GRA-LPO: Graph Convolution Based Leakage Power Optimization

Uday Mallappa[1] and Chung-Kuan Cheng[2]
[1]Electrical and Computer Engineering, [2]Computer Science and Engineering, UC San Diego, La Jolla, California, USA
{umallapp,ckcheng}@ucsd.edu

ABSTRACT

Static power consumption is a critical challenge for IC designs, particularly for mobile and IoT applications. A final post-layout step in modern design flows involves a leakage recovery step that is embedded in signoff static timing analysis tools. The goal of such recovery is to make use of the positive slack (if any) and recover the leakage power by performing cell swaps with footprint compatible variants. Though such swaps result in unaltered routing, the hard constraint is not to introduce any new timing violations. This process can require up to tens of hours of runtime, just before the tapeout, when schedule and resource constraints are tightest. The physical design teams can benefit greatly from a fast predictor of the leakage recovery step: if the eventual recovery will be too small, the entire step can be skipped, and the resources can be allocated elsewhere. If we represent the circuit netlist as a graph with cells as vertices and nets connecting these cells as edges, the leakage recovery step is an optimization step, on this graph. If we can learn these optimizations over several graphs with various logic-cone structures, we can generalize the learning to unseen graphs. Using graph convolution neural networks, we develop a learning-based model, that predicts per-cell recoverable slack, and translate these slack values to equivalent power savings. For designs up to 1.6M instances, our inference step takes less than 12 seconds on a Tesla P100 GPU, and an additional feature extraction, post-processing steps consuming 420 seconds. The model is accurate with relative error under 6.2%, for the design-specific context.

ACM Reference Format:
Uday Mallappa[1] and Chung-Kuan Cheng[2] . 2021. GRA-LPO: Graph Convolution Based Leakage Power Optimization . In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431574

1 INTRODUCTION

Multi-threshold CMOS provides many tradeoff points between device speed and leakage; these are leveraged in post-layout "swapping" optimizations that reduce leakage power without sacrificing timing-correctness. State-of-the-art commercial and academic tools use various sensitivity functions to guide iterative cell swapping meta-heuristics to minimize leakage. However, such methods are runtime-intensive since any (potential or actual) cell swap must be assessed using high-accuracy incremental static timing analysis before being committed.

Design methodology teams spend substantial time to develop flows and methodologies that are likely to achieve best-possible design PPA (power, performance and area) within schedule and engineering constraints. However, a design's true PPA quality is known only after a mandatory leakage recovery step that is executed by commercial tools, or by internally-developed scripts built around incremental STA engines. If designers could somehow achieve an accurate prediction of final design leakage power without executing the actual leakage recovery step, then a vast space of PPA optimizations could be evaluated earlier in the design process, with less schedule impact.

Moreover, today's physical implementation teams must perform leakage recovery as a signoff step, often doing this multiple times during the physical design passes leading up to tapeout. If the predicted leakage recovery is very small, then designers could choose to skip the leakage recovery step for a given netlist, and apply schedule and compute resources elsewhere. This gives rise to a need of a leakage recovery estimator, that can learn from the past experience and quickly predict the leakage recovery on unseen design implementations.

Intuitively, post-layout leakage recovery will reflect netlist attributes such as the number of timing paths with positive slack, the magnitude of positive slack on cell instances, etc. Design engineers will attempt to estimate (quickly) the leakage reduction opportunity using visualizations such as the slack histogram, which depicts design timing with amounts of path slack (or, endpoint slack) on the x-axis and number of timing paths (or, endpoints) with a particular amount of slack on the y-axis. The greater the timing slack on a timing path or in an endpoint's fanin cone, the greater is the potential to recover leakage by swapping one or more cells to slower and less-leaky cells (while avoiding creation of any timing violations).

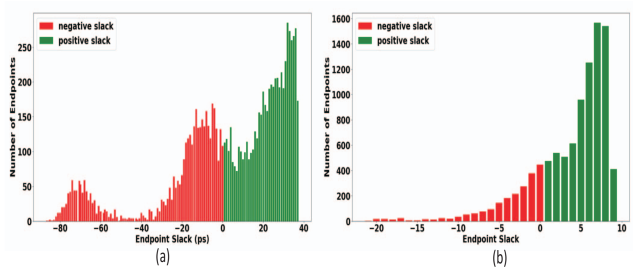

Figure 1: Slack histograms showing top 10K paths for two implementations in a 28nm design enablement of the *dec_viterbi* design [6].

Unfortunately, based on available aids such as slack histograms, it remains difficult for engineers to accurately estimate the available percentage of leakage recovery. Figure 1 shows slack histograms for top 10K timing paths of two implementations of the open-source design *dec_viterbi* [6]. The implementation in (a) leads to 40% leakage reduction achieved by a leading commercial STA tool's ECO function; the implementation in (b) leads to 0.07% leakage reduction achieved by the same tool. A key takeaway is that, it is not possible to accurately predict leakage recovery outcome of implementation (a), given the outcome of implementation (b), and vice-versa. Though it might be possible sometimes, to rank leakage recovery outcomes, by looking at the slack histograms, the above example demonstrates that it is not always possible to do so.

