A Circuit Domain Generalization Framework for Efficient Logic Synthesis in Chip Design

Zhihai Wang [*1] **Lei Chen** [*2] **Jie Wang** [1✉] **Yinqi Bai** [1] **Xing Li** [2] **Xijun Li** [2] **Mingxuan Yuan** [2] **Jianye Hao** [2 3] **Yongdong Zhang** [1] **Feng Wu** [1]

Abstract

Logic Synthesis (LS) plays a vital role in chip design. A key task in LS is to simplify circuits—modeled by directed acyclic graphs (DAGs)—with functionality-equivalent transformations. To tackle this task, many LS heuristics apply transformations to subgraphs—rooted at each node on an input DAG—sequentially. However, we found that a large number of transformations are *ineffective*, which makes applying these heuristics highly time-consuming. In particular, we notice that the runtime of the Resub and Mfs2 heuristics often dominates the overall runtime of LS optimization processes. To address this challenge, we propose a novel data-driven LS heuristic paradigm, namely PruneX, to reduce ineffective transformations. The major challenge of developing PruneX is to learn models that well generalize to unseen circuits, i.e., the *out-of-distribution (OOD) generalization* problem. Thus, the major technical contribution of PruneX is the novel *circuit domain generalization* framework, which learns domain-invariant representations based on the transformation-invariant domain-knowledge. To the best of our knowledge, PruneX is *the first* approach to tackle the OOD problem in LS heuristics. We integrate PruneX with the aforementioned Resub and Mfs2 heuristics. Experiments demonstrate that PruneX significantly improves their efficiency while keeping comparable optimization performance on industrial and very large-scale circuits, achieving up to $3.1\times$ faster runtime.

[*]Equal contribution This work was done when Zhihai Wang was an intern at Huawei. [1]CAS Key Laboratory of Technology in GIPAS & MoE Key Laboratory of Brain-inspired Intelligent Perception and Cognition, University of Science and Technology of China [2]Noah's Ark Lab, Huawei Technologies [3]College of Intelligence and Computing, Tianjin University. Correspondence to: Jie Wang <jiewangx@ustc.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Chip design is a cornerstone of the worldwide semiconductor industry, promoting the development of an extensive market of electronic devices, such as cellular phones, personal computers, smart automobiles, etc. Logic Synthesis (LS) is one of the most important modules in chip design, which aims to transform a behavioral-level description of a design into an optimized gate-level circuit (Berndt et al., 2022; Pasandi et al., 2023). In brief, LS is the "compiler" in chip design. A key task in LS is Circuit Optimization (CO), which aims to transform an input circuit into a simplified circuit with equivalent functionality and reduced size and/or depth. Thus, it is crucial to well tackle the CO task, as it can significantly improve the Quality of Results, i.e., various metrics that evaluate the quality of designed chips, such as the area, delay, and performance of the chips (De Abreu et al., 2021; Bertacco & Damiani, 1997; Berndt et al., 2022).

However, the CO task can be extremely hard to tackle as it is a $\mathcal{NP}$-hard problem (Micheli, 1994; Farrahi & Sarrafzadeh, 1994; Pasandi et al., 2023). To approximately tackle the CO task, many existing LS frameworks (Brayton & Mishchenko, 2010; Jiang et al.), including the open-source state-of-the-art LS framework known as ABC (Brayton & Mishchenko, 2010), have developed a rich set of LS heuristics, such as Mfs2 (Mishchenko et al., 2011), Resub (Brayton, 2006), Rewrite (Bertacco & Damiani, 1997; Mishchenko et al., 2006), Refactor (Brayton, 1982; Mishchenko et al., 2006), etc. Specifically, given an input circuit modeled by a directed acyclic graph (DAG), many commonly used LS heuristics apply transformations to subgraphs rooted at each node—that is, the node-level transformation—sequentially for all nodes on the DAG. We illustrate a unified paradigm of these LS heuristics as shown in Figure 1.

However, we found an important problem leading to inefficient LS, that is, a large number of node-level transformations are *ineffective*, making applying these heuristics highly time-consuming. In particular, we notice that applying the Resub (Brayton, 2006) and Mfs2 (Mishchenko et al., 2011) heuristics take much longer runtime, roughly ranging from $30\times$ to $70\times$, than the other heuristics (see Figure 2). Moreover, we found that the runtime of the two heuris-

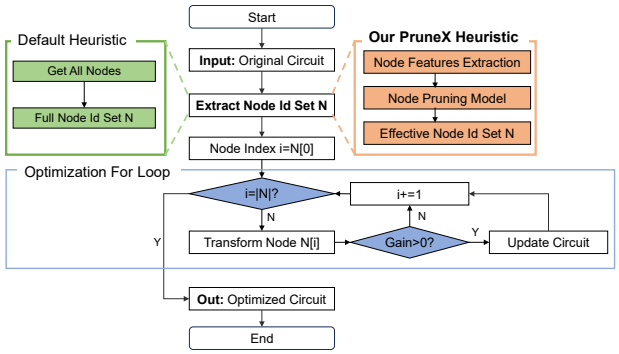


Figure 1. We illustrate the paradigm of many traditional LS heuristics and the PruneX LS heuristic.

tics often dominates the overall runtime of LS optimization processes—accounts for approximately 79% of the overall runtime (see Appendix B.2.2). Thus, the runtime of the two heuristics acts as a bottleneck to the efficiency of LS, and inefficient LS may significantly increase the Time to Market (Neto et al., 2021a; Sabbavarapu et al., 2014; Reddy et al., 2014), i.e., the overall duration for developing and commercializing new chips. Moreover, inefficient LS could significantly degrade the Quality of Results. For example, chip designers often have to reduce the number of times of using time-consuming LS heuristics to optimize large-scale circuits in design space exploration, which may significantly increase the area and delay of chips (Yuan et al., 2023).

To promote efficient LS, we propose a novel data-driven LS heuristics paradigm (see Figure 1), namely PruneX, which can significantly improve the efficiency of LS heuristics by learning to reduce a large number of ineffective node-level transformations. An appealing feature of PruneX is that it is applicable to many commonly used LS heuristics—which follow the paradigm illustrated in Figure 1—to significantly improve their efficiency. Specifically, PruneX learns a generalizable classifier to predict nodes with ineffective transformations (ineffective nodes) accurately on unseen circuits, and avoids applying transformations to these nodes. However, PruneX could significantly degrade the optimization performance compared to default heuristics when inaccurately classifying effective nodes in unseen circuits. Thus, the major challenge of developing PruneX is how to learn models that can well generalize to unseen circuits, that is, the out-of-distribution (OOD) generalization problem across circuits in LS heuristics (see Figure 3).

To enable generalization, the major technical contribution of PruneX is the novel **c**ircuit d**o**main **g**eneralization (COG) framework (see Figure 4) to learn domain-invariant representations for generalizable classifiers based on the transformation-invariant domain knowledge. To the best of our knowledge, COG is *the first* data-driven method to well formulate and tackle the OOD generalization problem across circuits in LS, which is critical for the success of data-driven LS algorithms. Specifically, COG formulates the OOD problem as a circuit domain generalization task by dividing circuits into multiple circuit domains instead of combining all circuits into a single dataset. The novel formulation allows us to incorporate prior knowledge regarding circuits into model learning to enhance the generalization capability of our models. Then, COG proposes to learn domain-invariant node representations by aligning node embeddings across circuits from different domains based on the transformation-invariant domain knowledge—a node-level transformation is significantly associated with the local subgraph rooted at the node, regardless of its associated circuit.

We integrate PruneX with the two aforementioned most time-consuming heuristics, i.e., the Resub (Brayton, 2006) and Mfs2 (Mishchenko et al., 2011) heuristics, among commonly used heuristics. Extensive experiments demonstrate that PruneX significantly and consistently improves their efficiency while keeping comparable optimization performance on three challenging benchmarks, achieving up to 3.1× faster runtime. The challenging benchmarks include industrial and very large-scale circuits. Moreover, we conduct experiments to demonstrate that applying PruneX heuristics twice can significantly improve the optimization performance while achieving faster runtime. Note that a one percent improvement in the optimization performance may yield substantial economic value. We release our codes and data at `https://github.com/MIRALab-USTC/AI4LogicSynthesis-PruneX`.

We summarize our major contributions as follows. (1) We found an important problem that leads to inefficient LS, i.e., many LS heuristics apply a large number of *ineffective* node-level transformations (see Figure 2). (2) To promote efficient LS, we propose an effective data-driven LS heuristic paradigm, namely PruneX, which is applicable to many LS heuristics to significantly improve their efficiency. (3) The major technical contribution of PruneX is the novel circuit domain generalization framework to learn domain-invariant representations based on the transformation-invariant domain knowledge. (4) To the best of our knowledge, PruneX is *the first* data-driven method to tackle the OOD generalization problem in LS heuristics, which is critical for the success of data-driven LS algorithms. (5) Experiments demonstrate that PruneX significantly and consistently improves the efficiency of LS heuristics on three challenging benchmarks, including industrial and very large-scale circuits.

2. Related Work and Background

Machine Learning for Logic Synthesis As chip complexity has grown exponentially with the development of semiconductor technology, using machine learning (ML) to assist the automated chip design workflow has been an active topic of significant interest in recent years (Mirhoseini et al., 2021; Huang et al., 2021; Sánchez et al., 2023; Neto et al., 2021a;

Lai et al., 2022; 2023; Ren & Hu, 2023). In most of stages in the chip design workflow, recent studies have demonstrated significant improvement by using ML methods compared with traditional methods, including high-level synthesis (Makrani et al., 2019; Kim et al., 2018; Liu & Carloni, 2013), logic synthesis (Neto et al., 2021a; Li et al., 2023a; Berndt et al., 2022; Neto et al., 2019a; Li et al., 2024a), and placement (Mirhoseini et al., 2021; Lai et al., 2022; 2023; Agnesina et al., 2023; Cheng et al., 2022). In this paper, we focus on using machine learning to promote efficient logic synthesis (LS), which plays a vital role in efficient chip design and can yield substantial economic value (Fawcett, 1994; Neto et al., 2021a). Existing research on machine learning for LS can be roughly divided into three categories (Berndt et al., 2022; Ren & Hu, 2023). First, (Synopsys, 2020; Cadence, 2021; Hosny et al., 2020b; Grosnit et al., 2022) use machine learning to tune the optimization flow of LS operators. Second, (Neto et al., 2021a; Kirby et al., 2019b; Zhou et al., 2019) use machine learning to predict key metrics after physical design and leverage the prediction to guide LS optimization. Third, (Neto et al., 2021b; 2019b) use machine learning to improve decision-making in traditional LS methods. Different from existing work, our work uses machine learning to improve the paradigm of traditional LS operators to promote efficient LS. An appealing feature of our PruneX is that it is applicable to many LS operators—which follow the paradigm illustrated in Figure 1—to significantly improve their efficiency.

Generalizable Prediction in Chip Design Workflow Prior research has investigated the utilization of machine learning (ML) techniques to develop generalizable congestion prediction models within the chip design workflow (Kirby et al., 2019a; Wang et al., 2022a). Nevertheless, our work differs from previous studies in two fundamental aspects. **First**, we address dissimilar input data. Prior research mainly focuses on the physical design stage with circuits represented by gate-level netlists or designed layouts, while we focus on the logic synthesis stage with circuits represented by Boolean networks. **Second**, we employ different methodologies for learning models. Generally, they propose problem-specific graph neural network architectures and learn the models by formulating all circuits as a single dataset. In contrast, our PruneX formulates the OOD generalization problem across circuits as a novel circuit domain generalization task and learn domain-invariant representations for enhanced generalization capabilities, which offers promising avenues for future research on prediction tasks in chip design. We defer more details about related work in Appendix D.1

Background on Circuit Representation In the LS stage, a circuit is usually modeled by a Boolean network. In this paper, we use the terms Boolean network and circuit interchangeably. A Boolean network is a directed acyclic graph (DAG), where nodes correspond to Boolean functions and

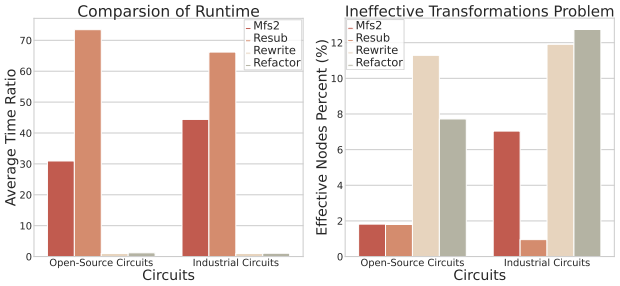


Figure 2. The results demonstrate that applying the Resub and Mfs2 heuristics are the most time-consuming among commonly used heuristics (Left), and a large number of transformations are ineffective in commonly-used heuristics (Right).

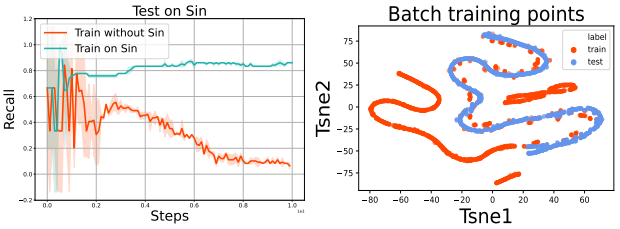


Figure 3. The results show that models trained without Sin (red) perform poorly on Sin. Furthermore, we visualize the data points from the dataset without Sin (red) and Sin circuit (blue), respectively. The results suggest that distribution shift indeed exists.

directed edges correspond to wires connecting these functions. A Boolean function takes the form $f : \mathbf{B}^n \to \mathbf{B}$, where $\mathbf{B} = \{0, 1\}$ denotes the Boolean domain. Given a node, its *fanins* are nodes connected by incoming edges of this node, and its *fanouts* are nodes connected by outgoing edges of this node. The *primary inputs (PIs)* are nodes with no fanin, and the *primary outputs (POs)* are nodes with no fanout. The *size* of a circuit denotes the number of nodes in the DAG. The *depth (level)* of a circuit denotes the maximal length of a path from a PI to a PO in the DAG. The size and depth of a circuit are proxy metrics for the area and delay of the circuit, respectively. We defer more details on Background to Appendix D.2.

