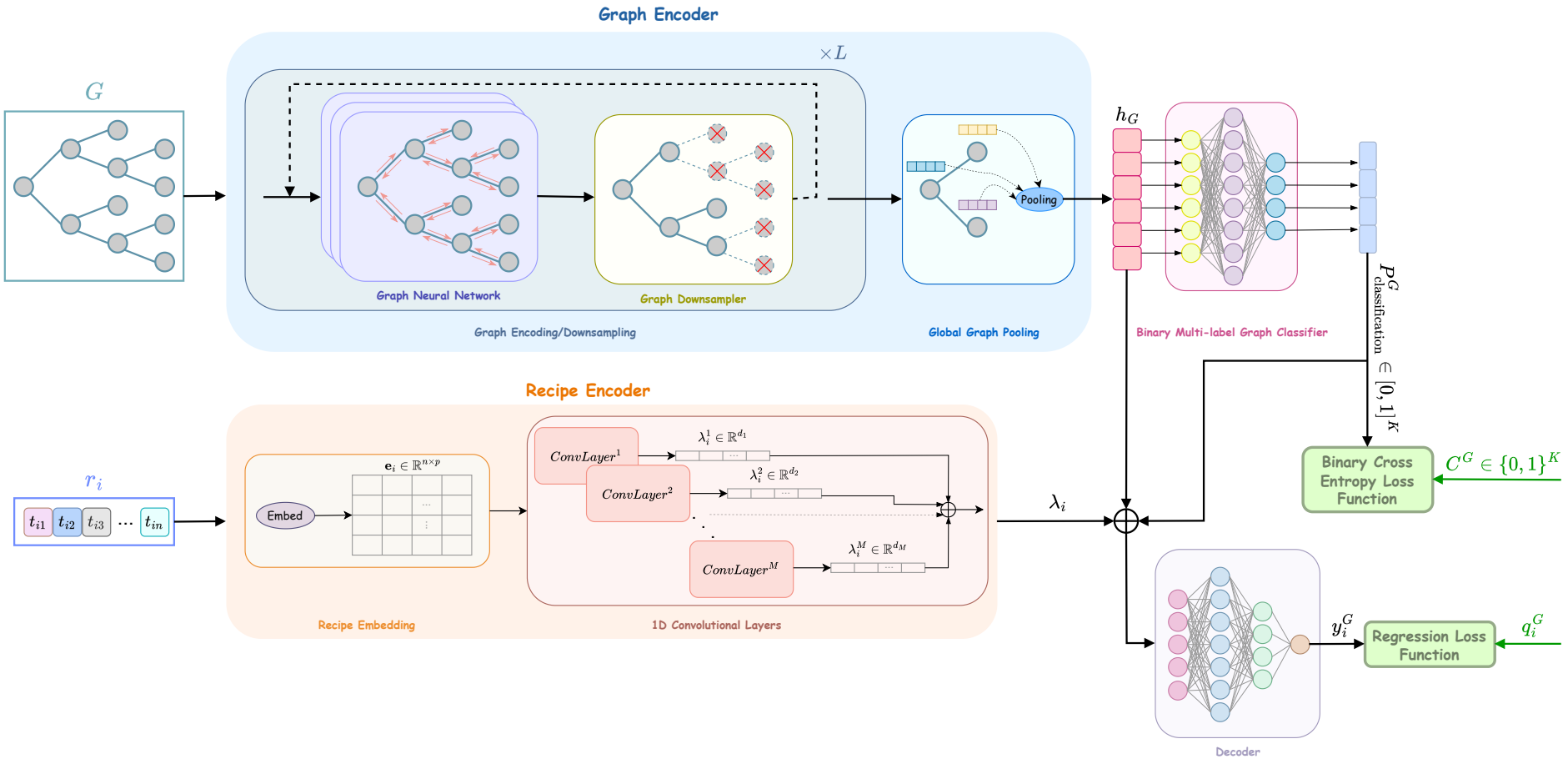
Output:

$$y_i^G = \textsc{Decoder}(\text{Concat}(h_G, \lambda_i, P^G_{\text{classification}}))$$

Regression Loss:

$$\mathcal{L}_{\text{regression}} = \textsc{RegressionLoss}(y_i^G, q_i^G)$$

Methodology → Overview



Experiments

- Datasets
- Baselines
- Metrics
- Implementation Details
- Results
- Ablation Study

Experiments

- **Datasets**
- Baselines
- Metrics
- Implementation Details
- Results
- Ablation Study

Experiments → Datasets

Dataset	Min #Nodes	Max #Nodes	Avg #Nodes	#Graphs
OpenABC-D	597	139,719	36,959.92	26
EPFL	207	57,503	15,833.40	15
CD	77	55,332	21,746.99	118

Table: Statistics of datasets used in the experiments.

