Transferable Parasitic Estimation via Graph Contrastive Learning and Label Rebalancing in AMS Circuits

Shan Shen[1,2], Shenglu Hua[3], Jiajun Zou[1], Jiawei Liu[3], Jianwang Zhai[3], Chuan Shi[3], and Wenjian Yu[2]

[1]*Nanjing University of Science and Technology, Nanjing 210094, China*
[2]*Tsinghua University, Beijing 100084, China*
[3]*Beijing University of Posts and Telecommunications, Beijing 100876, China*

Abstract—Graph representation learning on Analog-Mixed Signal (AMS) circuits is crucial for various downstream tasks, e.g., parasitic estimation. However, the scarcity of design data, the unbalanced distribution of labels, and the inherent diversity of circuit implementations pose significant challenges to learning robust and transferable circuit representations. To address these limitations, we propose CircuitGCL, a novel graph contrastive learning framework that integrates representation scattering and label rebalancing to enhance transferability across hetero-geneous circuit graphs. CircuitGCL employs a self-supervised strategy to learn topology-invariant node embeddings through hyperspherical representation scattering, eliminating dependency on large-scale data. Simultaneously, balanced mean squared error (BMSE) and balanced softmax cross-entropy (BSCE) losses are introduced to mitigate label distribution disparities between circuits, enabling robust and transferable parasitic estimation. Evaluated on parasitic capacitance estimation (edge-level task) and ground capacitance classification (node-level task) across TSMC 28nm AMS designs, CircuitGCL outperforms all state-of-the-art (SOTA) methods, with the R^2 improvement of $33.64\% \sim 44.20\%$ for edge regression and F1-score gain of $0.9\times \sim 2.1\times$ for node classification. Our code is available at https://github.com/ShenShan123/CircuitGCL.

I. INTRODUCTION

Modern Analog-Mixed Signal (AMS) circuits, which inte-grate analog blocks (e.g., amplifiers, oscillators) with digital subsystems (e.g., controllers, SRAM arrays), demand exten-sive manual iterations during design. Engineers must balance conflicting requirements—analog components require precise tuning of electrical parameters (gain, linearity), while digital blocks prioritize timing closure and power efficiency. Post-layout parasitic effects (e.g., unintended capacitive coupling) further compound this complexity, often necessitating time-consuming revisions across schematic design, layout optimiza-tion, and iterative transistor-level simulations. For instance, in high-speed SRAM designs, coupling capacitance between adjacent interconnects can degrade signal integrity, requiring weeks of manual adjustments to meet yield targets.

Recently, Deep Learning (DL) methods based on Neural Networks (NN) have offered transformative solutions to reduce the design complexity of AMS circuits. Among them, Graph Neural Networks (GNNs) natively model circuits as graphs,
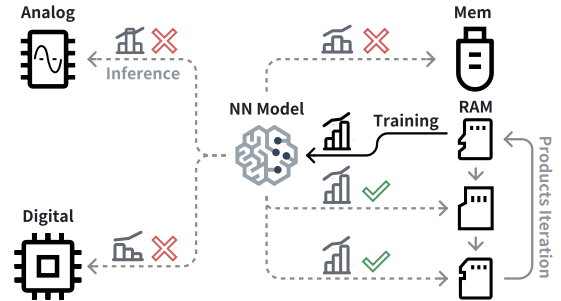
Fig. 1: NN-based DL model has poor transferability due to circuit heterogeneity.

where nodes represent components (i.e., transistors, nets) and edges encode connectivity or coupling effects [1]–[4]. By treating parasitics as learnable edge or node attributes, this enables pre-layout prediction of parasitic capacitance (a task conventionally deferred to post-layout verification), signifi-cantly reducing the need for iterative layout-simulation loops.

However, high-quality AMS circuit data, including SPICE netlists, layout parasitics, and performance metrics, is often proprietary and expensive to generate. This *scarcity* limits the adoption of large-scale GNNs (e.g., deep GNNs or graph transformers) in mixed-signal design flows. Consequently, supervised approaches struggle in this regime, resulting in overfitting and poor robustness when applied to unseen circuit topologies or advanced semiconductor technologies.

Furthermore, AMS circuits exhibit inherent *diversity*, span-ning analog, digital, and mixed-signal domains, as illustrated in Fig. 1. Each type of circuit is governed by distinct design principles and performance requirements. Memory circuits, which integrate analog and digital subsystems, exacerbate this variability. Traditional NNs trained on specific circuit types often fail to generalize to others due to differences in topology, operating regimes, and optimization objectives. While they show promise for predicting similar designs or performing iterative tasks within a product family, the lack of cross-domain transferability often leads to costly retraining or dataset regeneration, thereby limiting the extensive usage of DL-driven Electronic Design Automation (EDA) tools.

The combination of scarcity and diversity in circuit data further leads to a label *imbalance* problem, which fundamen-tally constrains the transferability of GNNs. This imbalance is inherently prevalent in AMS circuit datasets, where label

distributions are often skewed with long-tailed patterns rather than uniformly distributed across categories. For instance, parasitic capacitors with larger capacitance, which result in severe timing violations and signal integrity, are significantly underrepresented, as reported in recent studies [1], [2]. Such imbalance significantly undermines both the generalizability and fairness of Graph Neural Networks (GNNs), presenting a critical challenge that demands dedicated efforts to enhance data-driven transferability.

Can state-of-the-art transfer learning techniques address data scarcity and diversity in the EDA domain? In this work, we answer this question affirmatively by proposing **CircuitGCL**, a framework that integrates a Representation Scattering Mechanism (RSM) into Graph Contrastive Learning (GCL) and employs label rebalancing techniques. Specifically, we focus on parasitic capacitance estimation at the pre-layout stage. By modeling circuit nets as nodes and coupling effects as edges, we evaluate the proposed method on two downstream tasks: (i) *edge regression* to estimate coupling capacitance values and (ii) *node classification* to categorize the ground capacitance of each net into discrete ranges (small/medium/large).

Our key contributions, summarized in Fig. 2, are as follows:

- We adapt the Representation Scattering Mechanism (RSM) for GCL and demonstrate that it generates transferable representations for various circuit graphs. These representations are directly applicable to other unseen AMS designs without any task-specific fine-tuning.
- We address data imbalance in circuit datasets through label rebalancing, enhancing model transferability across domains. For regression tasks, we adopt balanced Mean Squared Error (MSE), while balanced softmax Cross-Entropy (BSCE) is applied to classification tasks.
- We deem the above two contributions make Circuit-GCL extend directly to resistance/inductance prediction, crosstalk analysis, IR drop estimation, and cross-technology transfer.

II. PRELIMINARY

We first introduce the basic task-related background of parasitic estimation and some preliminaries about GCL and imbalanced classification and regression.