3. Motivating Results

Problem Challenge: Efficiency Analysis of LS heuristics First, we evaluate the runtime of the Mfs2 and Resub heuristics over two commonly-used LS optimization optimization sequence flows (Brayton, 2006; Mishchenko et al., 2011). The results in Appendix B.2.2 demonstrate that the runtime of the two heuristics often dominates the overall runtime of LS optimization process—accounts for approximately 79% of the overall runtime. Second, we further compare the runtime of applying four commonly-used LS heuristics to open-source and industrial circuits. Specifically, the four heuristics include Rewrite (Bertacco & Damiani, 1997), Refactor (Brayton, 1982), Resub (Brayton, 2006), Mfs2 (Mishchenko et al., 2011). The results in Figure 2 show that applying the Resub and Mfs2 heuristics are the most time-consuming among commonly used heuristics. Specifically,

we report the ratio of the runtime of these heuristics to that of the Rewrite heuristic. The results demonstrate that applying the Resub and Mfs2 heuristics take much longer runtime, roughly ranging from $30\times$ to $70\times$, than the Rewrite heuristic. Therefore, the runtime of applying the Resub and Mfs2 heuristics acts as a bottleneck to the efficiency of LS.

Problem Challenge: Ineffective Node-Level Transformations Problem As shown in Figure 2, the Mfs2 and Resub heuristics apply a large number of Ineffective Node-Level Transformations (exceeding 93%) on open-source and industrial circuits. In addition, the results in Figure 2 show that a large number of node-level transformations applied by the Rewrite and Refactor heuristics are ineffective as well (exceeding 87%). We defer more results to Appendix B.2.1.

Technical Challenge: Out-of-Distribution (OOD) Generalization Problem in LS Motivated by the aforementioned analysis, we propose to learn models to reduce ineffective transformations to significantly improve the runtime of LS heuristics. However, we empirically show that the optimization performance of the heuristics significantly degrades as the prediction recall decreases in Appendix B.2.4. Thus, to achieve comparable optimization performance with default heuristics, our learned models require achieving high prediction recall, which is extremely challenging due to the following OOD Generalization Problem in LS. Specifically, we use the Mfs2 heuristic to collect a training dataset with all circuits from the EPFL benchmark (Amarú et al., 2015) and another one with all circuits except the Sin circuit. We use EnsembleMLP—a simple learning baseline using supervised learning—to train models on these two datasets, and evaluate the model on Sin. The results in Figure 3 show that it is challenging for models trained on circuits without Sin to generalize to the unseen Sin circuit. Moreover, we visualize the batch data from the training dataset without Sin and the Sin circuit as shown in Figure 3. The results show that the data distributions from the training and testing datasets are different. Moreover, we empirically show that the data distributions from different circuits are similar but different in Appendix B.2.3. Due to the distribution shift, it is extremely challenging to learn models that can well generalize to unseen circuits, i.e., the OOD generalization problem across circuits in LS heuristics.

4. Methods

To promote efficient Logic Synthesis (LS), we first provide a detailed description of our proposed PruneX heuristics paradigm in Section 4.1. Then we present our proposed **c**ircuit d**o**main **g**eneralization (COG) framework to learn generalizable classification models in LS in Section 4.2.

4.1. A Novel Data-Driven LS heuristic Paradigm

As shown in Figure 2, we found that the Ineffective Node-Level Transformations (INT) problem in many LS heuristics makes applying these heuristics highly time-consuming. To address this challenge, we propose a novel data-driven LS heuristics paradigm, namely PruneX, which incorporates learned classifiers into these heuristics to improve their efficiency. Specifically, PruneX consists of two phases: the offline and online phases. **(1) The Offline Phase: Data Collection and Model Learning** In this phase, we aim to collect a training dataset $\mathcal{D}$ from multiple existing logic circuits and learn a classification model from the dataset. Let $\{\mathbf{x}_j^i, y_j^i\}_{j=1}^{n_i}$ denote the data from the logic circuit. We define a **Circuit Dataset** by the data $\mathcal{D}_i = \{\mathbf{x}_j^i, y_j^i\}_{j=1}^{n_i}$. Let X denote a LS heuristic, such as Mfs2, Resub, Rewrite. Given the X heuristic and N circuits, we generate a dataset $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^N = \{\{\mathbf{x}_j^i, y_j^i\}_{j=1}^{n_i}\}_{i=1}^N$. Given the generated dataset $\mathcal{D}$, we learn a binary classifier $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ denotes the input space of nodes, and $\mathcal{Y}$ denotes $\{0, 1\}$. We provide more details on data collection and model learning in Appendix E.8.1. **(2) The Online Phase: Incorporating Learned Models into LS heuristics** As shown in Figure 1, PruneX uses the learned classifier to predict ineffective nodes to accelerate the X heuristic in the online phase.

4.2. A Circuit Domain Generalization Framework

In this subsection, we present our proposed **c**ircuit d**o**main **g**eneralization (COG) framework, which learns generalizable and scalable classifiers. Specifically, COG consists of three main components. (a) COG formulates the learning task as a circuit domain generalization task. (b) COG proposes a knowledge-driven subgraph representation learning method to learn domain-invariant representations. (c) COG proposes a domain-aware distributional classifier.

(a) A Circuit Domain Generalization Formulation Due to the distribution shift between training and testing Circuit Datasets as shown in Figure 3, learning classification models that well generalize to unseen circuits is challenging, i.e., the out-of-distribution (OOD) generalization problem. To address this challenge, we propose to formulate the OOD generalization problem across circuits as a circuit domain generalization task by proposing to divide circuits into multiple Circuit Domains. Instead of combining all circuits into a single dataset, the novel formulation allows us to *incorporate prior knowledge about circuits into model learning* to enhance the generalization capability of our models.

However, unlike traditional domain generalization setting in computer vision (Shen et al., 2021; Wang et al., 2022b), where each domain dataset is given as a priori, our circuit domain generalization problem necessitates the development of a mechanism for dividing the Circuit Domains. Thus, we first propose to formulate each circuit as an individual circuit domain. However, the sample sizes of different circuits can be quite variable, leading to two undesirable challenges. First, the sizes of some circuits are small, with only about 100 nodes. This results in a few shot learning challenge

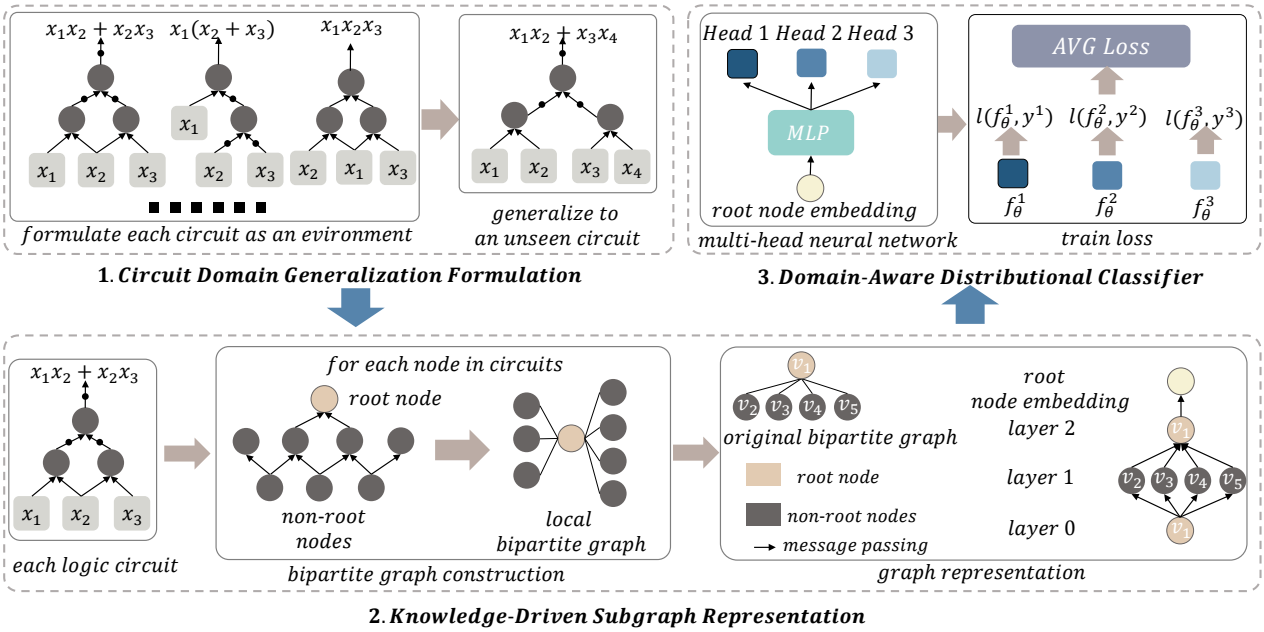


Figure 4. We illustrate our circuit domain generalization (COG) framework. Our COG consists of three main components to learn classification models that can well generalize to unseen circuits. Please see Section 4.2 for details.

(Wang et al., 2020). Second, the sizes of different circuits range from 100 to 100,000 nodes, resulting in an extreme imbalance of sample sizes across different Circuit Datasets. To address these challenges, we further propose a circuit aggregation mechanism g that maps the N Circuit Datasets to M hyper-circuit datasets ($M \leq N$) by aggregating certain circuits with related characteristics, such as the functionality. That is, $g(\{\mathcal{D}_i\}_{i=1}^N) = \{\mathcal{D}'_k\}_{k=1}^M$, where $\mathcal{D}'_k$ denotes the data from the k-th aggregated dataset. For each $\mathcal{D}'_k$, we assume that its data $\{\mathbf{x}_j^k, y_j^k\}_{j=1}^{n_k}$ are sampled from an underlying distribution $\mathbb{P}_{XY}^k$. We define a **Circuit Domain** by the underlying distribution $\mathbb{P}_{XY}^k$. Moreover, given M aggregated datasets $\{\mathcal{D}'_k\}_{k=1}^M$ from M source Circuit Domains $\mathbb{P}_{XY}^1, \ldots, \mathbb{P}_{XY}^M$, where $\mathbb{P}_{XY}^i \neq \mathbb{P}_{XY}^j$, $1 \leq i \neq j \leq M$, we define **Circuit Domain Generalization** as to learn a robust and generalizable predictive function $f : \mathcal{X} \to \mathcal{Y}$ from the M source domains, which achieves a minimum prediction error on an unseen testing circuit sampled from $\mathbb{P}_{XY}^t$. To the best of our knowledge, COG takes *the first* step towards formulating the OOD generalization problem across circuits as a circuit domain generalization task. The novel formulation is critical for the success of our COG, and carries the potential to motivate future remarkable research on this OOD generalization problem in chip design.

It remains challenging to instantiate the idea of aggregating circuits with related functionality, as the availability of functionality information for circuits may be limited or unavailable. To alleviate this problem, we make some theoretical analysis as follows, and the theoretical results show two more basic principles for the circuit aggregation mechanism g. (1) The sample sizes of different Circuit Domains should be close. (2) It is undesirable to aggregate all Circuit Datasets into one mixed domain. Finally, we propose a simple method to aggregate circuits based on their functionality and sample sizes (see Appendix E.6 for details).

Theoretical Analysis Based on the domain knowledge of the OOD generalization problems, we make the following assumption, which is commonly used in traditional domain generalization literature (Blanchard et al., 2011).

Assumption 4.1. The aggregated training Circuit Domains $\mathbb{P}_{XY}^1, \ldots, \mathbb{P}_{XY}^M$ are i.i.d. realizations from a hyper-distribution $\mathscr{P}$ and any possible testing Circuit Domain $\mathbb{P}_{XY}^t$ follows the same hyper-distribution.

The assumption usually holds in practice for our OOD problem with our circuit aggregation mechanism. We defer detailed discussion on this assumption to Appendix E.5. Based on Assumption 4.1, the learned classifier should include the Circuit Domain information $\mathbb{P}_{XY}$ into its input. In our case, the functional relationship $\mathbb{P}_{Y|X}$ across different Circuit Domains is stable, as whether a node-level transformation is effective is stable across different circuits. Thus, the prediction of the classifier takes the form of $y = f(\mathbb{P}_X, \mathbf{x})$ since the marginal distribution $\mathbb{P}_X$ contains the whole domain-specific information. Given the model f, its average risk over all possible target Circuit Domains takes the form of $\mathcal{R}(f) = \mathbb{E}_{\mathbb{P}_{XY}^t \sim \mathscr{P}} \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{XY}^t}[l(f(\mathbb{P}_X^t, \mathbf{x}), y)]$. Unfortunately, the expectation is intractable as the hyper-distribution $\mathscr{P}$ is unknown. Nevertheless, we can estimate $\mathcal{R}(f)$ using our empirical risk estimation objective, which takes the form of $\hat{\mathcal{R}}(f) = \frac{1}{M}\sum_{k=1}^M \frac{1}{n_k}\sum_{j=1}^{n_k} l(f(\hat{\mathbb{P}}_X^k, \mathbf{x}_j^k), y_j^k)$. M denotes the number of training Circuit Domains, n_k denotes the sample size of the k-th Circuit Domain, and $\hat{\mathbb{P}}_X^k$ denotes

the empirical estimation of the k-th Circuit Domain $\mathbb{P}_X^k$. To evaluate how well our estimation is, we show that there is a generalization error bound between $\mathcal{R}(f)$ and $\hat{\mathcal{R}}(f)$, which is inspired by previous work (Blanchard et al., 2011).

Theorem 4.2. *Under some mild and reasonable assumptions, we can conclude that with probability at least* $1-\delta$

$$\left(\sup_{f\in B_{\bar{K}}(r)}|\mathcal{R}(f)-\hat{\mathcal{R}}(f)|\right)^2 \leq \frac{C_1\log\delta^{-1}+C_2}{M} + \frac{C_3\log 2\delta^{-1}M+C_4\log\delta^{-1}+C_5}{M^2}\sum_{k=1}^{M}\frac{1}{n_k} \quad (1)$$

where C_1, C_2, C_3, C_4, C_5 are constants, $B_{\bar{K}}(r)$ denotes the ball of radius r of an Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_{\bar{K}}$, M is the number of training Circuit Domains and n_k is the sample size of the k-th Circuit Domain.