ASPDAC '21, January 18–21, 2021, Tokyo, Japan
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7999-1/21/01.
https://doi.org/10.1145/3394885.3431574

Inadvertently, this also gives us a clue about the complexity involved in the process of leakage recovery itself, primarily attributed to two reasons: (1) the hard constraint that the resulting netlist should not deteriorate the slack of timing paths with negative slack (2) the sharing of cells across many strongly intertwined timing paths. Even with the presence of multiple positive slack timing paths, any cell swap on these positive timing paths could result in newer violations in timing paths that share this cell. So, the timing path's recoverable slack needs to be shared by these strongly interacting cells. Therefore, if we get an estimate of recoverable slack per-cell, this gives us a very close idea about the potential power recovery (based on the available space of swaps). From our experiments, we observe that this per-cell recoverable slack in a design is influenced by various cell-level, path-level and design-level attributes such as cell types, frequency of the clock, placement utilization, etc. For example, a timing path with all high-Vth (higher threshold-voltage) cells will not admit leakage power recovery (except possibly through downsizing) even if it has a large amount of positive slack. And, attributes such as the depth and fanout of a swapped cell in a timing path (i.e., relative to the path's startpoint) determine the amount of available path-slack that a cell can make use of. So, any rational inference about the recovery outcome should carefully observe the entire netlist and its timing graph as a whole. This motivates us to think of a model that can understand and learn from the netlist and its associated timing graphs, and eventually predict the netlist's timing changes after the recovery process.

In this work, we use Graph Convolutional Networks (GCNs), to develop a learning model, that predicts recoverable slack per-cell, that is translated to the potential leakage recovery. Importantly, we do not seek to create a new gate-sizing signoff tool for leakage recovery, as that function is well-served by high-quality commercial and academic tools. Rather, we seek to provide an accurate estimate of the leakage recovery that will result if a specific "golden tool" is launched.

In the remainder of our paper, Section 2 reviews the relevant previous works, and Section 3 formulates our problem as a GCN-based prediction problem. Section 4 establishes a background on GCN notation and the intuition behind choosing GCN for this problem. We give details of our modeling flow and methodology in Section 5. Section 6 presents our experimental design, setup and results, and we conclude the paper by listing our ongoing and future research directions.

2 RELATED WORK

While no previous works use learning-based model for the problem of post-routing leakage recovery prediction, we note three relevant previous literature, on (i) gate sizing for leakage recovery, (ii) learning-based methods for leakage estimation and (iii) prior applications of Graph Convolutional Networks.

Previous works related to leakage recovery include both continuous and discrete gate sizing optimizations. The former is exemplified by the TILOS work of Fishburn and Dunlop [15], which applies continuous gate sizing to optimize transistor parameters. The latter works perform discrete sizing of standard cells that are characterized by drive strength, input pin capacitance, and other standard library parameters. Typically, combinatorial and/or metaheuristic global optimization approaches are applied, notably Lagrangian relaxation [10, 13, 16, 22, 29, 32]; dynamic programming [17, 21, 24]; slew budgeting [27]; network flow [23]; sensitivity-based optimization [18, 26, 31]; branch and bound [25]; linear programming [9, 12, 19]; parallel and randomized algorithms [33]; or simulated annealing [28]. All of the above-mentioned sizing techniques will require up to several tens of hours of runtime to perform leakage recovery for large, complex design blocks; this motivates our quest to predict the outcomes of the leakage recovery.

Learning-based techniques [14] [11] have been proposed for post-route timing optimization. The work of [2] uses gate count, gate type and state-dependent power values for leakage power prediction. Nemani and Najm [3, 4] estimate total power from the RTL netlist, even before gate-level synthesis is performed. They use entropy as a measure of the average activity to be expected in the actual implementation of a circuit, given only its Boolean functional description. The learning-based work of [1] proposes a regression model to determine change in timing slack with Vth-swap. The authors of [1] highlight two major drawbacks of the methodology, namely, training error associated with complex cells, and the lack of a complete timer graph, which causes large error. By contrast, we train a predictive model using netlist and timing graph features extracted using golden sizing tools (which are built on top of incremental STA engines).

Lastly, to work directly on optimization over timing graphs, we make use of the rich Graph Convolution Networks (GCN) algorithms [34] that have proven success in the generalization problems involving arbitrarily structured graphs such as social networks, biological protein structures, and brain structures. The basic idea is to to aggregate information of the neighbor nodes and its own information, while generating a node's representation. The non-spectral GCN works such as [35] and [36] make use of convolution over spatially close neighbors. Variants of non-spectral GCNs such as Graph Attention Networks [38] (GATs) are found to be very useful in several node classification and regression problems such as citation networks and protein classification. In GCNs, the aggregation over neighboring nodes is normalized by using the degree of the node as a metric unlike GraphSage [37], where the aggregation over neighboring nodes is not normalized. In GATs [38], the aggregation over node features makes use of the self-attention mechanism (higher attention factor for nodes with similar features). In the context of timing propagation, it is rational to aggregate over neighbors that are strongly correlated (critical paths) to the node of interest.