A. Parasitic Capacitance Estimation

The design of AMS circuits typically requires extensive manual intervention, relying heavily on iterative topology selection and component sizing. In traditional workflows, IC engineers optimize circuits through pre-layout simulations and verify designs using post-layout simulations. However, as technology scales to advanced nodes, reduced feature sizes, tighter spacing, and lower supply voltages collectively amplify parasitic effects. These effects introduce significant discrepancies between pre-layout and post-layout simulation results, with parasitic capacitance emerging as a critical factor that can no longer be neglected during early design stages [5]–[7]. Parasitic capacitance arises from unintended capacitive

coupling between conductive structures, electric field penetration through dielectrics, and non-ideal charge accumulation at interfaces. It is categorized into two types: (1) ground capacitance (between interconnects and the substrate) and (2) coupling capacitance (between adjacent interconnects). These parasitics degrade circuit performance by increasing propagation delays, raising power consumption, and compromising signal integrity.

To address these challenges, graph neural networks have been adopted for parasitic capacitance prediction. For instance, ParaGraph [1] converts circuit schematics into graphs and employs message-passing neural network (MPNN) layers to predict net capacitance and layout parameters. The framework uses three ensemble models, selecting the best-performing output to mitigate label imbalance in lumped capacitance predictions. Similarly, Shen et al. [2] developed DLPL-Cap, a deep learning model combining a GNN router with five expert regressors to handle imbalanced data distributions in SRAM circuits during pre-layout parasitic estimation. However, both works treat coupling capacitance as a component of lumped capacitance, limiting their granularity.

Recent advancements, such as CircuitGPS [3], propose few-shot learning for parasitic prediction using small-hop sub-graph sampling, a low-cost positional encoding (double-anchor shortest path distance, DSPD), and pre-training strategies. While DSPD reduces computational complexity compared to traditional positional encodings, its time and storage costs scale poorly with graph size, necessitating restrictive 1-hop subgraph sampling. This limitation inspired our use of GCL to generate initial node embeddings. Additionally, CircuitGPS manually constructs negative edge samples without accounting for imbalanced label distributions — a gap our method explicitly tackles.

B. Graph Conversion of AMS Circuits

The schematic netlist of an AMS circuit is modeled as a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ represents the set of nodes with attribute matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges. The adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ is defined such that $\mathbf{A}_{ij} = 1$ if an edge $(v_i, v_j) \in \mathcal{E}$ exists, and $\mathbf{A}_{ij} = 0$ otherwise. The degree matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ is diagonal, with entries $d_i = \sum_{j \in \mathcal{V}} \mathbf{A}_{ij}$ for each node v_i.

Following the conversion from [3], nodes in $\mathcal{V}$ are categorized into three types: nets, transistor devices, and pins (device terminals). Edges in $\mathcal{E}$ encode the schematic topology as either device-to-pin or net-to-pin connections. Coupling capacitance, which constitutes the prediction target, is excluded from $\mathcal{G}$ and modeled as candidate edges. These include three subtypes: pin-to-net, pin-to-pin, and net-to-net coupling. To simplify GNN models, heterogeneous AMS graphs are converted into homogeneous graphs by assigning node type attributes $\mathbf{X} \in \{0, 1, 2\}^{N \times 1}$, where each entry corresponds to a node type.

To address the specific requirements of downstream regression tasks, particularly coupling capacitance prediction, the model incorporates an enhanced feature matrix $\mathbf{X}_C \in \mathbb{R}^{N \times d_C}$
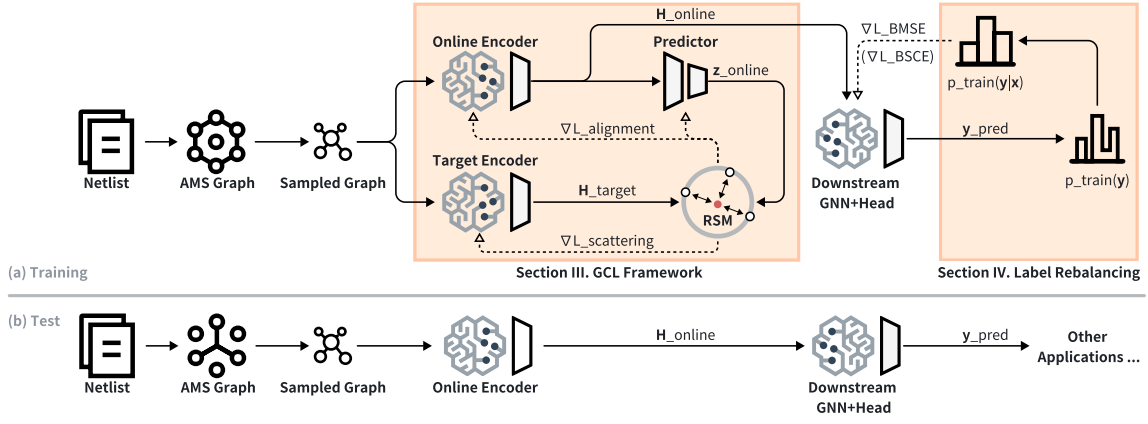
Fig. 2: Workflow of CircuitGCL. (a) During training, a target encoder applies a Representation Scattering Mechanism (RSM) to generate scattered embeddings ($\mathbf{H}_{\text{target}}$), while an online encoder produces embeddings ($\mathbf{H}_{\text{online}}$) that are passed to a downstream GNN. To improve transferability, a label rebalancing module adjusts the final loss based on the training label distribution, $p_{\text{train}}(\boldsymbol{y})$. (b) During testing, only the trained online encoder and downstream GNN are utilized to generate predictions ($\boldsymbol{y}_{\text{pred}}$).

that captures detailed design parameters and connectivity statistics. For net nodes, this matrix encodes comprehensive connectivity information, including the count and geometric properties of connected components such as transistors, capacitors, and resistors, along with their dimensional characteristics like width and length. Device nodes are characterized by their intrinsic parameters, including multiplier values, geometric dimensions, and device type codes. Pin nodes are distinguished by their functional roles within device instances, such as gate, drain, source, or bulk terminals in MOS devices. Notably, our method does not utilize edge attributes.

C. Graph Contrastive Learning

The objective of graph contrastive learning is to train an encoder $f(\cdot)$ in a self-supervised manner. The learned encoder generates node representations $\boldsymbol{H} = f(\mathbf{X}, \mathbf{A})$, where $\mathbf{H} \in \mathbb{R}^{N \times k}$ captures both topological relationships and dense semantic patterns. These representations are decoupled from specific topological biases and can be generalized to diverse downstream tasks.

He et al. [8] found that mainstream GCL frameworks [9]–[17] all inherently perform representation scattering, which plays a crucial role in their success. Here, we follow the definition of Representation Scattering in their work:

Definition 1 (Representation Scattering): In a D-dimensional embedding space $\mathbb{R}^D$ with N node embeddings represented as $\mathbf{X} \in \mathbb{R}^{N \times D}$, let $\mathbb{S}^k$ ($1 \leq k \leq D$) denotes a subspace of $\mathbb{R}^D$ and $\boldsymbol{c}$ denotes a scatter center. Representation scattering enforces two constraints: (i) center-away constraint, where node embeddings are maximally separated from $\boldsymbol{c}$; (ii) uniformity constraint, where node embeddings are uniformly distributed across $\mathbb{S}^k$.