- **OpenABC-D** [3]
- **EPFL** [1]
- **Commercial Dataset (CD)**: Proprietary dataset.

Experiments

- Datasets
- **Baselines**
- Metrics
- Implementation Details
- Results
- Ablation Study

Experiments → Baselines

- **Chowdhury et al. [3]**:
  - GCN [7] for AIG encoding.
  - 1D Convolutional layers for recipe encoding.
  - The two representations are concatenated to predict the QoR.
- **LOSTIN [8]**:
  - GIN [10] for AIG encoding.
  - LSTM for recipe representation learning.
  - The two representations are concatenated to predict the QoR.
- **GNN-H [9]**:
  - Adopting a similar strategy as LOSTIN [8], but utilizing PNA [4] for AIG encoding.
- **Yang et al. [11]**:
  - GraphSage [5] as the GNN.
  - Transformer for recipe encoding.

Experiments

- Datasets
- Baselines
- **Metrics**
- Implementation Details
- Results
- Ablation Study

Experiments → Metrics

- **Mean Absolute Percentage Error (MAPE)**:

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

  - Measures the average absolute percentage difference between actual and predicted QoR.
  - Lower MAPE indicates better performance.

Experiments

- Datasets
- Baselines
- Metrics
- **Implementation Details**
- Results
- Ablation Study

Experiments → Implementation Details

- **Framework**: PyTorch.
- **Graph Encoder**:
  - 2 layers of Graph Encoding/Downsampling ($L = 2$).
  - A 2-layer GCN [7] as our GNN, with a hidden layer size of 64.
- **Graph Downsampler**:
  - TopKPooling [2] with $\alpha = 0.5$.
- **Graph Pooling**:
  - Multi-readout: Mean + Max pooling.
  - Final graph-level representation: 128-dimensional.
- **Recipe Encoder**:
  - Embedding size: 60.
  - 4 one-dimensional convolutional layers.
- **Label Construction**:
  - $\rho = 0.5$ for selecting top recipes.
- **Data Split**:
  - 2/3 for training, 1/3 for testing.

Experiments

- Datasets
- Baselines
- Metrics
- Implementation Details
- **Results**
- Ablation Study

Experiments → Results → Delay

Metric	Dataset	Methods				
		Chowdhury et al. [3] (Y_1)	LOSTIN [8] (Y_2)	GNN-H [9] (Y_3)	Yang et al. [11] (Y_4)	**MTLSO** (X)
Delay	OpenABC-D	23.66 ± 0.19	24.61 ± 0.03	24.31 ± 0.27	24.58 ± 0.13	**22.93 ± 0.23**
	EPFL	4.18 ± 0.07	3.96 ± 0.02	3.96 ± 0.03	3.96 ± 0.02	**3.94 ± 0.01**
	CD	15.88 ± 0.41	16.76 ± 0.13	16.80 ± 0.06	17.09 ± 0.08	**13.75 ± 0.25**

Dataset	Gain (%)			
	$\frac{Y_1-X}{Y_1} \times 100$	$\frac{Y_2-X}{Y_2} \times 100$	$\frac{Y_3-X}{Y_3} \times 100$	$\frac{Y_4-X}{Y_4} \times 100$
OpenABC-D	3.09	6.83	5.68	6.71
EPFL	5.74	0.51	0.51	0.51
CD	13.41	17.96	18.15	19.54

Table: Comparative Results in Terms of MAPE (Avg. ± Std.) for Delay (lower is better).

- MTLSO outperforms all baselines across all datasets.
- It achieves an average gain of 8.22% in delay across all baselines and datasets.