3 FORMULATION:

Given: (1) *X_train* = a timing-optimized pre-leakage recovery netlist represented as a directed acyclic graph, with nets as the edges, and the nodes as the components (cells and ports) in the netlist. Each node is represented by a vector of its electrical attributes, along with its initial cell type (2) *Y_train* = the corresponding post-leakage recovery netlist (e.g., after running a signoff ECO tool on the initial netlist), again represented as the same directed acyclic graph, but with the nodes capturing only the post-recovery per-cell delay changes Δ_{delay} from pre-recovery delay values).

Train: a neural network model using *{X_train, Y_train}* to predict (i.e., **Inference**) the post-leakage recovery netlist *Y_test* for an unseen netlist, given its input *X_test* = the netlist's pre-recovery timing-optimized netlist with all of its pre-determined node features.

Formulation: We use the Graph Convolutional Networks (GCN) to formulate the post-recovery netlist prediction problem as a regression problem in directed acyclic graphs. Each cell in the netlist is labelled (output values for training) by its delay change from the recovery process. During the training process, the model minimizes the mean absolute loss of delay-change predictions over the complete graph, with the eventual goal of predicting the post-recovery node-level delay changes accurately. Once we have the per-node (i) delay changes, we translate these delay changes for each node (Δ^i_{delay}) to equivalent power changes (Δ^i_{power}). From the available space of cell-swaps and the node's electrical parameters (input slew and output load), we find the matching cell-swap that results in the model-predicted delay-change. We do this from the power-delay (function of input slew and load) look-up library table for each cell. In Figure 2, we show the pictorial illustration of our problem formulation.

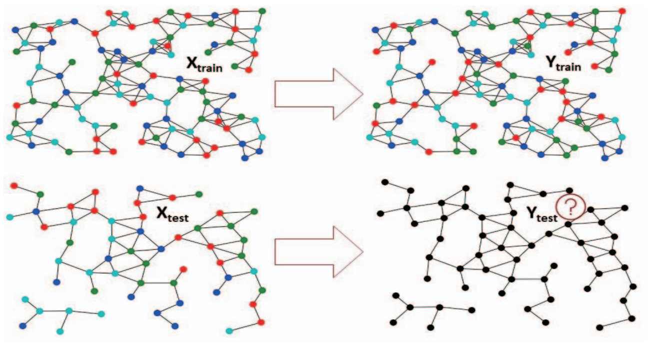

Figure 2: Netlist represented as a graph (undirected here for simplicity) and components (cell-types represented using various colors). The top row represents the training process using *X_train*, *Y_train*, and the bottom row illustrates the inference phase on an unseen graph, where the goal is to predict the per-node delay changes, and then translate these to matching cell-swaps.

Our Contributions. We do not seek to completely replace 'signoff' leakage recovery step performed by golden leakage recovery tools. We use machine learning to emulate the behaviour of such golden recovery tools, after observing the tool's behavior for multiple times. Such a model can quickly predict the outcome of leakage recovery step, and eventually help in reducing the license costs, along with faster design convergence time during methodology evaluations and time-critical tapeout phase. As detailed below, the main contributions of our work are as follows.

- To our knowledge, we are the first to completely rely on learning-based models, to solve the problem of predicting post-layout leakage recovery from a given netlist.
- We move away from the traditional approach of cell-based or path-based predictive models. In our approach, we retain the complete graph structure by translate the circuit netlist into a directed acylcic graph.
- A limitation of existing learning-based models in EDA is the ability to generalize on unseen designs. The reason for the inability lies in the formulation of the problem. With graph convolution framework predicting per-node recoverable slack, we seek to address this limitation and develop a design-independent model.

4 BACKGROUND

In regular structures, the two key advantages of any convolutional operator are (i) invariance and (ii) equivariance. The invariance property helps the representation become slightly invariant to small translations of the input. The equivariance property helps in sharing the learning parameters, thus making cardinality of learning parameters independent of the input size. Similar to the convolutional filter used in regular structures, the convolution operation on graphs involves two main steps (i) weighted aggregation of neighboring node features and (ii) applying some activation function (usually a non-linear) on the aggregated vector, to generate a latent representation of the node, in which the graph connectivity is embedded.

4.1 Notation

Now, we discuss mathematical notation that we use throughout this paper. A graph is represented as $G = (V, E)$, where V is the set of N vertices or nodes, and E is the set of edges. For our work, we assume that all the edges $e_{ij} \in E$ have unit weight.