According to Def. 1, representation scattering requires defining a scatter center $\boldsymbol{c}$ within the subspace $\mathbb{S}^k$ and enforcing both center-away and uniformity constraints. Such a mechanism constructs a common space for nodes from different circuit graphs, and also normalizes those representations.

D. Imbalanced Classification and Regression

In machine learning, label imbalance poses significant challenges for deep recognition models, motivating numerous techniques to address data imbalance [18]–[27]. Most prior works focus on imbalanced classification (also termed long-tailed recognition [28]), with solutions broadly categorized into: (i) data-based methods, such as oversampling minority classes or undersampling majority classes [29]; (ii) model-based methods, including loss reweighting or adjusted objective functions to mitigate class imbalance [18], [30].

In contrast, many EDA tasks involve regression with continuous, unbounded target values, where label imbalance is inherently more challenging. For example, Shen et al. [2] reported that the distribution of ground parasitic capacitance spans from 0.01 fF to 100 pF, where over 10^6 samples fall into the second bin [10^{-1}, 1]. While imbalanced classification is well-studied, imbalanced regression remains underexplored. Existing approaches often adapt the synthetic minority over-sampling technique (SMOTE) to regression scenarios [31], [32] or employ loss reweighting strategies [25], [33]. Reweighting assigns higher loss weights to rare samples and lower weights to frequent ones. However, recent studies [21], [26], [27] demonstrate limited effectiveness of reweighting in both classification and regression tasks.

Consider input $\boldsymbol{x} \in \mathbf{X}$ and label $\boldsymbol{y} \in \mathbf{Y} = \mathbb{R}^d$. We focus on univariate regression ($d = 1$), where training and test data originate from distinct AMS designs. This implies that the training set follows a skewed joint distribution $p_{\text{train}}(\boldsymbol{x}, \boldsymbol{y})$, while the test set adheres to a near-uniform or lightly skewed distribution $p_{\text{bal}}(\boldsymbol{x}, \boldsymbol{y})$ [21], [33]. Crucially, the label-conditional distribution $p(\boldsymbol{x}|\boldsymbol{y})$ is assumed invariant between training and testing. Under the assumption of a balanced test set, the goal of imbalanced regression shifts from estimating $p_{\text{train}}(\boldsymbol{y}|\boldsymbol{x})$ to learning $p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x})$, ensuring generalizability to unseen circuits. This equivalence aligns with the theoretical insight that balanced metrics on arbitrary test sets mirror overall metrics on

hypothetical balanced sets [34].

The mean squared error (MSE) loss is the most widely used objective function in regression tasks. For a predicted value y_{pred} and target y, the MSE loss is defined as:

$$\text{MSE}(y, y_{\text{pred}}) = \left\| y - y_{\text{pred}} \right\|_2^2, \tag{1}$$

where $\|\cdot\|_2$ denotes the ℓ_2-norm. From a probabilistic perspective, the prediction y_{pred} can be interpreted as the mean of a Gaussian distribution modeling the prediction noise [35]:

$$p(y|x; \theta) = \mathcal{N}(y; y_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}), \tag{2}$$

where θ represents the regressor's parameters, and $\sigma_{\text{noise}}^2 \mathbf{I}$ is the covariance matrix of the independent and identically distributed (i.i.d.) error term $\epsilon \sim \mathcal{N}(0, \sigma_{\text{noise}}^2 \mathbf{I})$. The MSE loss is equivalent to the Negative Log-Likelihood (NLL) of this Gaussian distribution [36], implying that MSE-trained regressors inherently model $p_{\text{train}}(y|x)$. In order to enhance performance on testsets, we aim to estimate $p_{\text{bal}}(y|x)$ instead of $p_{\text{train}}(y|x)$. Bayes' theorem is applied:

$$\frac{p_{\text{train}}(y|x)}{p_{\text{bal}}(y|x)} \propto \frac{p(x|y) \cdot p_{\text{train}}(y)}{p(x|y) \cdot p_{\text{bal}}(y)} = \frac{p_{\text{train}}(y)}{p_{\text{bal}}(y)}. \tag{3}$$

This proportionality reveals that the discrepancy between $p_{\text{train}}(y|x)$ and $p_{\text{bal}}(y|x)$ depends on the ratio of their label distributions. Since $p_{\text{train}}(y)$ is lower for rare labels, MSE-trained regressors systematically underestimate underrepresented targets in the training set.
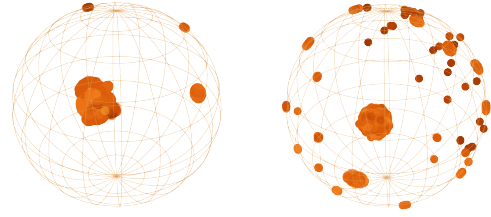
III. THE PROPOSED CIRCUITGCL FRAMEWORK

Overall workflow of CircuitGCL is depicted by Fig. 2. The proposed method contains four steps: (i) AMS netlist conversion, which is the same as work [3]; (ii) subgraph sampling; (iii) representation scattering in GCL; and (iv) label rebalancing through balanced MSE and balanced softmax cross-entropy (BSCE). In this section, we try to tackle the aforementioned scarcity and diversity of AMS circuits by introducing contrastive learning in the parasitic estimation field.

A. Representation Scattering Mechanism in GCL

To directly predict parasitic parameters from circuit topology, the absence of layout and detailed circuit information severely constrains the feature space. We employ contrastive learning to derive meaningful initial feature representations, while its self-supervised nature enables effective cross-domain transferability. Since current mainstream graph contrastive learning methods inherently employ representation scattering mechanisms (RSM), we adopt Scattering Graph Representation Learning (SGRL) [8] as our GCL foundation, which embeds node representations within a hypersphere while dispersing them from a central mean point (as depicted in Fig. 2), providing a bias-free method that preserves circuit structure and attributes.

RSM operates by defining a subspace $\mathbb{S}^k$ and a scatter center c. To project representations from the original space

(a) GNN without GCL.   (b) RSM of CircuitGCL

Fig. 3: t-SNE visualizations of node embeddings: comparisons between models with and without the GCL framework. Darker indicates larger parasitic capacitance.

$\mathbb{R}^D$ into $\mathbb{S}^k$, a transformation function $\text{Trans}(\cdot)$ applies ℓ_2-normalization to each row vector h_i in the target representation matrix $\mathbf{H}_{\text{target}}$:

$$\widetilde{h}_i = \frac{h_i}{\max\left(\|h_i\|_2, \varepsilon\right)}, \quad \mathbb{S}^k = \left\{ \widetilde{h}_i : \left\|\widetilde{h}_i\right\|_2 = 1 \right\}, \tag{4}$$

where h_i is the representation of node $v_i \in \mathcal{V}$ generated by the target encoder, $\left\|\widetilde{h}_i\right\|_2 = \left(\sum_{j=1}^k \widetilde{h}_{ij}^2\right)^{1/2}$ denotes the ℓ_2-norm, and ε is a small constant (e.g., 10^{-8}) to prevent division by zero. As per Eq. (4), all node representations are constrained to the hypersphere $\mathbb{S}^k$, ensuring stable training by preventing uncontrolled scattering in the embedding space.