Experiments → Results → Area

Metric	Dataset	Methods				
		Chowdhury et al. [3] (Y_1)	LOSTIN [8] (Y_2)	GNN-H [9] (Y_3)	Yang et al. [11] (Y_4)	**MTLSO** (X)
Area	OpenABC-D	2.71 ± 0.02	2.35 ± 0.07	**2.33 ± 0.06**	3.77 ± 0.00	2.57 ± 0.03
	EPFL	2.46 ± 0.03	2.30 ± 0.01	2.34 ± 0.02	2.39 ± 0.00	**2.23 ± 0.04**
	CD	3.57 ± 0.10	3.46 ± 0.17	3.59 ± 0.12	3.81 ± 0.05	**3.33 ± 0.08**

Dataset	Gain (%)			
	$\frac{Y_1-X}{Y_1} \times 100$	$\frac{Y_2-X}{Y_2} \times 100$	$\frac{Y_3-X}{Y_3} \times 100$	$\frac{Y_4-X}{Y_4} \times 100$
OpenABC-D	5.17	-9.36	-10.30	31.83
EPFL	9.35	3.04	4.70	6.69
CD	6.72	3.76	7.24	12.60

Table: Comparative Results in Terms of MAPE (Avg. ± Std.) for Area (lower is better).

- MTLSO achieves an average gain of 5.95% in area across all baselines and datasets.

Experiments → Results → Key Insights

- **Performance Improvements**
  - Consistent gains in Delay and Area with MTLSO.
  - Powered by the combination of multi-task learning and hierarchical graph representation learning.
- **Future Enhancement Opportunities**
  - Improvements achieved with simple GNNs (i.e., GCN [7]) and basic recipe encoders (i.e., 1D convolution layers).
  - Greater potential with advanced GNNs (e.g., GIN [10]) and more sophisticated recipe encoders (e.g., LSTM, Transformer).

Experiments

- Datasets
- Baselines
- Metrics
- Implementation Details
- Results
- **Ablation Study**

Goal: Assess the impact of two components:

1. Replacing Hierarchical Graph Representation Learning (HGRL) with Plain Graph Representation Learning (PGRL).
2. Retaining the HGRL module from MTLSO but removing the graph classification task, resulting in a Single-task Learning (STL) setup.

Findings:

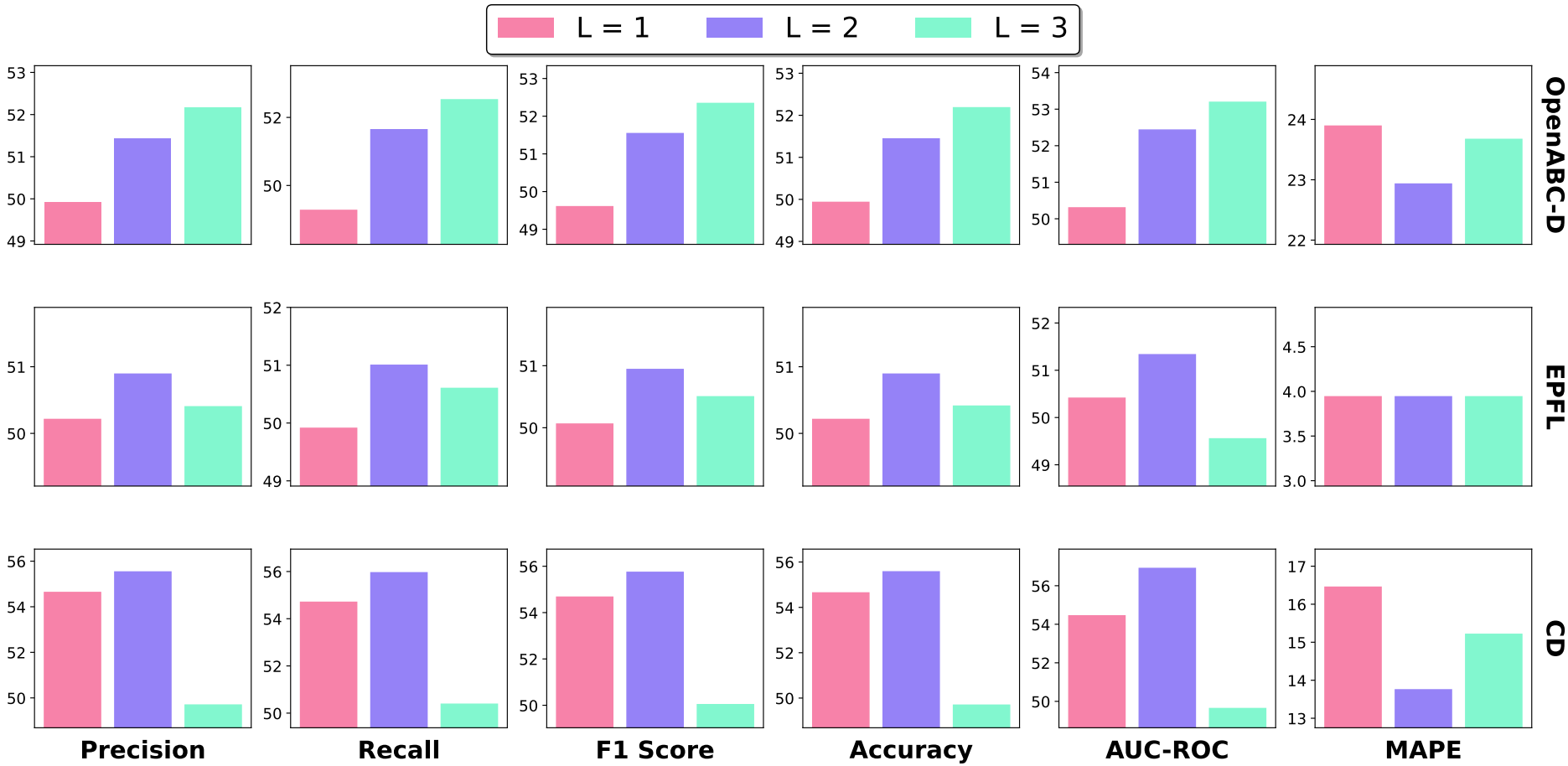
- Both multi-task learning and HGRL contribute significantly to better performance.
- Multi-task learning is more critical than HGRL, as even a simpler PGRL (trained in multi-task mode) surpasses the single-task (STL) variant.