The first term can be viewed as hyper-distribution estimation error, and the second term measures the average sampling error across M circuit domains. Please refer to Appendix A.1 for detailed proof. Theorem 4.2 shows that the generalization error bound depends on both M and n_k, thus depending on the circuit aggregation mechanism g. Due to the aggregation mechanism, M and n_k are variable, and the total number of samples n is fixed. This is quite different from existing domain generalization work. Based on Theorem 4.2, we show the following Corollaries to provide theoretical insights into the circuit aggregation mechanism.

Corollary 4.3. *If the number of Circuit Domains M and the total number of samples n is fixed, the generalization error bound reaches its minimum when $n_k = \frac{n}{M}$ for $k = 1, 2, \ldots, M$.*

Corollary 4.4. *Under some mild conditions, using domain-wise training circuit datasets (i.e., $M > 1$) will result in a smaller generalization error bound than just pooling them into one mixed dataset (i.e., $M = 1$).*

As a result, we easily conclude the aforementioned two principles for the circuit aggregation mechanism g inspired by Corollaries 4.3 and 4.4. We defer more discussion and detailed proof to Appendixes A.2 and A.3.

(b) Knowledge-Driven Subgraph Representation Previous work (Muandet et al., 2013) has theoretically and/or empirically shown that *learning domain-invariant representations* can well generalize to unseen domains. Thus, we propose a novel knowledge-driven subgraph representation (KDSR) learning method to learn domain-invariant representations based on the transformation-invariant domain knowledge. That is, the node-level transformation mechanism is invariant across circuits for a given LS heuristic. We defer more discussion on the transformation-invariant domain knowledge to Appendix E.7. Based on the transformation-invariant domain knowledge, KDSR effectively aligns node embeddings by focusing on the constructed subgraphs rooted at each node in the DAGs. In general, the construction of a subgraph in LS heuristics comprises the root node and a restricted number of its neighboring nodes. For further node embedding alignment, we transform the subgraph into a bipartite graph by modeling the root node and non-root nodes as two classes of nodes (see Figure 4). To encode the bipartite graph, we propose to leverage a graph convolutional neural network (GCNN) (Gori et al., 2005). Our GCNN takes as input the bipartite graph $\mathbf{x} = (\mathbf{T}, \mathbf{C}, \mathbf{A})$, where $\mathbf{T} \in \mathbb{R}^{1\times c}$ denotes the feature matrix of the root node, $\mathbf{C} \in \mathbb{R}^{m\times c}$ denotes the feature matrix of the non-root nodes, and $\mathbf{A} \in \mathbb{R}^{1\times m}$ denotes the adjacency matrix of the graph. In detail, our bipartite graph is a fully-connected graph. That is, $\mathbf{A}_{1j} = 1$ for all $j \in \{1, 2, \ldots, m\}$. We manually design the node features to contain its basic and functionality information (see Appendix E.8.2). Due to the bipartite structure of the input graph, our GCNN model g_ϕ performs a single graph convolution, in the form of two interleaved half-convolutions. Following the graph-convolution layer, we obtain a bipartite graph with the same topology as the input, but with the root node embedding $\mathbf{h}(\mathbf{t}_1) = g_\phi(\mathbf{x}, \mathbf{t}_1)$, where $\mathbf{t}_1$ denotes the root node features. The embedding contains rich information from the non-root nodes for discriminative and generalizable classification. We defer more details in Appendix E.4.

(c) Domain-Aware Distributional Classifier Based on the node embeddings given by the GCNN model, we further propose a domain-aware distributional classifier (DADC), which well incorporates the domain-specific information into parameterized models. Specifically, we parameterize the DADC via a multi-head neural network, where each head f_θ^k learns a classifier under the corresponding training domain. The multi-head neural network is a shared neural network architecture with M heads branching off independently as shown in Figure 4. The optimization objective for our proposed DADC takes the form of $\hat{\mathcal{R}}(\theta) = \frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k} l(f_\theta^k(g_\phi(\mathbf{x}_j^k, \mathbf{t}_1)), y_j^k)$, where l denotes the cross-entropy loss, $f_\theta^k(\cdot)$ denotes the output of the k-th head, and $g_\phi(\mathbf{x}_j^k, \mathbf{t}_1)$ denotes the node embedding given by the GCNN model. In the test phase, we use the *mean* of the M head values to approximate the classification values under testing circuits. Please see Appendix E.8.5 for more details. A major advantage of DADC is to learn a distributional representation of classifiers, which can well capture the uncertainty of classification values to enhance its robustness against the distribution shift.

(d) Discussion on Advantages and Generality First, our method is applicable to many commonly used LS heuristics, and can significantly improve their efficiency. Second, our method takes the first step towards formulating the OOD generalization problem across circuits as a novel circuit domain generalization task, which has broad applicability

in many prediction problems in the chip design workflow. Third, our graph learning method focuses on small-scale subgraphs rather than global graphs, enabling efficient parallel training and high scalability to very large-scale circuits.

5. Experiments

We conduct extensive experiments to evaluate PruneX with COG (PruneX-COG). Specifically, our experiments have four main goals: (1) to demonstrate that PruneX-COG can accurately predict effective nodes, and significantly improve the runtime of LS heuristics with comparable optimization performance (i.e., Quality of Results, QoR); (2) to show the effectiveness of PruneX-COG on industrial circuits and very large-scale circuits (up to twenty million nodes); (3) to demonstrate that PruneX-COG can not only achieve faster runtime but also improve the QoR; (4) to present a detailed ablation study of PruneX-COG.

Benchmarks We evaluate PruneX-COG on two widely-used public benchmarks EPFL (Amarú et al., 2015) and IWLS (Albrecht, 2005) and one industrial benchmark from an anonymous semiconductor company. These benchmarks consist of 69 circuits in total, including very large-scale circuits with up to twenty million nodes. Due to limited space, We defer more details to Appendix C.1.

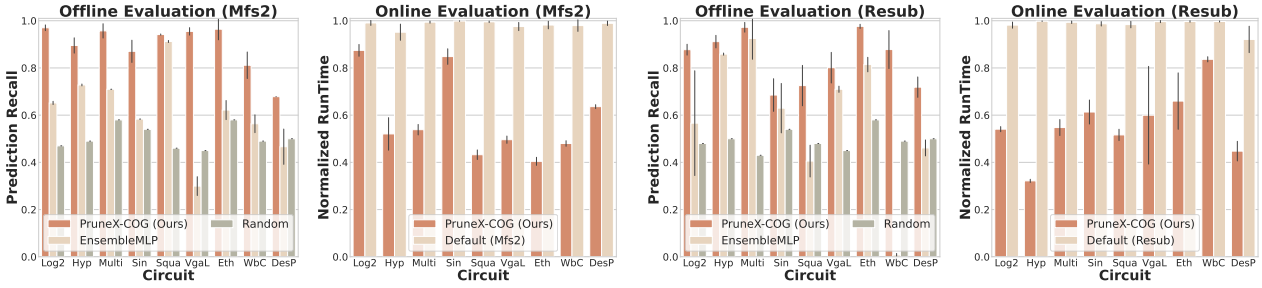
Experimental Setup Throughout all experiments, we use ABC as the backend LS framework, which is a state-of-the-art open source LS framework. In this paper, we apply PruneX to the Resub and Mfs2 heuristics to demonstrate that our method is applicable to many LS heuristics. We provide more details in Appendix E.1.

Evaluation Metrics and Evaluated Methods Throughout all experiments, we evaluate our method in two separate phases, i.e., the offline and online phases. **In the offline phase**, we evaluate the prediction recall of the effective nodes since the QoR improves with increased prediction recall (see Appendix B.2.4). Specifically, we present details as follows. (1) *Evaluation metrics* PruneX views the prediction task as a scoring task, and classify nodes with top k scores as positive samples. Then, we define a **top k accuracy metric** by the fraction of true positive nodes that are predicted to be positive, i.e., recall (see Appendix E.1.3). (2) *Evaluated methods* In the offline phase, we evaluate two baselines and our PruneX-COG. The baselines include Random, which randomly predicts a score between $[0, 1]$ for each node, and EnsembleMLP, which is our proposed simple learning-based baseline that uses a simple multi-layer perceptron to predict scores for nodes (see Appendix E.3). **In the online phase**, we evaluate the efficiency and QoR of PruneX-COG. (1) *Evaluation metrics* In terms of efficiency, we use the runtime metric. In terms of QoR, we mainly use the size, i.e., the number of nodes of optimized circuits, and depth, i.e. level of optimized circuits. Throughout all experiments, we found that our method achieves comparable optimization performance with the default heuristics in terms of the size and depth. Thus, we defer results in terms of the size and depth to Appendix B due to limited space. (2) *Evaluated methods* We evaluate the Default and PruneX-COG in the online phase. Default denotes the default heuristics, i.e., the Resub and Mfs2. PruneX-COG denotes new heuristics that apply our method to the Default heuristics. We set the top k hyperparameter as top 50% for all experiments unless mentioned otherwise.

Experiment 1. Evaluation on Open-Source Benchmarks In this subsection, we evaluate the offline prediction recall, online runtime, and online optimization performance on the open-source EPFL and IWLS benchmarks. Due to limited space, we defer more detailed results to Appendix B.5. Specifically, we design two evaluation strategies. **Evaluation Strategy 1: Generalization in Single Benchmark** Inspired by the leave-one-domain-out cross-validation strategy commonly used in previous literature (Wang et al., 2022b), we design nine leave-one-out datasets for evaluation. Specifically, given a benchmark, we construct a dataset by setting one circuit as the testing dataset, and the other circuits as the training dataset. Please refer to Appendix C.2 for more details. **Evaluation Strategy 2: Generalization from the IWLS to EPFL** In real industrial scenarios, we hope that the trained model can generalize to many unseen circuits. Thus, we design the second evaluation strategy. Specifically, we set the circuits from the IWLS as the training dataset, and the five hard-to-optimize circuits from the EPFL as the testing dataset. Due to limited space, *we defer results under the evaluation strategy 2 to Appendix B.5*.

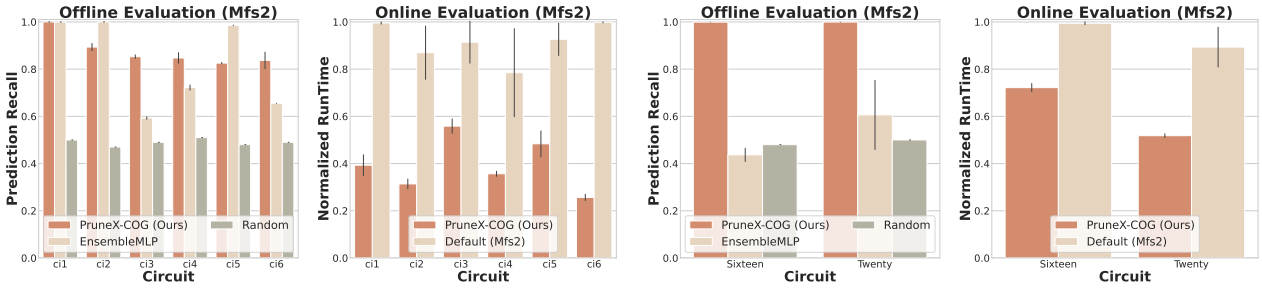
For the **offline evaluation** on the Mfs2 heuristic, Figure 5a shows that PruneX-COG significantly improves the prediction recall compared with the Random baseline in Evaluation Strategy 1. Specifically, PruneX-COG achieves 37% recall improvement on average. Furthermore, PruneX-COG achieves the prediction recall surpassing 90% on most testing circuits, indicating it can maintain applying most of the effective node-level transformations. Moreover, the results show that EnsembleMLP struggles to consistently achieve high prediction recall on all circuits, demonstrating that the OOD generalization problem across circuits in LS is challenging. For the **online evaluation** on the Mfs2 heuristic, Figure 5a shows that PruneX-COG significantly improves the runtime compared with the Default Mfs2 heuristic, achieving 40.96% improvement on average under the Evaluation Strategy 1. Moreover, PruneX-COG achieves marginal degradation in terms of the QoR (see Appendix B.5 for results). Overall, the results demonstrate that PruneX-COG significantly improves the efficiency of the Mfs2 heuristic while keeping comparable QoR.

For the evaluation on the Resub heuristic, Figure 5b shows



(a) Evaluation Strategy 1 on the Mfs2 heuristic. (b) Evaluation Strategy 1 on the Resub heuristic.

Figure 5. The results demonstrate that our PruneX-COG achieves significant prediction recall improvement (Left, ↑), runtime reduction (Middle, ↓), and marginal QoR (size) degradation (Right, ↓). The normalized runtime (node number) denotes the ratio of the runtime (node number) to that of applying the Default heuristic to testing circuits.



(a) Evaluation on the industrial benchmark. (b) Evaluation on the very large-scale circuits.

Figure 6. The results demonstrate the strong ability of our method to promote efficient LS on industrial and very large-scale circuits with the Mfs2 heuristic. In particular, PruneX-COG reduces the runtime of applying the Mfs2 heuristic once by up to 10.9 hours.

that PruneX-COG improves the recall by 33.7% and reduces the runtime by 42.67% on average (see Appendix B.5 for marginal size degradation results). In particular, PruneX-COG achieves 3.1× faster runtime on the Hyp circuit under Evaluation Strategy 1. The results suggest that our method is applicable to many LS heuristics, which can significantly improve their efficiency while keeping comparable QoR.

Experiment 2. Evaluation on Industrial and Large-Scale Circuits In this subsection, we further demonstrate the effectiveness of our method by deploying it to industrial circuits and very large-scale circuits from EPFL with the Mfs2 heuristic. Due to limited space, please refer to Appendix C.3 for more details of the datasets and experiment setting.

For the evaluation on industrial circuits, Fig. 6a shows that PruneX-COG significantly improves the prediction recall compared to the Random baseline and reduces the runtime compared to the Default Mfs2 heuristic. For the evaluation on very large-scale circuits from EPFL, Fig. 6b show that PruneX-COG significantly improves prediction recall while reducing the runtime by up to 42%. In particular, PruneX-COG reduces the runtime by up to 10.9 hours compared to the Default Mfs2. The results demonstrate the strong generalization ability and scalability of our PruneX-COG on industrial circuits and very large-scale circuits. We defer detailed results of the Resub heuristic to Appendix B.6.