- The adjacency matrix A representing the graph structure is a $N \times N$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ otherwise. Along with the influence of neighbors on a node's representation, to account for its own features, we introduce self loops in the adjacency matrix. Thus, $A_{ii} = 1$.
- Each node in the graph can be thought of having an input feature vector of dimension F, making the feature description $\vec{X}$ of the graph as a $N \times F$ feature matrix.
- Convolution operation and the activation function following it, can be represented using $f(A\vec{X}.\vec{W})$, where f is the activation function, and $\vec{W}$ is the shared weight vector.

The multiplication of feature vector $\vec{X}$ and A generates a new representation for each node, that is is an aggregation (simple summation) of node's neighbors and itself. With the introduction of a shared weight vector $\vec{W}$ of size $F \times F'$, the product $A\vec{X}.\vec{W}$ can be treated as a weighted aggregation over neighboring nodes and itself, where weights for each feature of its neighbors are embedded in $\vec{W}$. The output of such a matrix multiplication is an intermediate $N \times F'$ vector representation, that is a linear combination of its neighbors. After the above convolution operation, a non-linear activation generates a new non-linear representation for each node. If $\vec{H}^{(l)}$ is the node representation in a hidden layer (l), this combination of convolution and activation generates a new representation $\vec{H}^{(l+1)}$ in layer $(l + 1)$.

$$\vec{H}^{(l+1)} = f\left(\vec{H}^{(l)}.\vec{W}A\right)$$

A normalized version of the above convolution and activation can be obtained by making use of the diagonal node degree matrix D :

$$\vec{H}^{(l+1)} = f\left(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

In the convolution operation, Graph Attention Networks (GATs) introduce a self-attention mechanism based on node features. Meaning, when taking into account of neighbors, it is rational to give more attention to nodes that are similar to the node of interest. In problem involving timing graph, where a node is part of many timing paths, while computing the critical slack only few timing paths might be of interest. For example, using the arrival times on the pins and the propagation delays of the cells, critical paths can be traced. Therefore, attention factors can help in comprehending these critical paths. So, to generalize, GATs introduce an attention factor α, where α_{ij} represents a normalized (using softmax) attention factor of the edge e_{ij} based on the feature similarity.

$$\vec{H}^{(l+1)} = f\left(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^{(l)}W^{(l)}\alpha\right)$$

For each node in graph, a single convolution operation takes its neighbors into account while generating its representation. So, to account for the effect of nodes K hops from a given node, the natural choice is to perform a sequence of such convolution and activation operations, using K hidden layers.

4.2 Rationale for choosing GCN framework

In Figure 3, we illustrate the resulting delay changes during the leakage recovery process. Here, the timing path in red has highest positive slack of 40ps, whereas the green path has only 30ps positive slack. Any cell-swap changes the timing delay of all the paths associated with this swap, and impacts the chances of any further swaps because of its delay increase contribution all its associated timing paths. So, the available slack is shared among the cells, while adhering to individual path constraints. This per-cell recovered slack (indicated for each cell in the figure) is eventually manifested as per-cell leakage change. So, the fundamental change during the recovery process is the delay-change for each cell. The approach of directly predicting per-cell leakage power or predicting the post-recovery cell-type for each cell, restricts the model usage to a fixed design setup. Any change in experimental setup (such as opening more cells for swaps or using "dont_use" for few cell variants) needs a completely fresh model. Also, treating the problem as a cell-type (classification) prediction problem treats each of these cell-types as completely independent, without comprehending the smooth relation between these cell-types.

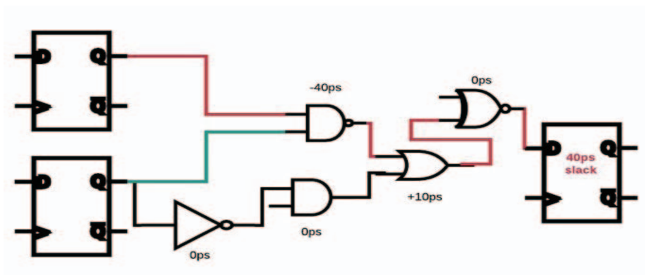

Figure 3: Post-recovery delay changes for each cell, while ensuring that new timing violations are not introduced. Here, -40ps indicates that the cell delay has increased by 40ps during the recovery process, and +10ps indicates that the cell delay has reduced by 10ps to ensure timing correctness of the green timing path.

So, by disentangling the delay-changes per cell from resulting cell-types, we makes the model somewhat independent on the problem setup, and therefore helps in generalizing the model across various design setups. Such a process involves a final post-processing step of matching the resulting delay change to a cell, eventually predicting the recoverable power. By using convolution operation over the neighbors, we generate a latent representation for each node, that embeds some of the path constraints. We begin with the hypothesis that having multiple sequences of such convolutions and repeating it over multiple training epochs will help us in predicting the per-node delay changes accurately. In Figure 4, we show training mean sqaured error (MSE) loss (for a design with 1.5M nodes) as a function of first 100 training epochs.