Next, we define the scattered center c and introduce a representation scattering loss $\mathcal{L}_{\text{scattering}}$ to push node representations away from c in $\mathbb{S}^k$:

$$\mathcal{L}_{\text{scattering}} = -\frac{1}{N} \sum_{i=1}^N \left\|\widetilde{h}_i - c\right\|_2^2, \quad c = \frac{1}{N} \sum_{i=1}^N \widetilde{h}_i, \tag{5}$$

where c represents the mean of all normalized node embeddings. By minimizing $\mathcal{L}_{\text{scattering}}$, RSM enforces global uniformity of representations across the dataset without enforcing strict local uniformity.

Fig. 3 visualizes the t-SNE embeddings of node representations learned by different contrastive learning frameworks. Fig. 3a depicts embeddings from a GNN trained solely on downstream tasks without pre-training. These embeddings lack structural and topological awareness, resulting in poor expressiveness that severely limits the transferability of conventional GNNs. In contrast, CircuitGCL's RSM enforces uniform distribution across the subspace (Fig. 3b), yielding highly expressive embeddings. Notably, RSM naturally clusters nodes with similar labels (lighter hues denote smaller parasitic capacitance values) while pushing nodes with dissimilar labels apart. This mechanism is particularly advantageous for circuit graph representation learning. While semantically distinct nodes (e.g., analog vs. digital components) are scattered across the hypersphere to maximize separation, functionally similar nodes (e.g., repeated inverter cells) naturally cluster in local regions. Consequently, RSM unifies embedding subspaces across diverse AMS designs, enabling robust transferability between circuits with varying topologies and label distributions.

TABLE I: Resource Usage of DSPD Calculation and GCL Training.

Dataset	SSRAM			ULTRA8T			SANDWICH-RAM		
Resource	Time	Mem.	Disk	Time	Mem.	Disk	Time	Mem.	Disk
DSPD	20.4m	0.2GB	68.6MB	907.9m	3.1GB	2.6GB	1115m	9.6GB	2.6GB
GCL	4.4m	3.8GB	5.4MB	109.7m	4.6GB	5.4MB	94.7m	3.0GB	5.4MB

Note: DSPD uses CPU memory, and GCL's pre-training uses GPU video memory. Unit 'm' stands for minute.

B. Online Encoder

After generating the scattered representations $\mathbf{H}_{\text{target}} = f_\phi(\mathbf{A}, \mathbf{X})$ using the target encoder, the online encoder $f_\theta(\cdot)$ produces intermediate representations $\mathbf{H}_{\text{online}}$. These are passed through a predictor $q_\theta(\cdot)$ to obtain predicted representations $\boldsymbol{z}_{\text{online}} = q_\theta(\mathbf{H}_{\text{online}})$. The objective is to align $\boldsymbol{z}_{\text{online}}$ with $\mathbf{H}_{\text{target}}$, enhancing the model's ability to capture semantically meaningful circuit patterns. The alignment loss $\mathcal{L}_{\text{alignment}}$ is defined as:

$$\mathcal{L}_{\text{alignment}} = -\frac{1}{N} \sum_{i=1}^{N} \frac{\boldsymbol{z}_i^T \boldsymbol{h}_i}{\|\boldsymbol{z}_i\|_2 \|\boldsymbol{h}_i\|_2}, \qquad (6)$$

where $\boldsymbol{z}_{\text{online}}$ and $\mathbf{H}_{\text{target}}$ denote predicted and target embeddings, respectively. During training, only the online encoder's parameters $\boldsymbol{\theta}$ are updated via gradient descent, while the target encoder's parameters ϕ remain fixed.

Unlike direct alignment of constrained and scattered representations, the predictor $q_\theta(\cdot)$ acts as an adaptive buffer, enabling the online encoder to learn stable, topology-aware embeddings.

To ensure the target encoder incorporates topological semantics into the scattering process, rather than optimizing solely for uniformity, we update ϕ using an exponential moving average (EMA) of $\boldsymbol{\theta}$ after each epoch:

$$\phi \leftarrow \tau\phi + (1-\tau)\boldsymbol{\theta}, \qquad (7)$$

where $\tau \in [0,1]$ is a decay rate (typically $\tau \geq 0.99$). This gradual update prevents adversarial collapse between the encoders and stabilizes training.

C. Comparison with DSPD

In CircuitGPS [3], double-anchor shortest path distance (DSPD) serves as a critical positional encoding (PE) for initial node embeddings during pre-training and fine-tuning. DSPD computes the relative shortest path distances between nodes in subgraphs using the resources of CPUs, but its computational and storage costs scale quadratically with subgraph size. As shown in Tab. I, DSPD becomes prohibitively expensive for large circuits, restricting 1-hop subgraph sampling in Circuit-GPS. By contrast, CircuitGCL pre-trains the encoders to generate the initial embeddings to replace the DSPD calculation with high parallelism and good model scalability.

IV. LABEL REBALANCING

In parasitic estimation, label imbalance refers to the distributions of parasitic capacitance that are heavily skewed with long-tailed patterns in AMS circuits, as shown in Fig. 4. However, our trained GNNs are used to other unseen designs. Such an imbalance degrades the generalizability and fairness of GNNs, and it poses a critical challenge that demands dedicated efforts to enhance data-driven transferability. Balanced MSE [21] and balanced softmax cross-entropy (BSCE) [18] address the distribution mismatch between training and test circuits by serving as statistically principled loss functions. Assuming the test set labels follow a balanced distribution with conditional probability $p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x})$, we derive $p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x})$ from the skewed training distribution $p_{\text{train}}(\boldsymbol{y}|\boldsymbol{x})$ using the training label distribution $p_{\text{train}}(\boldsymbol{y})$. Expanding Eq. (3), the relationship between the training and balanced distributions is expressed as:

$$p_{\text{train}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x}) \cdot p_{\text{train}}(\boldsymbol{y})}{\int_{\mathbf{Y}} p_{\text{bal}}(\boldsymbol{y}'|\boldsymbol{x}) \cdot p_{\text{train}}(\boldsymbol{y}') \, d\boldsymbol{y}'}. \qquad (8)$$

To estimate $p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x})$, we minimize the negative log-likelihood (NLL) of $p_{\text{train}}(\boldsymbol{y}|\boldsymbol{x})$ (see [21] for proof). During training, we: (i) compute $p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$ using the regressor, (ii) convert it to $p_{\text{train}}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$ via Eq. (8), (iii) update parameters $\boldsymbol{\theta}$ by minimizing the NLL loss. During inference, the regressor directly estimates $p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x})$ without conversion:

$$p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{y}; \boldsymbol{y}_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}\right), \qquad (9)$$

where $\boldsymbol{y}_{\text{pred}}$ is the model's prediction.