Experiment 3. Improving Quality of Results with PruneX-COG In this subsection, we conduct experiments to demonstrate that efficient LS heuristics can improve QoR. To improve QoR, we can sequentially apply PruneX-COG multiple times rather than once, namely 2PruneX-COG. Specifically, we compare the runtime and QoR of 2PruneX-COG with the Default Mfs2 heuristic. Moreover, we set the hyperparameter k as 30% and 40% to achieve faster runtime. Due to limited space, we present the results of six representative large-scale circuits in Table 1. More details and results are provided in Appendix B.7. Table 1 shows that 2PruneX-COG significantly reduces the size and depth of optimized circuits while achieving faster runtime compared with the Default Mfs2 heuristic. Specifically, 2PruneX-COG with $k = 40\%$ reduces the size/depth by 7.14% on average while reducing the runtime by 2.21%. Furthermore, Table 1 shows that 2PruneX-COG with $k = 30\%$ reduces the size/depth by 6.88% on average with 25.98% runtime reduction. In particular, our method achieves a significant reduction over the depth on Hyp, improving the level by 30.23%. Overall, the results suggest that our PruneX-COG can significantly improve the QoR while achieving faster runtime.

Experiment 4. Ablation Study In this section, we conduct an ablation study to understand the individual contribution of each component within our PruneX-COG. To this end, we compare our PruneX-COG with PruneX-COG without DADC and PruneX-COG without DADC and KDSR on open-source benchmarks under the Evaluation Strategy 2.

	Hyp				Wb conmax				Des perf			
Method	Lev ↓	Improvement ↑ (Lev, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)
Default (Mfs2)	8259.00 (0.0)	NA	274.13 (9.90)	NA	16509.00 (0.0)	NA	21.24 (0.53)	NA	30853.00 (0.0)	NA	29.51 (0.28)	NA
2PruneX-COG (0.3, Ours)	5762.00 (0.0)	**30.23**	222.14 (27.98)	**18.97**	16111.00 (97.40)	**2.41**	13.45 (0.73)	**36.68**	29807.66 (13.69)	**3.39**	24.79 (0.14)	**15.99**
2PruneX-COG (0.4, Ours)	5762.00 (0.0)	**30.23**	288.85 (34.96)	**-5.37**	16006.66 (112.77)	**3.04**	16.74 (0.72)	**21.19**	29538.66 (11.61)	**4.26**	31.67 (0.15)	**-7.32**

	ci2				ci5				ci6			
Method	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)
Default (Mfs2)	195665.00 (0.0)	NA	487.51 (51.68)	NA	215708.00 (0.0)	NA	201.43 (12.17)	NA	99245.00 (0.0)	NA	76.97 (0.12)	NA
2PruneX-COG (0.3, Ours)	193618.00 (0.81)	**1.05**	264.32 (7.05)	**45.78**	215447.00 (10.03)	**0.12**	170.85 (10.19)	**15.18**	95210.00 (0.0)	**4.07**	59.03 (2.37)	**23.31**
2PruneX-COG (0.4, Ours)	193618.00 (0.81)	**1.05**	328.74 (4.89)	**32.57**	215443.66 (8.80)	**0.12**	308.35 (9.05)	-53.08	95144.50 (65.5)	**4.13**	57.51 (1.16)	**25.28**

Table 1. We compare the Default Mfs2 heuristic with our 2PruneX-COG heuristic with the hyperparameter k set as 30% and 40% on open-source and industrial circuits. Let Nd denote the node number (size) of circuits, and Lev denote the level (depth) of circuits. We define an Improvement metric by $\frac{M(\text{Default})-M(\text{2PruneX-COG})}{M(\text{Default})}$, where $M(\cdot)$ denotes the Nd, Lev, or Time.

heuristic	Mfs2			
Method/Circuit	Log2	Hyp	Multiplier	Sin
	Recall ↑	Recall ↑	Recall ↑	Recall ↑
PruneX-COG	**0.88 (0.02)**	**0.85 (0.06)**	**0.87 (0.04)**	**0.79 (0.12)**
PruneX-COG without DADC	0.87 (0.05)	**0.85 (0.01)**	0.79 (0.02)	0.75 (0.06)
PruneX-COG without DADC and KDSR	0.08 (0.007)	0.78 (0.01)	0.33 (0.01)	0.65 (0.08)
heuristic	Resub			
Method/Circuit	Log2	Hyp	Multiplier	Sin
	Recall ↑	Recall ↑	Recall ↑	Recall ↑
PruneX-COG	**0.80 (0.005)**	**0.87 (0.03)**	**0.87 (0.0)**	**0.81 (0.03)**
PruneX-COG without DADC	0.45 (0.23)	0.72 (0.17)	0.41 (0.31)	0.46 (0.23)
PruneX-COG without DADC and KDSR	0.21 (0.02)	0.67 (0.12)	0.71 (0.06)	0.28 (0.0)

Table 2. We present the ablation study results

We defer additional details and results to Appendix B.1. Table 2 suggests the following two conclusions. First, PruneX-COG without DADC significantly outperforms PruneX-COG without DADC and KDSR in terms of offline prediction recall, demonstrating the importance of our KDSR module. That is, the results suggest that our representation module effectively learns domain-invariant representations for high generalization capability. Second, PruneX-COG also significantly outperforms PruneX-COG without DADC in terms of offline prediction recall. This demonstrates that it is important to learn domain-aware classifiers for further improving the generalization capability of our method.

6. Conclusion

In this paper, we found an important problem that a large number of node-level transformations in many LS heuristics are *ineffective*, making applying these heuristics highly time-consuming. To address this challenge, we propose a data-driven LS heuristic paradigm, namely PruneX. The major technical contribution of PruneX is the novel **c**ircuit d**o**main **g**eneralization (COG) framework, which is critical for the success of data-driven LS algorithms. Extensive experiments demonstrate that PruneX significantly improves the efficiency of LS heuristics, achieving up to $3.1\times$ faster runtime while keeping comparable optimization performance.

Acknowledgements

This work was supported in part by National Key R&D Program of China under contract 2022ZD0119801, and National Nature Science Foundations of China grants U23A20388, 62021001, U19B2026, U19B2044, and 623B1022. This work was supported in part by Huawei. We would like to thank all the anonymous reviewers for their insightful comments.

Impact Statement

This paper presents a novel data-driven logic synthesis heuristic paradigm whose goal is to promote efficient logic synthesis in chip design. This work takes the first step towards investigating the out-of-distribution generalization problem in logic synthesis. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agnesina, A., Rajvanshi, P., Yang, T., Pradipta, G., Jiao, A., Keller, B., Khailany, B., and Ren, H. Autodmp: Automated dreamplace-based macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, ISPD '23, pp. 149–157, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399784. doi: 10.1145/3569052.3578923. URL https://doi.org/10.1145/3569052.3578923.

Albrecht, C. Iwls 2005 benchmarks. 2005.

Amarú, L., Gaillardon, P.-E., and De Micheli, G. The epfl combinational benchmark suite. (CONF), 2015.

Bartlett, P. L. and Mendelson, S. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Berndt, A. A. S., Fogaça, M., and Meinhardt, C. A review of machine learning in logic synthesis. *Journal of Integrated Circuits and Systems*, 17(3):1–12, 2022.

Bertacco, V. and Damiani, M. The disjunctive decomposition of logic functions. In *iccad*, volume 97, pp. 78–82, 1997.

Blanchard, G., Lee, G., and Scott, C. Generalizing from several related classification tasks to a new unlabeled sample. *Advances in neural information processing systems*, 24, 2011.

Brayton, A. M. R. Scalable logic synthesis using a simple circuit structure. 6:15–22, 2006.

Brayton, A. M. S. C. R. and Kam, X. W. T. Technology mapping with boolean matching, supergates and choices.

Brayton, R. and Mishchenko, A. Abc: An academic industrial-strength verification tool. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pp. 24–40. Springer, 2010.

Brayton, R. K. The decomposition and factorization of boolean expressions. *ISCA-82*, pp. 49–54, 1982.

Cadence. Cadence cerebrus, 2021. URL https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/cerebrus-intelligent-chip-explorer.html.

Chen, S., Li, S., Zhuang, Z., Zheng, S., Liang, Z., Ho, T.-Y., Yu, B., and Sangiovanni-Vincentelli, A. L. Floorplet: Performance-aware floorplan framework for chiplet integration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023a.

Chen, S., Zheng, S., Bai, C., Zhao, W., Yin, S., Bai, Y., and Yu, B. Soc-tuner: An importance-guided exploration framework for dnn-targeting soc design. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 207–212. IEEE, 2024.

Chen, Y., Zhang, Y., Bian, Y., Yang, H., Kaili, M., Xie, B., Liu, T., Han, B., and Cheng, J. Learning causally invariant representations for out-of-distribution generalization on graphs. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22131–22148. Curran Associates, Inc., 2022.

Chen, Y., Zhou, K., Bian, Y., Xie, B., Wu, B., Zhang, Y., KAILI, M., Yang, H., Zhao, P., Han, B., and Cheng, J. Pareto invariant risk minimization: Towards mitigating the optimization dilemma in out-of-distribution generalization. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=esFxSb_0pSL.

Cheng, R., Lyu, X., Li, Y., Ye, J., HAO, J., and Yan, J. The policy-gradient placement and generative routing neural networks for chip design. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=uNYqDfPEDD8.

De Abreu, B. A., Berndt, A., Campos, I. S., Meinhardt, C., Carvalho, J. T., Grellert, M., and Bampi, S. Fast logic optimization using decision trees. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. IEEE, 2021.

Djukić, D., Janković, V., Matić, I., and Petrović, N. *The IMO Compendium: A Collection of Problems Suggested for the International Mathematical Olympiads: 1959-2009 Second Edition*. Springer, 2011.

Farrahi, A. H. and Sarrafzadeh, M. Complexity of the lookup-table minimization problem for fpga technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(11):1319–1332, 1994.

Fawcett, B. Synthesis for fpgas: an overview. *Proceedings of WESCON'94*, pp. 576–580, 1994.

Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Geng, Z., Xie, S., Xia, Y., Wu, L., Qin, T., Wang, J., Zhang, Y., Wu, F., and Liu, T.-Y. De novo molecular generation via connection-aware motif mining. In *The Eleventh International Conference on Learning Representations*, 2022.

Geng, Z., Li, X., Wang, J., Li, X., Zhang, Y., and Wu, F. A deep instance generative framework for milp solvers under limited data availability. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 26025–26047. Curran Associates, Inc., 2023.

Geng, Z., Wang, J., Liu, Z., Xu, S., Tang, Z., Yuan, M., Hao, J., Zhang, Y., and Wu, F. Reinforcement learning within tree search for fast macro placement. In *International Conference on Machine Learning*. PMLR, 2024.

Ghose, A., Zhang, V., Zhang, Y., Li, D., Liu, W., and Coates, M. Generalizable cross-graph embedding for gnn-based congestion prediction. In *2021 IEEE/ACM International Conference On Computer Aided Design*

(ICCAD), pp. 1–9. IEEE Press, 2021. doi: 10.1109/ICCAD51958.2021.9643446. URL https://doi.org/10.1109/ICCAD51958.2021.9643446.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Grosnit, A., Malherbe, C., Tutunov, R., Wan, X., Wang, J., and Ammar, H. B. Boils: Bayesian optimisation for logic synthesis. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1193–1196. IEEE, 2022.

Gulrajani, I. and Lopez-Paz, D. In search of lost domain generalization. In *International Conference on Learning Representations*.

Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

Hosny, A., Hashemi, S., Shalan, M., and Reda, S. Drills: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 581–586. IEEE, 2020a.

Hosny, A., Hashemi, S., Shalan, M., and Reda, S. Drills: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 581–586. IEEE, 2020b.

Huang, G., Hu, J., He, Y., Liu, J., Ma, M., Shen, Z., Wu, J., Xu, Y., Zhang, H., Zhong, K., et al. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(5):1–46, 2021.

Jiang, J.-H., Jiang, Y., Li, Y., Mishchenko, A., Sinha, S., Villa, T., Brayton, R., and Parades, R. I. Mvsis v1. 1 manual.

Khailany, B. Accelerating chip design with machine learning. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, pp. 33–33, 2020.

Kim, R. G., Doppa, J. R., and Pande, P. P. Machine learning for design space exploration and optimization of manycore systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, 2018. doi: 10.1145/3240765.3243483.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kirby, R., Godil, S., Roy, R., and Catanzaro, B. Congestionnet: Routing congestion prediction using deep graph neural networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 217–222, 2019a. doi: 10.1109/VLSI-SoC.2019.8920342.

Kirby, R., Godil, S., Roy, R., and Catanzaro, B. Congestionnet: Routing congestion prediction using deep graph neural networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 217–222. IEEE, 2019b.

Lai, Y., Mu, Y., and Luo, P. Maskplace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems*, 35:24019–24030, 2022.

Lai, Y., Liu, J., Tang, Z., Wang, B., Hao, J., and Luo, P. Chipformer: Transferable chip placement via offline decision transformer. In *International Conference on Machine Learning*, pp. 18346–18364. PMLR, 2023.

Li, X., Chen, L., Zhang, J., Wen, S., Sheng, W., Huang, Y., and Yuan, M. Effisyn: Efficient logic synthesis with dynamic scoring and pruning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023a. doi: 10.1109/ICCAD57390.2023.10323902.

Li, X., Li, X., Chen, L., Zhang, X., Yuan, M., and Wang, J. Circuit transformer: End-to-end circuit design by predicting the next gate, 2024a.

Li, X., Zhu, F., Zhen, H.-L., Luo, W., Lu, M., Huang, Y., Fan, Z., Zhou, Z., Kuang, Y., Wang, Z., Geng, Z., Li, Y., Liu, H., An, Z., Yang, M., Li, J., Wang, J., Yan, J., Sun, D., Zhong, T., Zhang, Y., Zeng, J., Yuan, M., Hao, J., Yao, J., and Mao, K. Machine learning insides optverse ai solver: Design principles and applications, 2024b.

Li, Y., Shen, Y., Chen, L., and Yuan, M. Orca: Scalable temporal graph neural network training with theoretical guarantees. *Proceedings of the ACM on Management of Data*, 1(1):1–27, 2023b.

Li, Y., Shen, Y., Chen, L., and Yuan, M. Zebra: When temporal graph neural networks meet temporal personalized pagerank. *Proceedings of the VLDB Endowment*, 16(6): 1332–1345, 2023c.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Ling, H., Wang, Z., and Wang, J. Learning to stop cut generation for efficient mixed-integer linear programming.