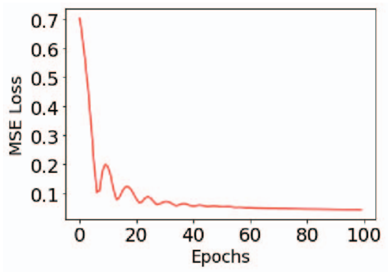

Figure 4: Training Loss during the first 100 epochs (*DesignA*).

4.3 Feature Selection

We evaluate a comprehensive set of instance-specific (nodes in the graph) electrical and physical features that influence leakage recovery of the design. We start with (i) instance's output pin arrival time, (ii) slack, (iii) worst load capacitance, (iv) fanout, and (v) instance's input pin's worst slew, (vi) pre-recovery leakage power, (vii) propagation delay as our basic features. We expand our feature list by

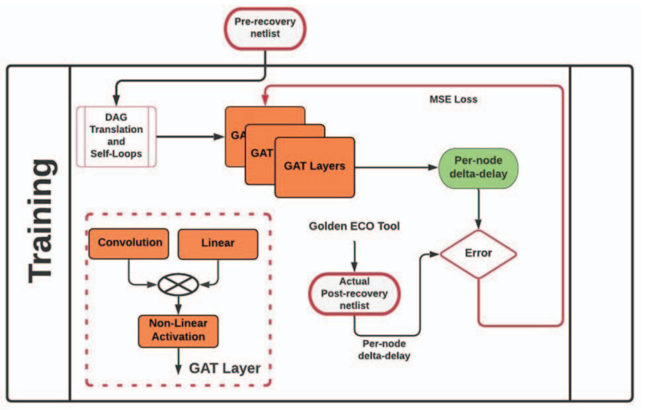

Figure 5: Summary of our model training flow. We use the mean-squared error (MSE) loss as our cost metric during back propagation.

adding four more features (viii) maximum power change, (ix) minimum power change, (x) maximum delay change, and (xi) minimum delay change from all possible swaps, that capture the physics of timing and power changes of each instance. We determine these, by estimating maximum and minimum gain that the instance can possibly achieve during optimization (among all possible swaps). We also include a (xii) sensitivity feature that is the ratio of maximum power and maximum delay change, thus making to a total of 12 features for each node. For the supervised learning task, the output for each node in the post-recovery netlist is represented by the instance-specific propagation delay (PD) change during recovery i.e., $\Delta Delay^i = PD^i_{post-recovery} - PD^i_{pre-recovery}$

5 MODEL FLOW

The modeling flow includes directed acyclic graph (DAG) construction along with graph's node-feature generation, model training phase, hyperparameter tuning, and model inference phase. Recall from our problem formulation (Section III) that the the pre-recovery netlist is represented using directed acyclic graphs X_train and Y_train. While generating the graph, we ignore the clock network (clock network is usually untouched for post-routing optimization). Also, to ensure that timing propagation stops at flipflop inputs, we clone the flip-flop nodes and remove the edge between the actual flop node and the cloned flop node. For each node, we add self-loops to account for its own features. Once we have the graph generated, we also get the corresponding node features from timing database.

The training phase of our supervised learning model is illustrated in Figure 5. Input graph contains node-wise features and adjacency matrix, and the output graph is represented using node-wise delta delay values ($\Delta delay^i$). As shown in the figure, we use a series of GAT layers to account for the effect of path-constraints and interacting timing paths. Each GAT layer takes into account, the effect of node one-hop away. Since convolution considers the local effect of the neighbors while determining the weight vector, we also combine a linear layer to capture the global effect. To recap, the goal of our model is to predict per-node delta delay value, and we use the mean squared loss of these delta delay values over all the nodes (i) as our optimization cost function.

$$\frac{\sum_i (\Delta delay^i_{model} - \Delta delay^i_{actual})^2}{\sum i}$$

Once the training phase is over, the model generates a weight vector ($\vec{W}$) and an attention factor (α) that can be used for the inference phase. Weight vector provides the relative weight of each feature while determining a node's representation. The inference

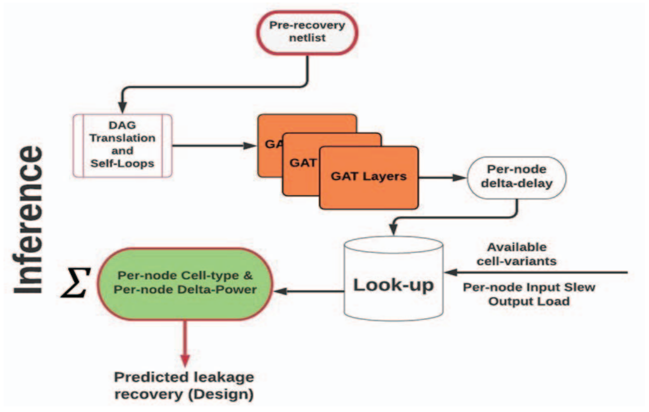
Inference
Pre-recovery netlist
DAG Translation and Self-Loops
GAT
GAT
GAT Layers
Per-node delta-delay
Look-up
Available cell-variants
Per-node Input Slew Output Load
Σ
Per-node Cell-type & Per-node Delta-Power
Predicted leakage recovery (Design)