The balanced MSE and softmax CE losses are both derived from Eq. (8) and Eq. (9), with the former amending the MSE loss and the latter scaling logits to reflect test-time class balance. They share a common theoretical foundation in distribution alignment but differ in their handling of continuous vs. discrete labels.

A. Balanced MSE for Regression

Definition 2 (Balanced MSE): Given a regressor's prediction $\boldsymbol{y}_{\text{pred}}$ and the training label distribution prior $p_{\text{train}}(\boldsymbol{y})$, the balanced MSE (BMSE) loss is defined as:

$$\begin{aligned} \mathcal{L}_{\text{BMSE}} &= -\log p_{\text{train}}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}) \\ &= -\log \frac{p_{\text{bal}}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}) \cdot p_{\text{train}}(\boldsymbol{y})}{\int_{\mathbf{Y}} p_{\text{bal}}(\boldsymbol{y}'|\boldsymbol{x}; \boldsymbol{\theta}) \cdot p_{\text{train}}(\boldsymbol{y}') \, d\boldsymbol{y}'} \\ &\cong -\log \mathcal{N}(\boldsymbol{y}; \boldsymbol{y}_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}) \\ &\quad + \log \int_{\mathbf{Y}} \mathcal{N}(\boldsymbol{y}'; \boldsymbol{y}_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}) \cdot p_{\text{train}}(\boldsymbol{y}') \, d\boldsymbol{y}', \end{aligned} \qquad (10)$$

where $\cong$ omits the constant term $-\log p_{\text{train}}(\boldsymbol{y})$.

The balanced MSE loss comprises two components: a standard MSE loss (first term), derived from the negative log-likelihood of the Gaussian prediction; a balancing term (second term), which corrects for label distribution skew by integrating over the entire label space $\mathbf{Y}$. As demonstrated by Ren et al. [21], the standard MSE loss emerges as a special case of balanced MSE when $p_{\text{train}}(\boldsymbol{y})$ is uniform.

To operationalize Eq. (10), the integral term must be evaluated in closed form. A key challenge lies in modeling $p_{\text{train}}(\boldsymbol{y})$ to ensure tractability. We propose two implementations:
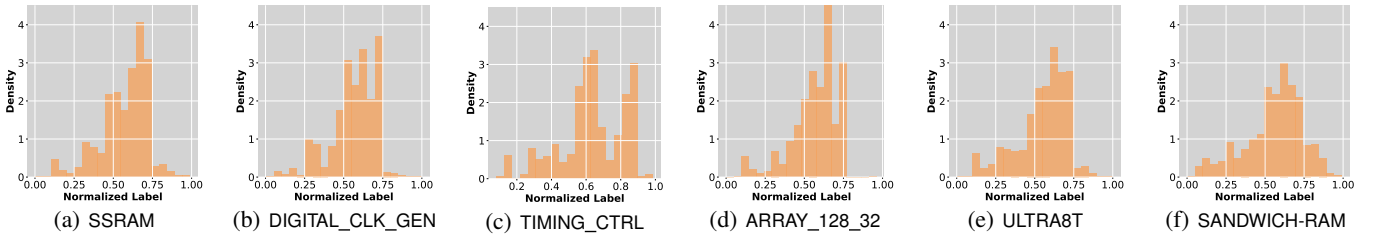
Fig. 4: Normalized label distributions of all AMS circuit datasets.

GMM-based Analytical Integration (GAI). Assume $p_{\text{train}}(\boldsymbol{y})$ follows a Gaussian Mixture Model (GMM), which enables analytical integration due to the closure property of Gaussians under multiplication. Let:

$$p_{\text{train}}(\boldsymbol{y}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \qquad (11)$$

where K is the number of components, and ϕ_i, $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ denote the weight, mean, and covariance of the i-th Gaussian. Substituting Eq. (11) into Eq. (10) yields:

$$\begin{aligned}\mathcal{L}_{\text{GAI}} = &-\log \mathcal{N}(\boldsymbol{y}; \boldsymbol{y}_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}) \\ &+ \log \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{y}_{\text{pred}}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i + \sigma_{\text{noise}}^2 \mathbf{I}).\end{aligned} \qquad (12)$$

The second term approximates the integral via GMM components, leveraging the conjugacy of Gaussians.

Batch-based Monte Carlo (BMC). With batch size N, BMC estimates the integral empirically using labels in a training batch $\mathcal{B}_{\boldsymbol{y}} = \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \dots, \boldsymbol{y}^{(N)}\}$, requiring no prior knowledge of $p_{\text{train}}(\boldsymbol{y})$. The loss becomes:

$$\begin{aligned}\mathcal{L}_{\text{BMC}} = &-\log \mathcal{N}(\boldsymbol{y}; \boldsymbol{y}_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}) \\ &+ \log \sum_{i=1}^{N} \mathcal{N}(\boldsymbol{y}^{(i)}; \boldsymbol{y}_{\text{pred}}, \sigma_{\text{noise}}^2 \mathbf{I}).\end{aligned} \qquad (13)$$

By rewriting Eq. (13), we can see its connection to temperature-scaled softmax:

$$\mathcal{L} = -\log \frac{\exp\left(-\|\boldsymbol{y}_{\text{pred}} - \boldsymbol{y}\|_2^2 / \tau\right)}{\sum_{\boldsymbol{y}' \in \mathcal{B}_{\boldsymbol{y}}} \exp\left(-\|\boldsymbol{y}_{\text{pred}} - \boldsymbol{y}'\|_2^2 / \tau\right)}, \qquad (14)$$

where $\tau = 2\sigma_{\text{noise}}^2$ controls the sharpness of the weighting.

B. Balanced Softmax for Classification

In imbalanced classification, where the label space $\mathcal{Y}$ is discrete and one-dimensional, the relationship between training and balanced distributions (Eq. (8)) remains valid but replaces integration with summation:

$$p_{\text{train}}(y|\boldsymbol{x}) = \frac{p_{\text{bal}}(y|\boldsymbol{x}) \cdot p_{\text{train}}(y)}{\sum_{y' \in \mathcal{Y}} p_{\text{bal}}(y'|\boldsymbol{x}) \cdot p_{\text{train}}(y')}, \qquad (15)$$

where $y \in \mathcal{Y}$ denotes discrete class labels, and $p_{\text{train}}(y)$ is the empirical frequency of class y in the training set. Moreover,

TABLE II: AMS Circuit Dataset Statistics. One Design Case Is Sufficient to Train CircuitGCL.