Proceedings of the AAAI Conference on Artificial Intelligence, 38(18):20759–20767, Mar. 2024. doi: 10.1609/aaai.v38i18.30064. URL https://ojs.aaai.org/index.php/AAAI/article/view/30064.

Liu, H.-Y. and Carloni, L. P. On learning-based methods for design-space exploration with high-level synthesis. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2013.

Liu, J., Rizzi, C., and Josipović, L. Load-store queue sizing for efficient dataflow circuits. In *2022 International Conference on Field-Programmable Technology (ICFPT)*, pp. 1–9, 2022. doi: 10.1109/ICFPT56656.2022.9974425.

Liu, J., Ni, L., Li, X., Zhou, M., Chen, L., Li, X., Zhao, Q., and Ma, S. Aimap: Learning to improve technology mapping for asics via delay prediction. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*, pp. 344–347. IEEE, 2023a.

Liu, J., Zhou, M., Ma, S., and Pan, L. Mata*: Combining learnable node matching with a* algorithm for approximate graph edit distance computation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 1503–1512, 2023b.

Lopera, D. S., Servadei, L., Kiprit, G. N., Hazra, S., Wille, R., and Ecker, W. A survey of graph neural networks for electronic design automation. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6. IEEE, 2021.

Makrani, H. M., Sayadi, H., Mohsenin, T., rafatirad, S., Sasan, A., and Homayoun, H. Xppe: Cross-platform performance estimation of hardware accelerators using machine learning. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, pp. 727–732, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450360074. doi: 10.1145/3287624.3288756. URL https://doi.org/10.1145/3287624.3288756.

McDiarmid, C. et al. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

Micheli, G. D. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Mishchenko, A., Chatterjee, S., and Brayton, R. Dag-aware aig rewriting a fresh look at combinational logic synthesis. In *Proceedings of the 43rd annual Design Automation Conference*, pp. 532–535, 12 2006.

Mishchenko, A., Cho, S., Chatterjee, S., and Brayton, R. Combinational and sequential mapping with priority cuts. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pp. 354–361. IEEE, 2007.

Mishchenko, A., Brayton, R., Jiang, J.-H. R., and Jang, S. Scalable don't-care-based logic optimization and resynthesis. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 4(4):1–23, 2011.

Muandet, K., Balduzzi, D., and Schölkopf, B. Domain generalization via invariant feature representation. In *International conference on machine learning*, pp. 10–18. PMLR, 2013.

Neto, W. L., Austin, M., Temple, S., Amaru, L., Tang, X., and Gaillardon, P.-E. Lsoracle: A logic synthesis framework driven by artificial intelligence. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6. IEEE, 2019a.

Neto, W. L., Tang, X., Austin, M., Amaru, L., and Gaillardon, P.-E. Improving logic optimization in sequential circuits using majority-inverter graphs. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 224–229. IEEE, 2019b.

Neto, W. L., Moreira, M. T., Amaru, L., Yu, C., and Gaillardon, P.-E. Read your circuit: leveraging word embedding to guide logic optimization. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 530–535, 2021a.

Neto, W. L., Moreira, M. T., Li, Y., Amarù, L., Yu, C., and Gaillardon, P.-E. Slap: a supervised learning approach for priority cuts technology mapping. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 859–864. IEEE, 2021b.

Pasandi, G., Pratty, S., and Forsyth, J. Aisyn: Ai-driven reinforcement learning-based logic synthesis framework. *arXiv preprint arXiv:2302.06415*, 2023.

Pinelis, I. F. and Sakhanenko, A. I. Remarks on inequalities for large deviation probabilities. *Theory of Probability & Its Applications*, 30(1):143–148, 1986.

Reddy, B. K., Sabbavarapu, S., and Acharyya, A. A new vlsi ic design automation methodology with reduced nre costs and time-to-market using the npn class representation and functional symmetry. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 177–180. IEEE, 2014.

Ren, H. and Hu, J. *Machine Learning Applications in Electronic Design Automation*. Springer Nature, 2023.

Rota Bulo, S., Neuhold, G., and Kontschieder, P. Loss max-pooling for semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2126–2135, 2017.

Sabbavarapu, S., Basireddy, K. R., and Acharyya, A. A new dynamic library based ic design automation methodology using functional symmetry with npn class representation approach to reduce nre costs and time-to-market. In *2014 Fifth International Symposium on Electronic System Design*, pp. 115–119. IEEE, 2014.

Sánchez, D., Servadei, L., Kiprit, G. N., Wille, R., and Ecker, W. A comprehensive survey on electronic design automation and graph neural networks: Theory and applications. *ACM Trans. Des. Autom. Electron. Syst.*, 28(2), feb 2023. ISSN 1084-4309. doi: 10.1145/3543853. URL https://doi.org/10.1145/3543853.

Shen, Z., Liu, J., He, Y., Zhang, X., Xu, R., Yu, H., and Cui, P. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*, 2021.

Shi, Z., Liang, X., and Wang, J. Lmc: Fast training of gnns via subgraph sampling with provable convergence. In *The Eleventh International Conference on Learning Representations*, 2023.

Synopsys. Design space optimization ai, 2020. URL https://www.synopsys.com/ai/chip-design/dso-ai.html.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Wang, B., Shen, G., Li, D., Hao, J., Liu, W., Huang, Y., Wu, H., Lin, Y., Chen, G., and Heng, P. A. Lhnn: Lattice hypergraph neural network for vlsi congestion prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, pp. 1297–1302, New York, NY, USA, 2022a. Association for Computing Machinery. ISBN 9781450391429. doi: 10.1145/3489517.3530675. URL https://doi.org/10.1145/3489517.3530675.

Wang, C. and Yu, T. Efficient training of multi-task neural solver with multi-armed bandits. *arXiv preprint arXiv:2305.06361*, 2023.

Wang, C., Yang, Y., Han, C., Guo, T., Zhang, H., and Wang, J. A game-theoretic approach for improving generalization ability of tsp solvers. 2021.

Wang, C., Han, C., Guo, T., and Ding, M. Solving uncapacitated p-median problem with reinforcement learning assisted by graph attention networks. *Applied Intelligence*, 53(2):2010–2025, 2023a.

Wang, C., Yu, Z., McAleer, S., Yu, T., and Yang, Y. Asp: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024a.

Wang, J., Lan, C., Liu, C., Ouyang, Y., Qin, T., Lu, W., Chen, Y., Zeng, W., and Yu, P. Generalizing to unseen domains: A survey on domain generalization. *IEEE Transactions on Knowledge and Data Engineering*, 2022b.

Wang, J., Wang, Z., Li, X., Kuang, Y., Shi, Z., Zhu, F., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning to cut via hierarchical sequence/set model for efficient mixed-integer programming, 2024b.

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=Zob4P9bRNcK.

Xie, B., Bian, Y., Zhou, K., Chen, Y., Zhao, P., Han, B., Meng, W., and Cheng, J. Enhancing neural subset selection: Integrating background information into set representations. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=eepoE7iLpL.

Xie, B., Chen, Y., Wang, J., Zhou, K., Han, B., Meng, W., and Cheng, J. Enhancing evolving domain generalization through dynamic latent representations. In Wooldridge, M. J., Dy, J. G., and Natarajan, S. (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 16040–16048. AAAI Press, 2024b. doi: 10.1609/AAAI.V38I14.29536. URL https://doi.org/10.1609/aaai.v38i14.29536.

Yang, S., Yang, Z., Li, D., Zhang, Y., Zhang, Z., Song, G., and HAO, J. Versatile multi-stage graph neural network for circuit representation. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=nax3ATLrovW.

Yao, X., Li, H., Chan, T. H., Xiao, W., Yuan, M., Huang, Y., Chen, L., and Yu, B. Hdldebugger: Streamlining hdl debugging with large language models. *arXiv preprint arXiv:2403.11671*, 2024.

Yuan, J., Wang, P., Ye, J., Yuan, M., Hao, J., and Yan, J. Easyso: Exploration-enhanced reinforcement learning for logic synthesis sequence optimization and a comprehensive rl environment. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

Zhou, Y., Ren, H., Zhang, Y., Keller, B., Khailany, B., and Zhang, Z. Primal: Power inference using machine learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

Zhou, Z.-H. and Zhou, Z.-H. *Ensemble learning*. Springer, 2021.

Zhu, X., Tang, R., Chen, L., Li, X., Huang, X., Yuan, M., Sheng, W., and Xu, J. A database dependent framework for k-input maximum fanout-free window rewriting. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023. doi: 10.1109/DAC56929.2023.10247727.

Zuo, D., Ouyang, Y., and Ma, Y. Rl-mul: Multiplier design optimization with deep reinforcement learning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2023.

A. Theoretical Analysis

A.1. Proof of Theorem 4.2

The proof of Theorem 4.2 draws inspiration from the methodology proposed by (Blanchard et al., 2011; Muandet et al., 2013). Nevertheless, our problem setting is different from that of (Blanchard et al., 2011; Muandet et al., 2013). Specifically, they assume equal values of n_k for $k = 1, 2, \cdots, M$ as a prior. In contrast, our setting allows for flexibility in these values as we propose a circuit aggregation mechanism g. Consequently, our conclusion holds a greater level of generality.

To complete the proof, we utilize the kernel method and introduce the concept of reproducing kernel Hilbert space (RKHS) $\mathcal{H}_{\bar{K}}$ with kernel:

$$\bar{K}\left(\left(P_x^1, \mathbf{x}_1\right), \left(P_x^2, \mathbf{x}_2\right)\right) = K_{\mathcal{P}}\left(P_x^1, P_x^2\right) K_{\mathcal{X}}\left(\mathbf{x}_1, \mathbf{x}_2\right)$$

Here, $K_{\mathcal{P}}$ and $K_{\mathcal{X}}$ represent kernel functions on distributions and input space, respectively. Suppose the RKHS corresponding to K (K can be any kernel function) is $\mathcal{H}_K$. We consider the feature mapping (Blanchard et al., 2011) $\Psi : \mathfrak{P}_{\mathcal{X}} \to \mathcal{H}_{K_{\mathcal{X}}}$:

$$P_x \to \Psi(P_x) := \int_{\mathcal{X}} K_{\mathcal{X}}(\mathbf{x}, \cdot) dP_X(\mathbf{x})$$

a universal kernel κ (Blanchard et al., 2011) on $\mathcal{H}_{K_{\mathcal{X}}}$ which satisfies:

$$K_{\mathcal{P}}(P_x^1, P_x^2) = \kappa\left(\Psi(P_x^1), \Psi(P_x^2)\right)$$

and the mapping $\Phi_\kappa : \mathcal{H}_{K_{\mathcal{X}}} \to \mathcal{H}_\kappa$ which satisfies:

$$\kappa\left(\Psi(P_x^1), \Psi(P_x^2)\right) = \langle \Phi_\kappa(\Psi(P_X^1)), \Phi_\kappa(\Psi(P_X^2)) \rangle$$

Notations Let $\mathcal{X}$ denote the input space and $\mathcal{Y} = \{0, 1\}$ the output space. Let $\mathfrak{P}_{\mathcal{X}}$ denote the set of probability distributions on $\mathcal{X}$. A decision function is a function $f : \mathfrak{P}_{\mathcal{X}} \times \mathcal{X} \to \mathcal{Y}$. The loss function has the form $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. We consider a scenario where iid training distributions $\mathbb{P}_X^k$ and test distribution $\mathbb{P}_X^t$ are drawn according to a hyper-distribution $\mathscr{P}$. $\hat{\mathbb{P}}_X^k$ denote the empirical estimation of $\mathbb{P}_X^k$.

Remark Recall that M is the number of training domains, n_k is the sample size of the k-th domain, and $n = \sum_{k=1}^{M} n_k$ is the total number of samples.

Assumptions (1) The loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is ϕ_X-Lipschitz in its first variable and bounded by U_l. (2) The kernel $K_{\mathcal{X}}$ and κ are bounded by U_K^2 and U_κ^2, respectively. (3) The feature map $\Phi_\kappa : \mathcal{H}_{K_{\mathcal{X}}} \to \mathcal{H}_\kappa$ satisfies a specific Hölder condition with constant L_κ on $B_{K_{\mathcal{X}}}(U_K)$:

$$\forall v, w \in B_{K_{\mathcal{X}}}(U_K) : ||\Phi_\kappa(v) - \Phi_\kappa(w)|| \leq L_\kappa ||v - w||$$

Under the aforementioned assumptions, we present the following proof.