Figure 6: The trained model predicts the post-recovery per-cell delay changes, using which we estimate the recovered leakage power.

phase involving DAG generation, and matrix multiplications, to predict per-node delta delay value is summarized in Figure 6. Once we have the predicted delta delay values, we determine the matching post-recovery cell-variant using the library look-up table given by f(slew, load, available variants). An important observation is that this modeling approach is independent of the cardinality of the input graph.

Model Configuration: We have a large number of hyperparameters (number of intermediate layers, cardinality of node's hidden representations in these layers, learning rate of the model, activation functions, and attention hyperparameters as well. We use Talos [5] framework to sweep across the space of hyper-parameters and choose the optimal hyperparameters. For example, in $DesignB_2$, we use seven convolution layers (GAT + linear); sizes and activation functions of each layer, determined by Talos.

6 EXPERIMENTAL VALIDATION

Our experiments use four benchmark designs [6] that were part of ISPD-13 Gate-sizing contest [7] and IWLS-05 benchmark suite [8]; that we name as *DesignA* (1.5M cells), *DesignB* (540K cells), *DesignC* (500K cells) and *DesignD* (72K cells). Our experiments are validated in 28nm FDSOI design enablement. For footprint compatible cell-swaps in the leakage recovery flow, we use three Vth-variants (Low, Standard and High) and four channel-length variants (P0, P4, P10, and P16). The eventual goal of the learning model is to accurately predict the magnitude of design's leakage recovery. So, we compare design's predicted leakage recovery ΔP_{mod}, with the golden tool's recovery outcome ΔP_{act}. The relative leakage-recovery difference is given by ϵ_{model} (%).

$$\epsilon_{model} = \frac{|\Delta P_{mod} - \Delta P_{act}|}{\Delta P_{act}} \; X \; 100\%$$

6.1 Experimental Results

We use process corner (typical, fast), clock period and placement-utilization parameters, to generate six ($i = 1, 2, 3, 4, 5, 6$) synthesis, place & route implementations for each of the four designs. We represent these variants using $DesignA_i$, $DesignB_i$, $DesignC_i$, $DesignD_i$. The netlists (in the form a DAG) from these variants are used for training and inference purposes. To recap, we seek to provide an estimate the leakage recovery that will result if a specific "golden tool" is launched.

Results of Design-Specific Model. To validate our model's usefulness in design-specific context (design-dependent), we train the model of three variants ($i = 4, 5, 6$) of each design, and test the model on three unseen implementation variants ($i = 1, 2, 3$). For all the four benchmark designs, Table 1 shows the results of our design-specific model for 28nm enablement. From the table, it is evident that the design-specific model can predict leakage recovery with relative error under 6.2%. Also, for $DesignB_2$, we observe that our predictive model can accurately estimate the relatively low leakage recovery. This proves the usefulness of our learning model, in predicting potential recoverable leakage power in a design.

Results of Cross-Design Model. To prove the usefulness of our model in design-independent usecase (cross-design), we train the model on three designs, and test the model on the fourth unseen design. For all the four benchmark designs, Table 2 shows the results of our design-independent model for 28nm enablement. From the table, it is evident that the design-independent model can predict leakage recovery with relative error less than 10%. For $DesignA_1$, the relative error is 9.1% (with the corresponding difference of power as 6 mW). The usefuleness of the model is to aid in making decisions about launch the signoff recovery flow or in a faster design-space exploration.

Summary of Results. The scatter plot for both design-specific and cross-design models are illustrated in the scatter plot Figure 7. In the plot, x-axis represents actual recovery for each design, and the y-axis represents predicted recovery from the learning model. As evident from the plot, most of the predictions are close to $y = x$ line.

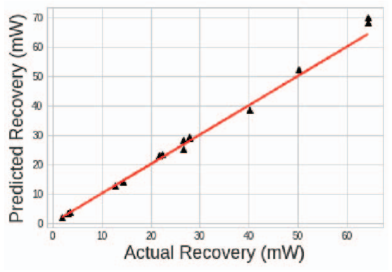

Figure 7: Scatter plot of leakage-recovery (Actual vs Predicted) for design-specific and cross-design experiments

6.2 Runtime Metrics

For a 1.6 M instance benchmark *DesignA*, on a Tesla P100 GPU, the inference runtime is around 12 seconds. Feature extraction and translating per-node delta-delay to power savings consume an additional 420 seconds. The total runtime for inference phase adds up to 432 seconds. However, with node-specific modeling, the feature extraction time and post-processing can be reduced with parallel processing. If we compare the inference runtime with the actual ECO tool runtime, the leakage recovery takes 10800 seconds for the same design implementation. As for all the learning models, a one-time investment for model training is required.