Split	Dataset	N	N_E	#Links
Train.&Val.	SSRAM	87K	134K	131K
Test	DIGITAL_CLK_GEN	17K	36K	4K
	TIMING_CTRL	18K	44K	5K
	ARRAY_128_32	144K	352K	110K
	ULTRA8T	3.5M	13.4M	166K
	SANDWICH-RAM	4.3M	13.3M	154K

the balanced distribution $p_{\text{bal}}(y|\boldsymbol{x}; \boldsymbol{\theta})$ is typically modeled via softmax normalization for classification:

$$p_{\text{bal}}(y|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp(\eta[y])}{\sum_{y' \in \mathcal{Y}} \exp(\eta[y'])}, \qquad (16)$$

where $\eta[y] \in \mathbb{R}$ is the logit (unnormalized score) for class y. Substituting Eq. (16) into Eq. (15) yields:

$$p_{\text{train}}(y|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp(\eta[y]) \cdot p_{\text{train}}(y)}{\sum_{y' \in \mathcal{Y}} \exp(\eta[y']) \cdot p_{\text{train}}(y')}. \qquad (17)$$

This aligns with logit adjustment techniques in imbalanced classification [18]–[20], which offset logits by class frequency logarithms.

Definition 3 (Balanced Softmax CE): Given a classifier's raw logits $\eta[y]$ and the training label distribution prior $p_{\text{train}}(y)$, the balanced Softmax cross-entropy loss is defined as:

$$\begin{aligned}\mathcal{L}_{\text{BSCE}} &= -\log \frac{\exp(\eta[y]) \cdot p_{\text{train}}(y)}{\sum_{y' \in \mathcal{Y}} \exp(\eta[y']) \cdot p_{\text{train}}(y')} \\ &= -(\eta[y] + \log p_{\text{train}}(y)) \\ &\quad + \log \left(\sum_{y' \in \mathcal{Y}} \exp(\eta[y'] + \log p_{\text{train}}(y')) \right).\end{aligned} \qquad (18)$$

CircuitGCL provides a unified view of imbalanced regression and classification. By deriving BMSE and BSCE from the same distribution alignment principle, we validate these loss functions as dual instantiations of a core theoretical insight.

V. EXPERIMENTS

Our implementation leverages PyG [37] for graph processing. All experiments were conducted on a shared computing cluster equipped with 40 Intel Xeon Silver 4314 CPUs (2.4 GHz), 128GB RAM, and four NVIDIA RTX 4090 GPUs (24GB VRAM). Each training run utilized 4-6 CPU cores,

TABLE III: Error Comparison of CircuitGCL and Prior Methods on Edge Regression.

Testset	TIMING_CTRL				ARRAY_128_32				ULTRA8T				SANDWICH-RAM			
Metric	Loss	MAE↓	MSE↓	R^2 ↑	Loss	MAE↓	MSE↓	R^2 ↑	Loss	MAE↓	MSE↓	R^2 ↑	Loss	MAE↓	MSE↓	R^2 ↑
ParaGraph	0.0153	0.0914	0.0153	0.5250	0.0115	0.0788	0.0115	0.4252	0.0175	0.0937	0.0175	0.3200	0.0223	0.1087	0.0223	0.3389
CircuitGPS	0.0105	0.0742	0.0105	0.6911	0.0108	0.0701	0.0108	0.4576	0.0158	0.0818	0.0158	0.3845	0.0225	0.1039	0.0225	0.3326
DLPL-Cap	0.0093	0.0701	0.0093	0.7056	0.0123	0.0806	0.0123	0.3853	0.0160	0.0813	0.0160	0.3704	0.0214	0.1012	0.0214	0.3622
CircuitGCL (MSE)	0.0118	0.0868	0.0118	0.6521	0.0093	0.0671	0.0093	0.5350	0.0144	0.0794	**0.0144**	**0.4398**	0.0193	0.0992	0.0193	0.4280
CircuitGCL (BMC)	71.626	0.0628	**0.0088**	**0.7407**	74.313	**0.0650**	0.0092	0.5418	117.73	0.0771	0.0145	0.4350	152.34	0.0938	0.0188	0.4422
CircuitGCL (GAI)	0.0091	**0.0610**	0.0091	0.7300	0.0089	0.0667	**0.0089**	**0.5556**	0.0145	**0.0762**	0.0145	0.4358	0.0187	**0.0935**	**0.0187**	**0.4445**
Max. Impr.	-	33.26%	42.48%	41.08%	-	19.35%	27.64%	44.20%	-	18.68%	17.71%	37.44%	-	0.1398%	16.89%	33.64%

TABLE IV: Accuracy Comparison of CircuitGCL and Prior Methods on Node Classification.

Test Set	DIGITAL_CLK_GEN					ARRAY_128_32					ULTRA8T					SANDWICH-RAM				
Metric	Loss	Acc.↑	Precision↑	Recall↑	F1↑	Loss	Acc.↑	Precision↑	Recall↑	F1↑	Loss	Acc.↑	Precision↑	Recall↑	F1↑	Loss	Acc.↑	Precision↑	Recall↑	F1↑
ParaGraph	0.1973	0.2359	0.1249	0.3128	0.1771	0.0730	0.6024	0.2923	0.3239	0.2945	0.3228	0.6252	0.3058	0.3129	0.2973	0.8511	0.3651	0.2485	0.2675	0.2199
CircuitGPS	3.1762	0.2082	0.2449	0.3119	0.2259	1.5075	0.2200	0.4000	0.1628	0.2299	1.0922	0.3925	0.4541	0.2506	0.2892	2.307	0.4185	0.4494	0.3573	0.3002
DLPL-Cap	2.0780	**0.7130**	**0.6918**	**0.6755**	**0.6167**	0.0142	0.8888	0.6943	0.6481	0.6578	0.3289	0.8683	0.6099	0.6109	0.5984	2.6968	0.5701	0.4783	0.4140	0.4248
CircuitGCL (CE)	20.936	0.5400	0.3611	0.4121	0.3328	0.5537	0.6792	0.5037	0.5304	0.5174	0.9860	0.5787	0.6194	0.4316	0.4592	2.1311	0.4441	0.4990	0.3044	0.3607
CircuitGCL (Focal)	3.9955	0.5380	0.3503	0.4021	0.3145	0.0226	0.8994	0.5756	0.5402	0.5533	0.4021	0.8163	**0.6381**	0.6040	0.6157	1.2232	0.5809	0.5227	0.4210	0.4580
CircuitGCL (BSCE)	18.898	0.6760	0.6855	0.5981	0.5309	0.4989	**0.9622**	0.7195	**0.7211**	**0.7185**	0.9616	**0.9146**	0.6376	**0.6391**	**0.6382**	1.7694	**0.7231**	**0.5826**	**0.5392**	**0.5514**
Max. Impr.	-	2.2×	4.5×	0.9×	0.9×	-	3.4×	0.8×	3.4×	2.1×	-	1.3×	1.1×	1.6×	1.2×	-	1.0×	1.3×	1.0×	1.5×

one GPU, and 128GB of memory. In data preparation, full schematic netlists were first parsed to extract graph structures and node features (circuit statistics), which is the same as work [3]; then, post-layout netlists (Standard Parasitic Format/SPF files) were also processed to collect ground-truth coupling capacitance labels. The subgraph sampling is implemented by "LinkNeigbhorLoader" in PyG. We compare CircuitGCL against three state-of-the-art approaches for parasitic capacitance prediction: (i) ParaGraph [1] is a MPNN-based ensemble model; (ii) DLPL-Cap [2] is a multi-expert GNN regressor; (iii) CircuitGPS [3] uses few-shot learning with positional encoding.