Proof. Based on the inequality $|a_1 + \cdots + a_n|^2 \leq n|a_1|^2 + \cdots + n|a_2|^2$ and $\sup(A + B) \leq \sup A + \sup B$, we decompose

$$\begin{aligned}
&\left(\sup_{f \in B_{\bar{K}}(r)} |\mathcal{R}(f) - \hat{\mathcal{R}}(f)|\right)^2 \\
=&\left(\sup_{f \in B_{\bar{K}}(r)} \left| \mathbb{E}_{\mathbb{P}_{XY}^t \sim \mathscr{P}} \mathbb{E}_{(\mathbf{x},y) \sim \mathbb{P}_{XY}^t} [l(f(\mathbb{P}_X^t, \mathbf{x}), y)] - \frac{1}{M} \sum_{k=1}^{M} \frac{1}{n_k} \sum_{j=1}^{n_k} l(f(\hat{\mathbb{P}}_X^k, \mathbf{x}_j^k), y_j^k) \right| \right)^2 \\
\leq& 2 \left(\sup_{f \in B_{\bar{K}}(r)} \left| \mathbb{E}_{\mathbb{P}_{XY}^t \sim \mathscr{P}} \mathbb{E}_{(\mathbf{x},y) \sim \mathbb{P}_{XY}^t} [l(f(\mathbb{P}_X^t, \mathbf{x}), y)] - \frac{1}{M} \sum_{k=1}^{M} \mathbb{E}_{(\mathbf{x},y) \sim P_{XY}^k} [l(f(\mathbb{P}_X^k, \mathbf{x}), y)] \right| \right)^2 \\
+& 2 \left(\sup_{f \in B_{\bar{K}}(r)} \left| \frac{1}{M} \sum_{k=1}^{M} \mathbb{E}_{(\mathbf{x},y) \sim \mathbb{P}_{XY}^k} [l(f(\mathbb{P}_X^k, \mathbf{x}), y)] - \frac{1}{M} \sum_{k=1}^{M} \frac{1}{n_k} \sum_{j=1}^{n_k} l(f(\hat{\mathbb{P}}_X^k, \mathbf{x}_j^k), y_j^k) \right| \right)^2 \\
=&(I) + (II)
\end{aligned}$$

Control of I

$$(I) = 2\left(\sup_{f\in B_{\bar{K}}(r)}\left|\mathbb{E}_{\mathbb{P}^t_{XY}\sim\mathscr{P}}\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^t_{XY}}[l(f(\mathbb{P}^t_X,\mathbf{x}),y)] - \frac{1}{M}\sum_{k=1}^{M}\mathbb{E}_{(\mathbf{x},y)\sim P^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)]\right|\right)^2$$

Since the $(\mathbb{P}^k_{XY})_{1\le k\le M}$ are iid, so we let

$$\beta((\mathbb{P}^k_{XY})_{1\le k\le M}) := \sup_{f\in B_{\bar{K}}(r)}\left|\mathbb{E}_{\mathbb{P}^t_{XY}\sim\mathscr{P}}\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^t_{XY}}[l(f(\mathbb{P}^t_X,\mathbf{x}),y)] - \frac{1}{M}\sum_{k=1}^{M}\mathbb{E}_{(\mathbf{x},y)\sim P^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)]\right|$$

By McDiarmid inequality (McDiarmid et al., 1989) in Hilbert space, the inequality holds with probability $1-\delta$

$$\beta - \mathbb{E}[\beta] \le U_l\sqrt{\frac{\log\delta^{-1}}{2M}}$$

where $\mathbb{E}[\beta]$ is denoted as

$$\mathbb{E}[\beta] = \mathbb{E}_{(\mathbb{P}^k_{XY})_{1\le k\le M}}\sup_{f\in B_{\bar{K}}(r)}\left|\mathbb{E}_{\mathbb{P}^t_{XY}\sim\mathscr{P}}\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^t_{XY}}[l(f(\mathbb{P}^t_X,\mathbf{x}),y)] - \frac{1}{M}\sum_{k=1}^{M}\mathbb{E}_{(\mathbf{x},y)\sim P^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)]\right|$$

To bound $\mathbb{E}[\beta]$, we use Rademacher complexity analysis. We denote $(\mathbf{x}_k,y_k)$ a single draw from distribution $\mathbb{P}^k_{XY}$ and these draws are independent. We also denote $(\sigma_k)_{1\le k\le M}$ $\{\pm1\}$-valued iid Rademacher variables which are independent from everything else. We have:

$$\begin{aligned}
\mathbb{E}[\beta] \le& \mathbb{E}_{(\mathbb{P}^k_{XY})_{1\le k\le M}}\mathbb{E}_{\sigma_k}\left[\sup_{f\in B_{\bar{K}}(r)}\left|\frac{2}{M}\sum_{k=1}^{M}\sigma_k\mathbb{E}_{(\mathbf{x}_k,y_k)\sim P^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}_k),y_k)]\right|\right]\\
\le& \mathbb{E}_{(\mathbb{P}^k_{XY})_{1\le k\le M}}\mathbb{E}_{(\mathbf{x}_k,y_k)\sim P^k_{XY}}\mathbb{E}_{\sigma_k}\left[\sup_{f\in B_{\bar{K}}(r)}\left|\frac{2}{M}\sum_{k=1}^{M}\sigma_k l(f(\mathbb{P}^k_X,\mathbf{x}_k),y_k)\right|\right]\\
\le& \frac{2r\phi_X U_K U_\kappa}{\sqrt{M}}
\end{aligned}$$

The first inequality is a standard symmetrization argument. The second inequality pulls the inner expectation on $(\mathbf{x}_k,y_k)$ outwards. The last inequality is a standard bound for the Rademacher complexity of a Lipschitz loss function on the ball of radius r of $\mathcal{H}_{\bar{K}}$ (Bartlett & Mendelson, 2002), where the kernel $\bar{K}$ is bounded by $U_K^2U_\kappa^2$. Based on the above analysis, we can conclude that

$$\begin{aligned}
(I) =& 2\beta^2\\
\le& 4(\mathbb{E}[\beta]^2 + \frac{U_l^2\log\delta^{-1}}{2M})\\
\le& \frac{16r^2\phi_X^2U_K^2U_\kappa^2}{M} + \frac{2U_l^2\log\delta^{-1}}{M}
\end{aligned}$$

Control of II

$$\begin{aligned}
(II) =& 2\left(\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)] - \frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}l(f(\hat{\mathbb{P}}^k_X,\mathbf{x}^k_j),y^k_j)\right|\right)^2\\
\le& 4\left(\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)] - \frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j)\right|\right)^2\\
&+4\left(\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j) - \frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}l(f(\hat{\mathbb{P}}^k_X,\mathbf{x}^k_j),y^k_j)\right|\right)^2\\
=& (IIa) + (IIb)
\end{aligned}$$

Control of IIa We investigate term (IIa) conditional to $(\mathbb{P}^k_{XY})_{1\leq k\leq M}$. Under this conditional distribution, it is noteworthy that the variables $(\mathbf{x}^k_j, y^k_j)_{jk}$ are now independent. We let

$$\xi((x^k_j, y^k_j)_{jk}) = \sup_{f\in B_{\bar{K}}(r)} \left| \frac{1}{M}\sum_{k=1}^{M} \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)] - \frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k} l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j)\right|$$

based on the McDiarmid inequality (McDiarmid et al., 1989), we deduce that with probability $1-\delta$ over the draw of $(\mathbf{x}^k_j, y^k_j)_{jk}$, it holds

$$\xi - \mathbb{E}[\xi|(\mathbb{P}^k_{XY})_{1\leq k\leq M}] \leq \frac{U_l}{M}\sqrt{\sum_{k=1}^{M}\frac{\log\delta^{-1}}{n_k}}$$

To bound $\mathbb{E}[\xi|(\mathbb{P}^k_{XY})_{1\leq k\leq M}]$, we can use relatively standard Rademacher complexity analysis. Denoting $(\sigma_{kj})_{1\leq k\leq M, 1\leq j\leq n_k}$ i.i.d Rademacher variables (Bartlett & Mendelson, 2002), we have

$$\begin{aligned}
\mathbb{E}[\xi|(\mathbb{P}^k_{XY})_{1\leq k\leq M}] &= \mathbb{E}_{(\mathbf{x}^k_j,y^k_j)}\left[\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}\left(\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}^k_{XY}}[l(f(\mathbb{P}^k_X,\mathbf{x}),y)] - l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j)\right)\right| \middle| (\mathbb{P}^k_{XY})_{1\leq k\leq M}\right] \\
&\leq \mathbb{E}_{(\mathbf{x}^k_j,y^k_j)}\mathbb{E}_{\sigma_{kj}}\left[\sup_{f\in B_{\bar{K}}(r)}\left|\frac{2}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}\sigma_{kj}l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j)\right| \middle| (\mathbb{P}^k_{XY})_{1\leq k\leq M}\right] \\
&\leq \frac{2r\phi_X U_K U_\kappa}{M}\sqrt{\sum_{k=1}^{M}\frac{1}{n_k}}
\end{aligned}$$

Therefore, we can conclude that

$$\begin{aligned}
(IIa) &= 4\xi^2 \\
&\leq 8(\mathbb{E}[\xi]^2 + \frac{U_l^2\log\delta^{-1}}{M^2}\sum_{k=1}^{M}\frac{1}{n_k}) \\
&\leq \frac{32r^2\phi_X^2U_K^2U_\kappa^2 + 8U_l^2\log\delta^{-1}}{M^2}\sum_{k=1}^{M}\frac{1}{n_k}
\end{aligned}$$

Control of IIb

$$\begin{aligned}
(IIb) =& 4\left(\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k} l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j) - \frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k} l(f(\hat{\mathbb{P}}^k_X,\mathbf{x}^k_j),y^k_j)\right|\right)^2 \\
=& 4\left(\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}[l(f(\mathbb{P}^k_X,\mathbf{x}^k_j),y^k_j) - l(f(\hat{\mathbb{P}}^k_X,\mathbf{x}^k_j),y^k_j)]\right|\right)^2 \\
\leq& 4\phi_X^2\left(\sup_{f\in B_{\bar{K}}(r)}\left|\frac{1}{M}\sum_{k=1}^{M}\frac{1}{n_k}\sum_{j=1}^{n_k}[f(\mathbb{P}^k_X,\mathbf{x}^k_j) - f(\hat{\mathbb{P}}^k_X,\mathbf{x}^k_j)]\right|\right)^2 \\
\leq& 4\phi_X^2\left(\sup_{f\in B_{\bar{K}}(r)}\frac{1}{M}\sum_{k=1}^{M}||f(\mathbb{P}^k_X,\cdot) - f(\hat{\mathbb{P}}^k_X,\cdot)||_\infty\right)^2
\end{aligned}$$

Here $f(P^k_X,\cdot)$ denotes a vector in Hilbert space and $||\cdot||_\infty$ is the infinite norm of the vector. Using the reproducing property

of $\bar{K}$ and the condition that $||\Psi(\mathbb{P}_X)|| \leq U_K$, we have for any $\mathbf{x} \in \mathcal{X}$ and $f \in B_{\bar{K}}(r)$

$$
\begin{aligned}
&|f(\mathbb{P}_X^k, \mathbf{x}) - f(\hat{\mathbb{P}}_X^k, \mathbf{x})| \\
=&|\langle \bar{K}((\mathbb{P}_X^k, \mathbf{x}), \cdot) - \bar{K}((\hat{\mathbb{P}}_X^k, \mathbf{x}), \cdot), f\rangle| \\
\leq&||f|| \cdot ||\bar{K}((\mathbb{P}_X^k, \mathbf{x}), \cdot) - \bar{K}((\hat{\mathbb{P}}_X^k, \mathbf{x}), \cdot)|| \\
\leq&rK_{\mathcal{X}}(\mathbf{x}, \mathbf{x})^{\frac{1}{2}} \cdot \left(\kappa(\Psi(\mathbb{P}_X^k), \Psi(\mathbb{P}_X^k)) + \kappa(\Psi(\hat{\mathbb{P}}_X^k), \Psi(\hat{\mathbb{P}}_X^k)) - 2\kappa(\Psi(\mathbb{P}_X^k), \Psi(\hat{\mathbb{P}}_X^k))\right)^{\frac{1}{2}} \\
\leq&rU_K||\Phi_\kappa(\Psi(\mathbb{P}_X^k)) - \Phi_\kappa(\Psi(\hat{\mathbb{P}}_X^k))|| \\
\leq&rU_K L_\kappa||\Psi(\mathbb{P}_X^k) - \Psi(\hat{\mathbb{P}}_X^k)||
\end{aligned}
$$

By Hoeffding's inequality in Hilbert space (Pinelis & Sakhanenko, 1986), we can conclude that with probability $1 - \delta$

$$
\begin{aligned}
||\Psi(\hat{\mathbb{P}}_X^k) - \Psi(\mathbb{P}_X^k)|| &= \left|\left|\frac{1}{n_k}\sum_{j=1}^{n_k} K_{\mathcal{X}}(\mathbf{x}_j^k, \cdot) - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_X^k}[K_{\mathcal{X}}(\mathbf{x}, \cdot)]\right|\right| \\
&\leq 3U_K\sqrt{\frac{\log 2\delta^{-1}}{n_k}}
\end{aligned}
$$

Therefore, we have

$$
\begin{aligned}
(IIb) &\leq 4\phi_X^2 r^2 U_K^2 L_\kappa^2 \left(\frac{1}{M}\sum_{k=1}^{M}||\Psi(\mathbb{P}_X^k) - \Psi(\hat{\mathbb{P}}_X^k)||\right)^2 \\
&\leq 4\phi_X^2 r^2 U_K^2 L_\kappa^2 \frac{1}{M}\sum_{k=1}^{M}||\Psi(\mathbb{P}_X^k) - \Psi(\hat{\mathbb{P}}_X^k)||^2 \\
&\leq 36\phi_X^2 r^2 U_K^4 L_\kappa^2 \frac{1}{M}\sum_{k=1}^{M}\frac{\log 2\delta^{-1}}{n_k}
\end{aligned}
$$

Combining all of the above inequalities, we obtain the announced results of the theorem 4.2 that with probability at least $1 - \delta$

$$
\left(\sup_{f\in B_{\bar{K}}(r)}|\mathcal{R}(f) - \hat{\mathcal{R}}(f)|\right)^2 \leq \frac{C_1\log\delta^{-1} + C_2}{M} + \frac{C_3\log 2\delta^{-1}M + C_4\log\delta^{-1} + C_5}{M^2}\sum_{k=1}^{M}\frac{1}{n_k}. \tag{2}
$$

where $C_1 = 2U_l^2$, $C_2 = 16r^2\phi_X^2 U_K^2 U_\kappa^2$, $C_3 = 36\phi_X^2 r^2 U_K^4 L_\kappa^2$, $C_4 = 8U_l^2$, $C_5 = 32r^2\phi_X^2 U_K^2 U_\kappa^2$. □

Discusson on the generalization error bound The inequality 2 states that the generalization error can be controlled by a training circuit domain number and sample size trade-off. The first term can be viewed as hyper-distribution estimation error, which is inversely proportional to the training circuit domain M since increasing M leads to an enhanced simulation of the hyper-distribution. The second term can be interpreted as average interdomain estimation error which measures the average sampling error across M circuit domains. Note that this error bound indicates that increasing the value of n_k can enhance the fitting to the circuit domain, thereby reducing the bound.