7 CONCLUSIONS

In this work, we show a new data-driven approach to prediction of leakage recovery in advanced-node IC design. We use the framework of Graph Convolution Networks (GCNs) to learn these optimizations over several graphs with various logic-cone structures, and generalize the learning to unseen graphs. For designs up to 1.6M instances, our design-specific prediction step achieves less than 6.2% discrepancy from the actual leakage-power recovered. As part of our next steps, we seek to validate our model on various contest benchmarks. We also seek to improve our cross-design model, with inclusion of artificial graphs that can span a larger distribution of sub-graphs. Also, we approximate the net connectivity using a pairwise relationships between pins of the cells, while in reality, the connectivity is

Table 1 Leakage recovery prediction for design-specific experiments.

28nm FDSOI					
Design	Initial Leakage Power (mW)	Post-recovery Leakage Power (Actual) (mW)	Recovered Leakage Power (Actual) (mW)	Recovered Leakage Power (Model) (mW)	Model Relative Error (ϵ_{model})
DesignA_1	271.9	207.7	64.2	**68.12**	6.2%
DesignA_2	229.2	179	50.2	**52.5**	4.5%
DesignA_3	130.7	90.6	40.1	**38.5**	3.9%
DesignB_1	81.8	55.2	26.6	**25.2**	5.2%
DesignB_2	62.1	54.3	7.8	**8.2**	5.1%
DesignB_3	39.2	26.3	12.8	**12.6**	1.5%
DesignC_1	83.9	55.9	27.9	**28.8**	3.2%
DesignC_2	60.1	38.5	21.7	**22.9**	5.5%
DesignC_3	36.02	21.6	14.4	**13.9**	3.5%
DesignD_1	16.06	12.9	3.16	**3.3**	4.4%
DesignD_2	8.7	5.1	3.6	**3.5**	2.7%
DesignD_3	5.3	3.4	1.9	**1.8**	5.2%

Table 2 Leakage recovery prediction for cross-design experiments.

28nm FDSOI					
Design	Initial Leakage Power (mW)	Post-recovery Leakage Power (Actual) (mW)	Recovered Leakage Power (Actual) (mW)	Recovered Leakage Power (Model) (mW)	Model Relative Error (ϵ_{model})
DesignA_1	271.9	207.7	64.2	**70.1**	9.1%
DesignB_1	81.8	55.2	26.6	**28.2**	5.2%
DesignC_1	83.9	55.9	27.9	**29.3**	5.0%
DesignD_1	16.06	12.9	3.16	**3.4**	7.5%

on a high-order hypergraph. So, using a combination of hypergraph convolution and attention is part of future works. We also seek to work on model's scalability for larger designs and the relation of GCN hyper-parameters with the design's logic-cone structure.

8 ACKNOWLEDGMENTS

We thank Prof. Andrew B. Kahng of U C San Diego, Dr. Tuck-Boon Chan and Ms. Payal Agarwal of Qualcomm, for their valuable contribution to this work. C. K. Cheng would like to acknowledge the support by the NSF under Grant CCF-1564302.