Table: Ablation Study Results of Model Components in Terms of MAPE.

	Delay			Area		
	OpenABC-D	EPFL	CD	OpenABC-D	EPFL	CD
PGRL	23.49%	3.95%	16.48%	2.99%	2.24%	3.44%
STL	23.61%	4.51%	15.56%	2.69%	2.57%	3.45%
MTLSO	**22.93%**	**3.94%**	**13.75%**	**2.57%**	**2.23%**	**3.33%**

Experiments → Ablation Study → Number of Graph Encoding/Downsampling Layers (I)

- Ablation on the number of Graph Encoding/Downsampling layers (L).

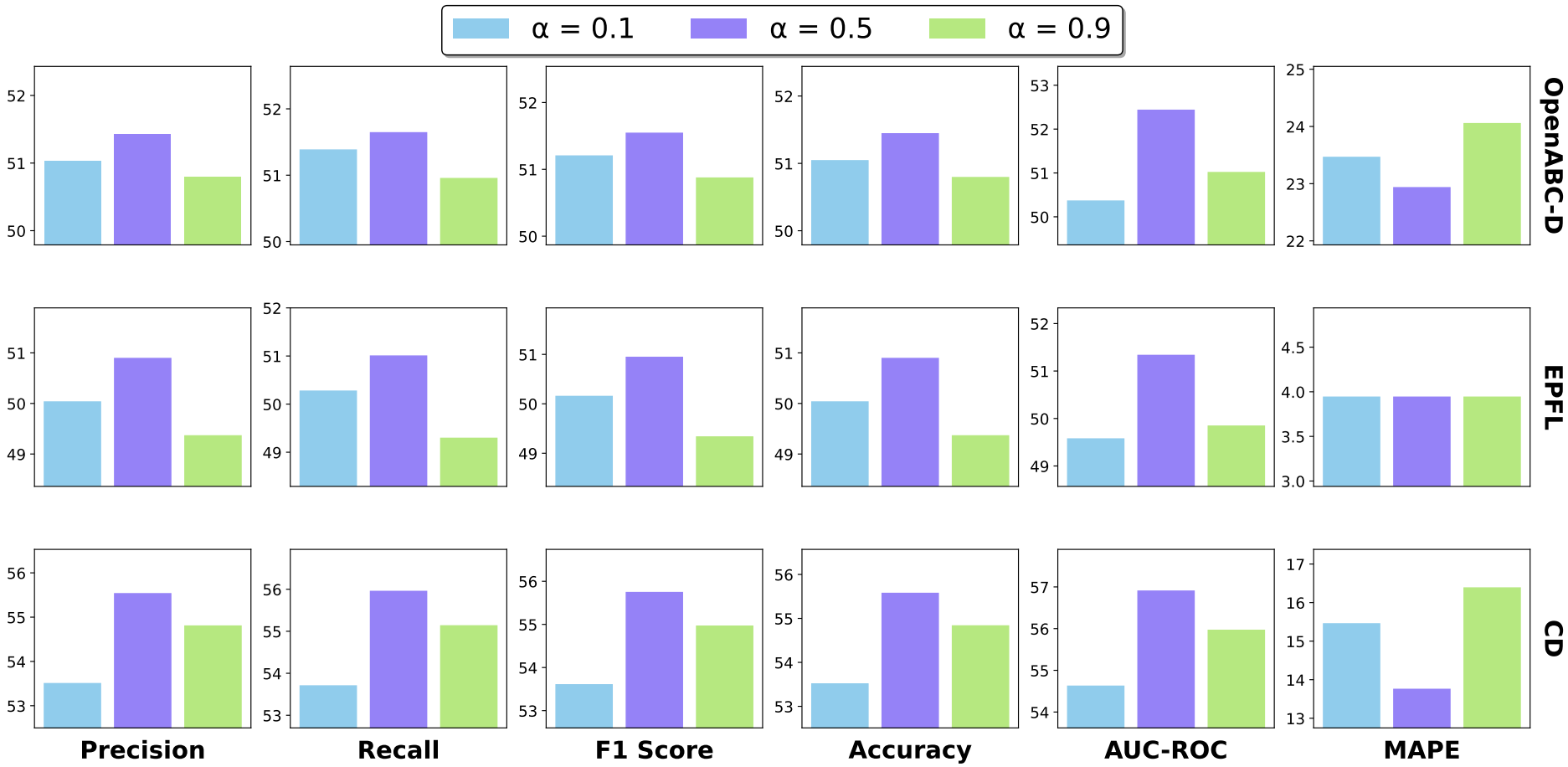


Experiments → Ablation Study → Number of Graph Encoding/Downsampling Layers (II)

- More than one encoding layer consistently improves performance, highlighting the need for a hierarchical strategy for large AIGs.
- Optimal L is 2 for EPFL and CD, and 3 for OpenABC-D based on classification metrics.
    - Tuning L for each dataset can further enhance graph representation quality.

Experiments → Ablation Study → Node Retainment Ratio (α) (I)

- Ablation on the node retainment ratio α in the graph downsampling module.



Experiments → Ablation Study → Node Retainment Ratio (α) (II)

- $\alpha = 0.5$ yields the best performance, surpassing the extremes ($\alpha = 0.1$ or $\alpha = 0.9$).
- Retaining too many nodes ($\alpha = 0.9$) degrades performance, indicating some nodes are less informative.
- Pruning too aggressively ($\alpha = 0.1$) harms the results, indicating that certain nodes play a significant role in the graph-level representation.
- The findings highlight:
  - The importance of adopting such a hierarchical strategy for encoding AIGs.
  - The need to set an optimal value for this hyperparameter.

MTLSO: Summary

- **Novel Multi-Task Learning Approach for LSO:**
  - MTLSO mitigates overfitting caused by limited data availability.
  - Combines multi-label graph classification and regression tasks.
- **Hierarchical Graph Encoding:**
  - Employs multiple layers of Graph Encoding/Downsampling.
  - Effectively handles large, complex AIGs where plain GNNs struggle.
- **Key Results:**
  - **Delay Minimization:** +8.22% improvement.
  - **Area Minimization:** +5.95% improvement.

Any Questions?

References I

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The epfl combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

[2] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

[3] Animesh Basak Chowdhury, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis. *arXiv preprint arXiv:2110.11292*, 2021.

[4] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33: 13260–13271, 2020.

References II

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[6] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, et al. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(5):1–46, 2021.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] Nan Wu, Jiwon Lee, Yuan Xie, and Cong Hao. Lostin: Logic optimization via spatio-temporal information with hybrid graph models. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 11–18. IEEE, 2022.

References III

[9] Nan Wu, Yuan Xie, and Cong Hao. Ai-assisted synthesis in next generation eda: Promises, challenges, and prospects. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 207–214. IEEE, 2022.

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

[11] Chenghao Yang, Yinshui Xia, and Zhufei Chu. The prediction of the quality of results in logic synthesis using transformer and graph neural networks. *arXiv preprint arXiv:2207.11437*, 2022.