A. AMS Datasets

Table II summarizes the AMS circuit datasets used in our experiments, all implemented in TSMC 28nm CMOS technology. To demonstrate data transferability, we train CircuitGCL on a single mid-sized design and evaluate it on four unseen test cases. The graph contrastive learning (GCL) component is also pretrained on the same single design to learn expressive node representations.

SSRAM [38] is a medium-scale energy-efficient design combining standard digital cells and SRAM arrays. SANDWICH-RAM [39] features a balanced architecture with computational digital circuits and storage SRAM arrays in alternating layers. ULTRA8T SRAM [40] is the largest design with multi-voltage domains and extensive analog modules. As for test sets, DIGITAL_CLK_GEN is a digital/SRAM hybrid for internal SRAM clock generation. TIMING_CONTROL is a digital control signal generator for SRAM operations. ARRAY_128_32 is a standalone

128×32 SRAM array. All test sets are strictly excluded from training/validation data, ensuring zero-shot evaluation.

Fig. 4 illustrates the significant distribution shifts between datasets, highlighting the challenge of developing universal transfer learning methods for such AMS circuits. In the regression task, we keep the coupling capacitor with capacitance $y_i \in [1^{-21}, 1^{-15}]$. In the classification task, we divide the ground capacitors connecting to net nodes into 5 categories in terms of the magnitude of their capacitance values.

B. Edge Regression - Coupling Capacitance Estimation

In this task, we configure encoders in CircuitGCL with 4-layer ClusterGCN [41], 256 hidden dimensions, Tanh activation, 0.3 dropout, and 1^{-6} learning rate. The downstream GNN adopts a 5-layer GraphSAGE [42], with 144 hidden dimensions, PReLU activation, 0.3 dropout, and 1^{-4} learning rate. The task-specific heads are 2-layer MLPs with the same number of hidden dimensions as the CL encoders and the downstream GNN, respectively. The σ_{noise} in Eq. (12) and Eq. (14) is set to be 0.001.

Table III compares CircuitGCL against prior methods on parasitic capacitance regression. GAI variant achieves the best overall performance across all test sets. On large-scale datasets, GAI significantly outperforms existing methods—for example, 23% improvement over DLPL-Cap on SANDWICH-RAM. The technique shows consistent advantages on both small (TIMING_CTRL) and large (ULTRA8T, SANDWICH-RAM) designs, while prior approaches struggle with scalability. Although BMC exhibits high absolute loss values due to batch normalization, it maintains competitive relative metrics.
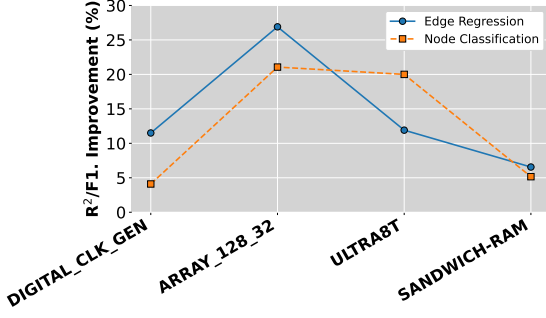
Fig. 5: R^2 and F1 improvements of applying RSM to regression and classification tasks, respectively.
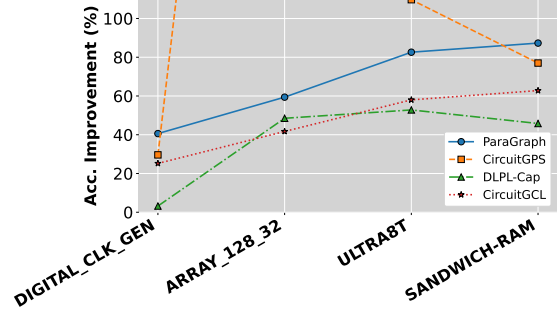

Fig. 6: Accuracy improvement of applying BSCE to other baselines. The accuracy gains are normalized to CE.
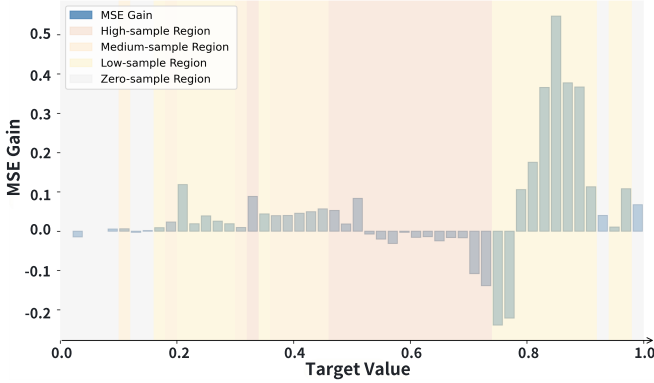

Fig. 7: MSE improvement through label rebalancing technique. Balanced MSE significantly enhances model performance in data-scarce regions (pale yellow background).

These results validate CircuitGCL's transferable representation learning capability for parasitic prediction.

C. Node Classification - Ground Capacitance Classification

In node classification, we configure encoders as in edge regression (Sect. V-B) and use 4-layer GraphSAGE with focal loss for fair comparison. We categorize net nodes into 5 classes based on normalized ground capacitance (Fig. 4), excluding the most frequent category (with label 2) when calculating metrics to emphasize model differences.

Table IV compares CircuitGCL against baselines on node classification. CircuitGCL with BSCE achieves state-of-the-art performance, particularly on large-scale designs with significant improvements over DLPL-Cap (e.g., 3.4× accuracy improvement on ARRAY_128_32). While DLPL-Cap excels on DIGITAL_CLK_GEN, it fails to generalize to larger designs, highlighting its reliance on task-specific tuning. ParaGraph and CircuitGPS show poor scalability with limited F1 scores on SANDWICH-RAM. CircuitGCL's balanced softmax CE effectively addresses label imbalance, achieving robust performance across diverse circuit topologies and demonstrating superior scalability over existing baselines.

D. Extended Study on RSM of GCL

Here, we further discuss the impact of using the RSM of contrastive learning. As depicted in Fig. 5, RSM provides the most significant R^2 improvement on ARRAY_128_32 (26.9%), suggesting it effectively handles the complex coupling patterns in SRAM arrays. The largest F1 gain occurs on ULTRA8T (20.0%), demonstrating RSM's ability to improve semantic clustering in large-scale designs. While improvements vary by dataset, RSM never degrades performance, with minimum gains of 4.1% (F1) and 6.56% (R^2). These results validate RSM as a crucial component of CircuitGCL for learning transferable representations in AMS circuits.

E. Extended Study on Label Rebalancing

To further demonstrate the effectiveness of balanced loss functions, we conduct an extended study using BSCE as the loss function for all baselines. As shown in Fig. 6, BSCE yields significant accuracy improvements across all methods, with particularly substantial gains observed in large-scale designs, highlighting BSCE's effectiveness for scalable imbalanced classification. Fig. 7 demonstrates the performance improvements of label rebalancing across data-scarce regions, where MSE gain shows it significantly enhances model performance in these challenging regions.