REFERENCES

[1] S. Bao, "Optimizing Leakage Power using Machine Learning", http://cs229.stanford.edu/proj2010/Bao_OptimizingLeakagePowerUsingMachineLearning.pdf, 2010, pp. 1-5.
[2] J. Derakhshandeh et al., "A Precise Model for Leakage Power Estimation in VLSI Circuits", *Proc. IDEAS*, 2005, pp. 1-4.
[3] M. Nemani and F. N. Najm, "High-Level Area and Power Estimation for VLSI Circuits", *IEEE TCAD* 18(6) (1999), pp. 697–713.
[4] M. Nemani and F. N. Najm, "Towards a High-Level Power Estimation Capability", *IEEE TCAD* 15(6) (1996), pp. 588–598.
[5] *Autonomio Talos*, 2019, http://github.com/autonomio/talos
[6] *OpenCores*, https://opencores.org
[7] M. M. Ozdal, C. Amin et al., "The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ISPD*, 2012, pp. 161-164.
[8] *IWLS 2005 Benchmarks*, https://iwls.org/iwls2005/benchmarks.html
[9] M. R. C. M. Berkelaar and J. A. G. Jess, "Gate Sizing in MOS Digital Circuits with Linear Programming", *Proc. EURO-DAC*, 1990, pp. 217–221.
[10] C.-P. Chen, C. Chi and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation", *IEEE TCAD* 18(7) (1999), pp. 1014–1025.
[11] A. B. Kahng, U. Mallappa, L. Saul and S. Tong, "Unobserved Corner Prediction: Reducing Timing Analysis Effort for Faster Design Convergence in Advanced-Node Design", *Proc. DATE*, 2019, pp. 168–173.
[12] D. G. Connery and K. Keutzer, "Linear Programming for Sizing, V_{th} and V_{dd} Assignment", *Proc. ISLPED*, 2005, pp. 149–154.
[13] H. Chou, Y.-H. Wang, and C. C.-P. Chen, "Fast and Effective Gate Sizing with Multiple-V_t Assignment Using Generalized Lagrangian Relaxation", *Proc. ASP-DAC*, 2005, pp. 381–386.
[14] A. B. Kahng, U. Mallappa, and L. Saul , "Using Machine Learning to Predict Path-Based Slack from Graph-Based Timing Analysis",*Proc. ICCD*, 2018, pp. 603–612.
[15] J. P. Fishburn and A. E. Dunlop, "Tilos: A Posynomial Programming Approach to Transistor Sizing", *Proc. ICCAD*, 1985, pp. 326–328.
[16] Y. L. Huang, J. Hu and W. Shi, "Lagrangian Relaxation for Gate Implementation Selection", *Proc. ISPD*, 2011, pp. 167-174.
[17] S. Hu, M. Ketkar and J. Hu, "Gate Sizing for Cell-Library-Based Designs", *IEEE TCAD* 28(6) (2009), pp. 818–825.
[18] J. Hu et al., "Sensitivity-guided Metaheuristics for Accurate Discrete Gate Sizing", *Proc. ICCAD*, 2012, pp. 233–239.
[19] K. Jeong et al., "Revisiting the Linear Programming Framework for Leakage Power vs. Performance Optimization", *Proc. ISQED*, 2009, pp. 127–134.
[20] A. B. Kahng, S. Kang, H. Lee, I. L. Markov and P. Thapar, "High-Performance Gate Sizing with a Signoff Timer", *Proc. ICCAD*, 2013, pp. 450–457.
[21] Y. Liu and J. Hu, "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment", *IEEE TCAD* 29(2) (2010), pp. 223–234.
[22] V. S. Livramento et al., "A Hybrid Technique for Discrete Gate Sizing Based on Lagrangian Relaxation", *ACM TODAES* 19(4) (2014), no. 40.
[23] L. Li et al., "An Efficient Algorithm for Library-based Cell-type Selection in High-Performance Low-Power Designs", *Proc. ICCAD*, 2012, pp. 226–232.
[24] M. M. Ozdal et al., "Gate Sizing and Device Technology Selection Algorithms for High-Performance Industrial Designs", *Proc. ICCAD*, 2011, pp. 724–731.
[25] M. Rahman, H. Tennakoon and C. Sechen, "Power Reduction via Near-Optimal Library-Based Cell-Size Selection", *Proc. DATE*, 2011, pp. 867–870.
[26] M. Rahman and C. Sechen, "Post-synthesis Leakage Power Minimization", *Proc. DATE*, 2012, pp. 99–104.
[27] S. Held, "Gate Sizing for Large Cell-Based Designs", *Proc. DATE*, 2009, pp. 827–832.
[28] T. Reimann, G. Posser, G. Flach, M. Johann and R. Reis, "Simultaneous Gate Sizing and Vt Assignment using Fanin/Fanout Ratio and Simulated Annealing", *Proc. ISCAS*, 2013, pp. 2549–2552.
[29] S. Roy et al., "OSFA: A New Paradigm of Gate-sizing for Power/Performance Optimizations under Multiple Operating Conditions", *Proc. DAC*, 2015.
[30] http://vlsicad.ucsd.edu/SIZING/optimizer.html, UCSD SensOpt Leakage Optimizer (A. B. Kahng, S. Kang, 2010-2011).
[31] A. Srivastava et al., "Power Minimization Using Simultaneous Gate Sizing, Dual-V_{dd} and Dual-V_{th} Assignment", *Proc. DAC*, 2004, pp. 783–787.
[32] H. Tennakoon and C. Sechen, "Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-Based Pre-Processing Step", *Proc. ICCAD*, 2002, pp. 395–402.
[33] T.-H. Wu and A. Davoodi, "PaRS: Parallel and Near-Optimal Grid-Based Cell Sizing for Library-Based Design", *IEEE TCAD* 28(11) (2009), pp. 1666–1678.
[34] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks. *Proc. ICLR*, 2017.
[35] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. A. Guzik, and R. P. Adams, Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, pp. 2224–2232, 2015.
[36] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, Structural-rnn: Deep learning on spatio-temporal graphs, *Proc. CVPR*, 2016, pp. 5308–5317.
[37] W. Hamilton, Z. Ying, and J. Leskovec, Inductive representation learning on large graphs, *Proc. NIPS*, 2017, pp. 1024–1034.
[38] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, Graph attention networks, *Proc. ICLR*, 2017.