A.2. Proof of Corollary 4.3

Proof. The generalization error bound in Theorem 4.2 is

$$
\frac{C_1\log\delta^{-1} + C_2}{M} + \frac{C_3\log 2\delta^{-1}M + C_4\log\delta^{-1} + C_5}{M^2}\sum_{k=1}^{M}\frac{1}{n_k}
$$

The condition to prove Corollary 4.3 is that the number of domains M and the total number of samples $n = \sum_{k=1}^{M} n_k$ are fixed. Besides, based on the AM-HM Inequality (Djukić et al., 2011)

$$
(n_1 + n_2 + \cdots + n_M)(\frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_M}) \geq M^2
$$

we have

$$\begin{aligned}
&\frac{C_1 \log \delta^{-1} + C_2}{M} + \frac{C_3 \log 2\delta^{-1} M + C_4 \log \delta^{-1} + C_5}{M^2} \sum_{k=1}^{M} \frac{1}{n_k} \\
\geq &\frac{C_1 \log \delta^{-1} + C_2}{M} + \frac{C_3 \log 2\delta^{-1} M + C_4 \log \delta^{-1} + C_5}{M^2} \frac{M^2}{\sum_{k=1}^{M} n_k} \\
\geq &\frac{C_1 \log \delta^{-1} + C_2}{M} + \frac{C_3 \log 2\delta^{-1} M + C_4 \log \delta^{-1} + C_5}{n}
\end{aligned}$$

The condition for the above inequality to be equal is that $n_k = \frac{n}{M}$ for $k = 1, 2, \cdots M$. Therefore, we can conclude that the error bound reaches its minimum when $n_k = \frac{n}{M}$ for $k = 1, 2, \cdots M$, which proves that Corollary 4.3 holds. □

A.3. Proof of Corollary 4.4

Proof. Under the condition

$$n_k = \frac{n}{M} \text{ for } k = 1, 2, \cdots, M \tag{3}$$

and

$$1 \leq M \leq \frac{C_1 \log \delta^{-1} + C_2}{C_3 \log 2\delta^{-1}} \cdot n \tag{4}$$

we can conclude that using a multi-domain dataset (i.e., $M > 1$) will result in a smaller generalization error bound than just pooling them into one mixed dataset (i.e., M=1).

Based on the condition 3, the generalization error bound in Theorem 4.2 can be represented as a function on discrete variable M

$$B(M) = \frac{C_1 \log \delta^{-1} + C_2}{M} + \frac{C_3 \log 2\delta^{-1} M}{n} + \frac{C_4 \log \delta^{-1} + C_5}{n} \quad (M \geq 1)$$

where n representing the total number of samples is a constant and M denotes the number of domains. To prove Corollary 4.4, we just need to prove that $B(1) \geq B(M)$ for $M \geq 1$. Consequently, under the condition 4, we have:

$$\begin{aligned}
&1 \leq M \leq \frac{C_1 \log \delta^{-1} + C_2}{C_3 \log 2\delta^{-1}} \cdot n \\
\Rightarrow &\frac{C_3 \log 2\delta^{-1}}{n}(1 - M) - \frac{C_1 \log \delta^{-1} + C_2}{M}(1 - M) \geq 0 \\
\Rightarrow &(C_1 \log \delta^{-1} + C_2 + \frac{C_3 \log 2\delta^{-1}}{n} + \frac{C_4 \log \delta^{-1} + C_5}{n}) - (\frac{C_1 \log \delta^{-1} + C_2}{M} + \frac{C_3 \log 2\delta^{-1} M}{n} + \frac{C_4 \log \delta^{-1} + C_5}{n}) \geq 0 \\
\Rightarrow &B(1) \geq B(M) \quad (M \geq 1)
\end{aligned}$$

which proves that Corollary 4.4 holds.

Finally, we explain the reasonableness of condition 3 and 4. The condition 3 is a result in Corollary 4.3 and can be realized easily. Besides, in practical applications, n is often significantly larger than M and the constants C_1, C_2, C_3 are typically fixed values as demonstrated in A.1. Therefore, the condition 3 and 4 are both considered reasonable. □

B. Additional Results

B.1. Additional Results of Ablation Study

In this subsection, we perform ablation study to understand the contribution of each component in COG. Specifically, PruneX-COG without DADC aggregates all Circuit Datasets into a single domain, and uses our KDSR to learn node embeddings for classification. Then, PruneX-COG without DADC and KDSR further replaces our KDSR module with manually designed features. We report the offline prediction accuracy of COG, COG without Distributional Classifier

Table 3. The ablation study of COG on the Mfs2 heuristic under Evaluation Strategy 1. The best performance is marked in bold.

	Log2			Hyp			Multiplier		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.89 (0.04)**	**0.97 (0.01)**	**0.99 (0.01)**	**0.79 (0.04)**	**0.90 (0.03)**	**0.95 (0.008**	**0.91 (0.03)**	**0.96 (0.03)**	**0.98 (0.01)**
COG without DC	0.74 (0.12)	0.86 (0.10)	0.92 (0.04)	0.78 (0.06)	0.85 (0.03)	0.92 (0.02)	0.52 (0.12)	0.75 (0.15)	0.90 (0.03)
EnsembleMLP	0.61 (0.0)	0.65 (0.007)	0.85 (0.03)	0.62 (0.005)	0.73 (0.004)	0.82 (0.003)	0.63 (0.0)	0.71 (0.0)	0.75 (0.0)
	Sin			Square			Vga lcd		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	0.71 (0.06)	0.87 (0.04)	**0.95 (0.02)**	**0.90 (0.01)**	**0.94 (0.002)**	**0.95 (0.0)**	**0.93 (0.007)**	**0.95 (0.01)**	**0.98 (0.007)**
COG without DC	**0.72 (0.06)**	**0.91 (0.013)**	0.94 (0.02)	0.40 (0.009)	0.54 (0.04)	0.70 (0.02)	0.88 (0.08)	0.93 (0.07)	0.98 (0.0)
EnsembleMLP	0.56 (0.0)	0.58 (0.0)	0.67 (0.0)	0.85 (0.02)	0.91 (0.004)	0.92 (0.0)	0.19 (0.04)	0.29 (0.02)	0.53 (0.02)
	Ethernet			Wb conmax			Des perf		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.91 (0.09)**	**0.96 (0.04)**	**0.97 (0.03)**	**0.69 (0.04)**	**0.81 (0.05)**	**0.89 (0.04)**	**0.53 (0.0006)**	**0.68 (0.0007)**	**0.82 (0.0004)**
COG without DC	**0.97 (0.004)**	**0.98 (0.01)**	**0.98 (0.01)**	**0.71 (0.05)**	**0.84 (0.05)**	**0.92 (0.04)**	**0.53 (0.005)**	**0.68 (0.003)**	**0.82 (0.003)**
EnsembleMLP	0.53 (0.004)	0.62 (0.01)	0.63 (0.03)	0.43 (0.07)	0.57 (0.05)	0.68 (0.03)	0.44 (0.07)	0.49 (0.06)	0.58 (0.03)

Table 4. The ablation study of COG on the Resub heuristic under Evaluation Strategy 1. The best performance is marked in bold.

	Log2			Hyp			Multiplier		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.66 (0.02)**	**0.88 (0.02)**	**0.97 (0.004)**	**0.88 (0.008)**	**0.91 (0.02)**	0.94 (0.03)	0.74 (0.17)	**0.97 (0.02)**	**0.99 (0.003)**
COG without DC	0.55 (0.22)	0.67 (0.22)	0.69 (0.21)	0.86 (0.03)	0.91 (0.007)	**0.98 (0.01)**	0.85 (0.10)	0.95 (0.03)	0.99 (0.004)
EnsembleMLP	0.39 (0.1)	0.57 (0.2)	0.62 (0.20)	0.72 (0.02)	0.86 (0.005)	0.97 (0.01)	**0.88 (0.12)**	0.93 (0.08)	0.99 (0.01)
	Sin			Square			Vga lcd		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	0.46 (0.17)	**0.69 (0.07)**	0.76 (0.07)	**0.60 (0.11)**	**0.73 (0.08)**	**0.81 (0.07)**	0.47 (0.34)	**0.77 (0.04)**	**0.94 (0.02)**
COG without DC	0.37 (0.11)	0.52 (0.05)	**0.78 (0.04)**	0.56 (0.12)	0.68 (0.11)	0.76 (0.08)	0.48 (0.33)	0.56 (0.39)	0.62 (0.43)
EnsembleMLP	**0.48 (0.03)**	0.63 (0.1)	0.76 (0.07)	0.39 (0.09)	0.52 (0.12)	0.63 (0.08)	**0.68 (0.03)**	0.71 (0.01)	0.87 (0.01)
	Ethernet			Wb conmax			Des perf		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	0.82 (0.02)	**0.98 (0.009)**	**0.99 (0.0)**	**0.66 (0.07)**	**0.88 (0.08)**	**0.96 (0.01)**	**0.55 (0.07)**	**0.72 (0.05)**	**0.89 (0.009)**
COG without DC	**0.85 (0.08)**	0.97 (0.009)	0.98 (0.01)	0.49 (0.39)	0.56 (0.40)	0.69 (0.37)	0.40 (0.004)	0.56 (0.02)	0.71 (0.04)
EnsembleMLP	0.61 (0.01)	0.82 (0.04)	0.87 (0.05)	0.18 (0.26)	0.18 (0.26)	0.40 (0.11)	0.36 (0.04)	0.46 (0.03)	0.56 (0.03)

(COG without DC), and EnsembleMLP in Tables 3, 4, and 5. COG without DC aggregates the datasets from all circuits into a mixed domain, and uses our knowledge-driven subgraph representation to learn node embeddings for classification. Compared to COG without DC, EnsembleMLP further replaces our knowledge-driven subgraph representation module with manually designed node features (see Section E.8.2).

From the results in Tables 3, 4, and 5, we can draw two conclusions. First, COG without DC significantly outperforms EnsembleMLP in terms of the prediction accuracy, demonstrating the importance of our knowledge-driven subgraph representation module. Moreover, the results suggest that our representation module effectively learns domain-invariant representations for high generalization capability. Second, COG further improves COG without DC in terms of the prediction accuracy on most circuits. This demonstrates that formulating the circuits as multiple domains and learning domain-aware classifiers is important for further improving the generalization capability.

B.2. More Motivating Results

Here we present more motivating results.

B.2.1. Ineffective Node-Level Transformations problem

As shown in Table 6, the Mfs2 and Resub heuristics apply a large number of ineffective node-level transformations, with average of 93.81%. Moreover, the results in Table 7 demonstrate that the Rewrite and Refactor heuristics apply a large

Table 5. The ablation study of COG on the Mfs2 and Resub heuristics under Evaluation Strategy 2. The best performance is marked in bold.

heuristic/Circuit		Log2	Hyp	Multiplier	Sin	Square
	Method	top 50% acc	top 50% acc	top 50% acc	top 50% acc	top 50% acc
Mfs2	COG	**0.88 (0.02)**	**0.85 (0.06)**	**0.87 (0.04)**	**0.79 (0.12)**	**0.58 (0.05)**
	COG without DC	0.87 (0.05)	0.85 (0.01)	0.79 (0.02)	0.75 (0.06)	0.55 (0.02)
	EnsembleMLP	0.08 (0.007)	0.78 (0.01)	0.33 (0.01)	0.65 (0.08)	0.45 (0.007)
Resub	COG	**0.80 (0.005)**	**0.87 (0.03)**	**0.87 (0.0)**	**0.81 (0.03)**	**0.89 (0.01)**
	COG without DC	0.45 (0.23)	0.72 (0.17)	0.41 (0.31)	0.46 (0.23)	0.75 (0.03)
	EnsembleMLP	0.21 (0.02)	0.67 (0.12)	0.71 (0.06)	0.28 (0.0)	**0.92 (0.008)**

number of ineffective node-level transformations as well.

B.2.2. Analysis on the Runtime of Commonly Used LS heuristics

Runtime Percentage of the Mfs2 and Resub heuristics over Common LS Optimization Processes To evaluate the runtime percentage of applying the Mfs2 and Resub heuristics over common LS process (i.e., common LS optimization sequences of heuristics), we apply the following optimization sequence flows. (1) For the Resub, we apply the flow *strash; resyn2; resub -K 16 -N 3 -z*, which is commonly used in industrial LS process (Brayton, 2006). Note that *resyn2* denotes a fixed sequence of LS heuristics, i.e., *balance; rewrite; refactor; balance; rewrite; rewrite -z; balance; refactor -z; rewrite -z; balance*. (2) For the Mfs2, we apply the flow *strash; dch; if -C 12; mfs2 -W 4 -M 5000 -l*, which is commonly used in industrial LS process as well (Mishchenko et al., 2011).

The results in Table 11 show that the runtime percentage of applying the Mfs2 and Resub heuristics is about 79% of the total runtime of common LS optimization sequences. Thus, the results demonstrate that the runtime of applying the two heuristics acts as a bottleneck to the efficiency of LS.

Runtime of heuristics To evaluate the runtime of applying the commonly used heuristics, we apply the following optimization sequnece flows. (1) Given a logic optimization heuristic X, we apply the flow *strash; X*. Specifically, we apply the flow for the Rewrite, Refactor, Resub, and Balance heuristics. (2) Given a post-mapping optimization heuristic X, we apply the flow *strash; if -C 12; X*. Speicifically, we apply the flow for the Mfs2 heuristic.

We provide detailed results on the runtime analysis of these heuristics in the industrial setting in Tables 8 and 9. The results demonstrate that applying the Resub and Mfs2 heuristics take the longest runtime among the commonly used LS heuristics. For the Resub heuristic, K is an important hyperparameter, and represents the number of primary input nodes of subgraphs at each node when applying the heuristic. Moreover, the results in Tables 8 and 9 demonstrate that applying the Rewrite and Refactor heuristics are much faster than the Resub and Mfs2 heuristics. Nevertheless, we found that applying the Rewrite and Refactor heuristics are highly time-consuming on very large-scale circuits as shown in Table 10. Therefore, it is also valuable to improve the efficiency of the Rewrite and Refactor heuristics. Fortunately, the Rewrite/Refactor heuristics follow the same paradigm as the Resub/Mfs2 heuristics as shown in Fig. 4.1 in the main text. Moreover, the results in Table 7 demonstrate that the Rewrite and Refactor heuristics apply a large number of ineffective node-level transformations as well. Therefore, our proposed PruneX is applicable to the Rewrite and Refactor heuristics as well to improve their efficiency, especially on very large-scale circuits. We provide more discussion on how to apply PruneX to the two heuristics in Appendix E.8.4

B.2.3. More Details of the Out-of-Distribution (OOD) Generalization Problem in LS

In terms of the visualization experiments in Fig. 3 in the main text, we present more implementation details. We use the t-distributed stochastic neighbor embedding (t-SNE) (Van der Maaten & Hinton, 2008) algorithm to reduce the node features to two-dimensional space. That is, each point illustrates a reduced node feature. Moreover, the visualized points of the training data points can be too dense, as the training data points are much more than the testing data points. Thus, for visual clarity, we sample the same number of training data points as the testing set for visualization. In addition, we visualize data points from different circuits as well in Fig. 9a. The results show that the data distributions from different circuits are

Table 6. The results show that the proportion of effective nodes is very low, with an average of 6.19%.