VI. CONCLUSION

CircuitGCL's self-supervised paradigm and distribution-aware losses address two universal EDA challenges, particularly pronounced in AMS circuit design: data scarcity (via GCL) and label imbalance (via rebalancing). Its graph-native architecture aligns naturally with circuit netlists, enabling seamless adoption in commercial tools for tasks requiring rapid design-space exploration. This positions CircuitGCL as a foundational framework for next-generation EDA tools, bridging the gap between data-driven automation and precision-critical circuit design. Future work could extend the framework to broader AMS circuit varieties, adapt CircuitGCL to parasitic resistance estimation, or integrate it into an RC-aware placement & routing tool.

REFERENCES

[1] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proc. DAC*, 2020, pp. 1–6.

[2] S. Shen, D. Yang, Y. Xie, C. Pei, B. Yu, and W. Yu, "Deep-learning-based pre-layout parasitic capacitance prediction on sram designs," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 440–445.

[3] S. Shen, Y. Zhang, H. Rodriguez, and W. Yu, "Few-shot learning on ams circuits and its application to parasitic capacitance prediction," in *Proceedings of the 64th Annual Design Automation Conference*, 2025.

[4] W. Yu, S. Shen, D. Yang, H. Li, J. Huang, and C. Pei, "Deep learning inspired capacitance extraction techniques," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 106–112.

[5] W. Yu and X. Wang, *Advanced field-solver techniques for RC extraction of integrated circuits*. Springer, 2014.

[6] W. Yu, C. Hu, and W. Zhang, "Variational capacitance extraction of on-chip interconnects based on continuous surface model," in *Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 758–763.

[7] W. Yu, Z. Wang, and X. Hong, "Preconditioned multi-zone boundary element analysis for fast 3D electric simulation," *Engineering Analysis with Boundary Elements*, vol. 28, no. 9, pp. 1035–1044, 2004.

[8] D. He, L. Shan, J. Zhao, H. Zhang, Z. Wang, and W. Zhang, "Exploitation of a latent mechanism in graph contrastive learning: Representation scattering," *Advances in Neural Information Processing Systems*, vol. 37, pp. 115 351–115 376, 2024.

[9] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[10] ——, "Graph contrastive learning with adaptive augmentation," in *Proceedings of the web conference 2021*, 2021, pp. 2069–2080.

[11] J. Xia, L. Wu, G. Wang, J. Chen, and S. Z. Li, "Progcl: Rethinking hard negative mining in graph contrastive learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 24 332–24 346.

[12] Y. Zheng, S. Pan, V. Lee, Y. Zheng, and P. S. Yu, "Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination," *Advances in Neural Information Processing Systems*, vol. 35, pp. 10 809–10 820, 2022.

[13] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax." *ICLR (poster)*, vol. 2, no. 3, p. 4, 2019.

[14] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International conference on machine learning*. PMLR, 2020, pp. 4116–4126.

[15] S. Thakoor, C. Tallec, M. G. Azar, R. Munos, P. Veličković, and M. Valko, "Bootstrapped representation learning on graphs," in *ICLR 2021 workshop on geometrical and topological representation learning*, 2021.

[16] N. Lee, J. Lee, and C. Park, "Augmentation-free self-supervised learning on graphs," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 7, 2022, pp. 7372–7380.

[17] W. Sun, J. Li, L. Chen, B. Wu, Y. Bian, and Z. Zheng, "Rethinking and simplifying bootstrapped graph latents," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024, pp. 665–673.

[18] J. Ren, C. Yu, X. Ma, H. Zhao, S. Yi *et al.*, "Balanced meta-softmax for long-tailed visual recognition," *Advances in neural information processing systems*, vol. 33, pp. 4175–4186, 2020.

[19] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, "Long-tail learning via logit adjustment," *arXiv preprint arXiv:2007.07314*, 2020.

[20] Y. Hong, S. Han, K. Choi, S. Seo, B. Kim, and B. Chang, "Disentangling label distribution for long-tailed visual recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 6626–6636.

[21] J. Ren, M. Zhang, C. Yu, and Z. Liu, "Balanced mse for imbalanced visual regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7926–7935.

[22] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, "Decoupling representation and classifier for long-tailed recognition," *arXiv preprint arXiv:1910.09217*, 2019.

[23] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, "Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9719–9728.

[24] X. Wang, L. Lian, Z. Miao, Z. Liu, and S. X. Yu, "Long-tailed recognition by routing diverse distribution-aware experts," *arXiv preprint arXiv:2010.01809*, 2020.

[25] M. Steininger, K. Kobs, P. Davidson, A. Krause, and A. Hotho, "Density-based weighting for imbalanced regression," *Machine Learning*, vol. 110, pp. 2187–2211, 2021.

[26] J. Byrd and Z. Lipton, "What is the effect of importance weighting in deep learning?" in *International conference on machine learning*. PMLR, 2019, pp. 872–881.

[27] D. Xu, Y. Ye, and C. Ruan, "Understanding the role of importance weighting for deep learning," *arXiv preprint arXiv:2103.15209*, 2021.

[28] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, "Large-scale long-tailed recognition in an open world," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2537–2546.

[29] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[30] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," *Advances in neural information processing systems*, vol. 32, 2019.

[31] L. Torgo, R. P. Ribeiro, B. Pfahringer, and P. Branco, "Smote for regression," in *Portuguese conference on artificial intelligence*. Springer, 2013, pp. 378–389.

[32] P. Branco, L. Torgo, and R. P. Ribeiro, "Smogn: a pre-processing approach for imbalanced regression," in *First international workshop on learning with imbalanced domains: Theory and applications*. PMLR, 2017, pp. 36–50.

[33] Y. Yang, K. Zha, Y. Chen, H. Wang, and D. Katabi, "Delving into deep imbalanced regression," in *International conference on machine learning*. PMLR, 2021, pp. 11 842–11 851.

[34] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 3121–3124.

[35] P. McCullagh, *Generalized linear models*. Routledge, 2019.

[36] D. A. Nix and A. S. Weigend, "Estimating the mean and variance of the target probability distribution," in *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, vol. 1. IEEE, 1994, pp. 55–60.

[37] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[38] S. Shen, T. Shao, X. Shang, Y. Guo, M. Ling, J. Yang, and L. Shi, "TS cache: A fast cache with timing-speculation mechanism under low supply voltages," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 252–262, 2019.

[39] J. Yang, Y. Kong, Z. Wang, Y. Liu, B. Wang, S. Yin, and L. Shi, "24.4 sandwich-RAM: An energy-efficient in-memory BWN architecture with pulse-width modulation," in *Proc. Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 394–396.

[40] S. Shen, H. Xu, Y. Zhou, M. Ling, and W. Yu, "Ultra8t: A sub-threshold 8T SRAM with leakage detection," *Integration*, vol. 98, p. 102233, 2024.

[41] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 257–266.

[42] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.