				Mfs2					**Avg**
Stats/Circuit	hyp	log2	multiplier	square	sin	adder	sqrt	div	
Num (effective nodes)	664.00	67.00	93.00	182.00	36.00	0.00	27.00	16.00	
Num (total nodes)	64245.00	10648.00	7821.00	5709.00	2000.00	339.00	5673.00	9318.00	
Percent (%)	1.03	0.63	1.19	3.19	1.80	0.00	0.48	0.17	**1.06**
Stats/Circuit	mem ctrl	priority	router	int2float	cavlc	voter	ctrl	i2c	
Num (effective nodes)	4016.00	58.00	32.00	12.00	20.00	205.00	4.00	18.00	
Num (total nodes)	17430.00	261.00	111.00	91.00	286.00	2454.00	53.00	464.00	
Percent (%)	23.04	22.22	28.83	13.19	6.99	8.35	7.55	3.88	**14.26**
				Resub					
Stats/Circuit	hyp	log2	multiplier	square	sin	adder	sqrt	div	
Num (effective nodes)	6745.00	106.00	120.00	763.00	18.00	126.00	121.00	30.00	
Num (total nodes)	211330.00	29370.00	24556.00	16623.00	5039.00	1019.00	19437.00	40772.00	
Percent (%)	3.19	0.36	0.49	4.59	0.36	12.37	0.62	0.07	**2.76**
Stats/Circuit	mem ctrl	priority	router	int2float	cavlc	voter	ctrl	i2c	
Num (effective nodes)	999.00	93.00	2.00	4.00	24.00	974.00	17.00	62.00	
Num (total nodes)	45614.00	676.00	177.00	214.00	662.00	9756.00	108.00	1162.00	
Percent (%)	2.19	13.76	1.13	1.87	3.63	9.98	15.74	5.34	**6.70**
Total									**6.19**

Table 7. The results show that the proportion of effective nodes is low on the Rewrite and Refactor heuristics, with an average of 11.29% and 7.72%, respectively.

					Rewrite					Avg
Stats/Circuit	hyp	log2	multiplier	square	sin	des perf	ethernet	vga lcd	wb conmax	
Num (effective nodes)	83.00 (0.0)	1574.00 (0.0)	1716.00 (0.0)	605.00 (0.0)	198.00 (0.0)	6648.00 (0.0)	16636.00 (0.0)	34163.00 (0.0)	2250.00 (0.0)	
Num (total nodes)	214252.00 (0.0)	30486.00 (0.0)	25346.00 (0.0)	17879.00 (0.0)	5218.00 (0.0)	71651.00 (0.0)	53048.00 (0.0)	92548.00 (0.0)	45603.00 (0.0)	
Percent (%)	0.04	5.16	6.77	3.38	3.79	9.28	31.36	36.91	4.93	**11.29**
					Refactor					
Stats/Circuit	hyp	log2	multiplier	square	sin	des perf	ethernet	vga lcd	wb conmax	
Num (effective nodes)	1886.00 (0.0)	568.00 (0.0)	268.00 (0.0)	191.00 (0.0)	101.00 (0.0)	3339.00 (0.0)	12884.00 (0.0)	25350.00 (0.0)	4617.00 (0.0)	
Num (total nodes)	212449.00 (0.0)	31492.00 (0.0)	26794.00 (0.0)	18293.00 (0.0)	5315.00 (0.0)	74960.00 (0.0)	56800.00 (0.0)	101361.00 (0.0)	43236.00 (0.0)	
Percent (%)	0.89	1.80	1.00	1.04	1.90	4.45	22.68	25.01	10.68	**7.72**

similar but different, which demonstrates the reasonableness of our circuit domain formulation as well.

B.2.4. The Importance of the Prediction Recall on Optimization Performance

To analyze the relationship between the prediction recall of effective nodes and the optimization performance of heuristics, we evaluate the optimization performance of the Random method with different values of the hyperparameter k. Note that Random is a baseline that randomly predicts a score between $[0, 1]$ for each node, and selects the top k nodes to apply node-level transformations. Specifically, we report the recall and optimization performance (i.e., And Reduction) of Random with different values of k in Table 12. The results show that the value of k is approximately linearly positively correlated with the recall, and the recall is approximately linearly positively correlated with the optimization performance as well. Therefore, in order not to degrade the optimization performance of heuristics, the prediction recall of our model should be as high as possible. Thus, it is critical to tackle the out-of-distribution generalization problem in LS to improve the prediction recall on unseen circuits.

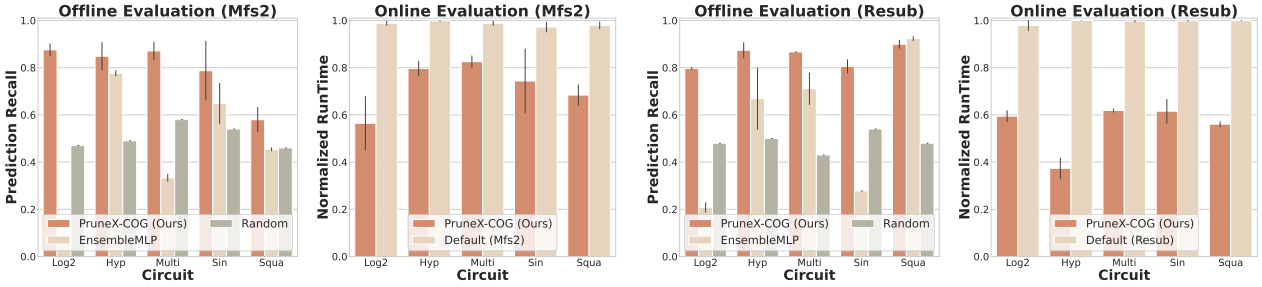
B.3. Oracle Prediction Results

Here we present the oracle prediction results on the Mfs2 heuristic and Log2 circuit. To evaluate whether only applying transformations on effective nodes can achieve similar optimization performance to that of the default heuristic, we conducted

the following oracle prediction experiment. Specifically, we apply the Mfs2 heuristic to the Log2 circuit and recorded the set of node ids for all effective nodes. Based on this set, we implemented an Oracle version of Mfs2 that only apply transformations to nodes in the set. The results in Table 14 show that Oracle significantly improves the runtime while achieving the same size (i.e., the number of nodes) and depth (i.e., level) compared to Default (i.e., the default Mfs2 heuristic).

B.4. The Relationship between the Number of Nodes to Apply Transformations and Runtime

To evaluate whether reducing node-level transformations can reduce the runtime of heuristics, we test the Random method with different values of k on open-source circuits. Note that Random is a baseline that randomly predicts a score between $[0, 1]$ for each node, and selects the top k nodes to apply node-level transformations. The results in Table 13 demonstrate that the runtime of applying the resub heuristic significantly increases with the number of applied node-level transformations. Therefore, our method can significantly reduce the runtime of applying heuristics by reducing a large number of ineffective node-level transformations.



(a) Evaluation Strategy 2 on the Mfs2 heuristic.

(b) Evaluation Strategy 2 on the Resub heuristic.

Figure 7. The results demonstrate that our PruneX-COG achieves significant prediction recall improvement (Left, ↑), runtime reduction (Middle, ↓), and marginal QoR (size) degradation (Right, ↓). The normalized runtime (node number) denotes the ratio of the runtime (node number) to that of applying the Default heuristic to testing circuits.

B.5. More Results on the Open-Source Benchmarks

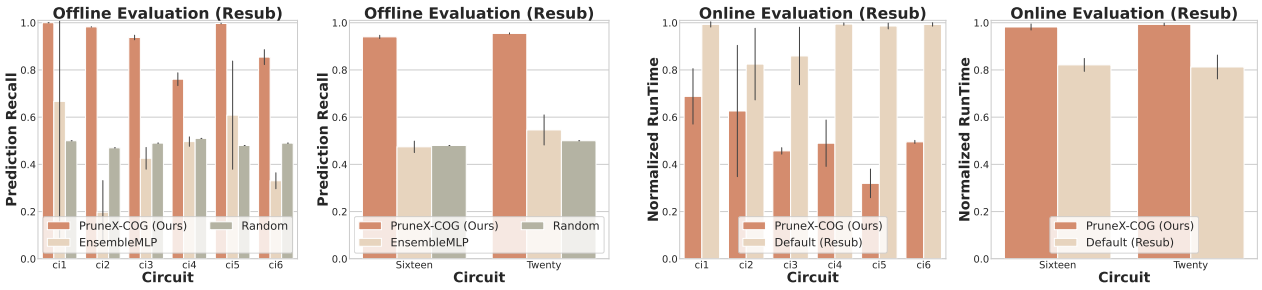
We first provide details of the results on the open-source benchmarks. Specifically, we use three metrics, i.e., top 50% acc, normalized runtime, and normalized node number. The top 50% acc denotes the top 50% accuracy metric. The normalized runtime denotes the ratio of the runtime to that of the default heuristic. The normalized node number denotes the ratio of the node number to that of the circuits optimized by the default heuristic.

Then, in terms of experiments on the open-source benchmarks, we provide detailed **offline** evaluation results of our PruneX on both the Mfs2 and Resub heuristics in Tables 15, 16, 17, and Figure 7a. Moreover, we provide detailed **online** evaluation results of our PruneX on both the Mfs2 and Resub heuristics in Tables 18, 19, 20 and Figure 7b. Note that compared with the first strategy, it is more challenging to achieve good generalization performance under Evaluation Strategy 2 due to the larger distribution shift between the training and testing datasets. Specifically, for the Mfs2 heuristic under Evaluation Strategy 2, PruneX-COG achieves 28% recall improvement. Moreover, PruneX-COG reduces the runtime by 26.66% with 0.07% degradation on average in terms of the size of circuits. Under Evaluation Strategy 1, PruneX-COG degrades the size by 0.36%. For the Resub heuristic, PruneX-COG reduces the runtime by 44.42% and degrades the size by 0.22% on average under Evaluation Strategy 2. Under Evaluation Strategy 1, PruneX-COG degrades the size by 0.48%. Overall, the results demonstrate that our proposed PruneX significantly improves the efficiency of the Resub and Mfs2 heuristics, while keeping comparable optimization performance.

B.6. More Results on Industrial Circuits and Very Large-Scale Circuits

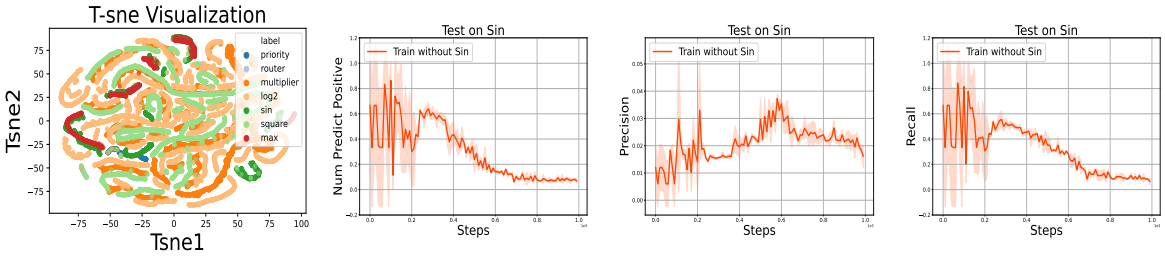
We first provide details of the results on industrial circuits and very large -scale circuits. Specifically, we use three metrics, i.e., top 50% acc, normalized runtime, and normalized node number. The top 50% acc denotes the top 50% accuracy metric. The normalized runtime denotes the ratio of the runtime to that of the default heuristic. The normalized node number denotes the ratio of the node number to that of the circuits optimized by the default heuristic.

Then, in terms of experiments on the industrial circuits and very large-scale circuits, we provide detailed **offline** evaluation



(a) Offline evaluation on the industrial and very large-scale circuits. (b) Online evaluation on the industrial and very large-scale circuits..

Figure 8. The results demonstrate the strong ability of our method to promote efficient LS on industrial and very large-scale circuits with the Resub heuristic.



(a) Visualization

(b) Class Imbalance Problem

Figure 9. (a) We visualize the data points from different circuits. (b) We train models using EnsembleMLP with a dataset without Sin, and evaluate the models on Sin. The results (Left) show that the number of predicted positive samples is quite small. Moreover, the results (Middle and Right) show that learned models performs poorly in terms of the precision and recall.

results of our PruneX on both the Mfs2 and Resub heuristics in Tables 21, 22, 25, 26 and Figure 8a. Moreover, we provide detailed **online** evaluation results of our PruneX on both the Mfs2 and Resub heuristics in Tables 23, 24, 27, 28 and Figure 8b. For the large-scale circuits, PruneX-COG only degrades the size by 0.001% on average. Overall, the results demonstrate the strong performance of our proposed PruneX on the industrial circuits and very large-scale circuits.

B.7. More Results on Improving QoR with PruneX-COG

Optimization Sequence Flows for 2PruneX-COG To apply our PruneX-COG twice, we apply the sequence of heuristics, i.e., *strash; dch; if -C 12; mfs2 -W 4 -M 5000; strash; if -C 12; mfs2 -W 4 -M 5000*, to evaluate the performance of 2PruneX-COG. Note that the mfs2 heuristic is a post-mapping optimization heuristic, whose input DAG is a k-input look up table graph (K-LUTs). Moreover, the strash heuristic transforms the current circuit representation into an And-Inverter Graph (AIG) by one-level structural hashing. Then, the if (Mishchenko et al., 2007) heuristic maps an AIG into a K-LUTs. Finally, the Mfs2 heuristic optimizes the input K-LUTs. Note that the strash and if heuristics are much faster than the Mfs2 heuristic.

We provide additional results of improving QoR with 2PruneX-COG on the Log2 and Sin circuit in Table 29. The results demonstrate that our PruneX can not only reduce the runtime of heuristics, but also improve its optimization performance in terms of the size and depth.

B.8. Discussion on the Class Imbalance Problem

We use EnsembleMLP to train models on a dataset without Sin, and evaluate the models on Sin. We use 0.5 as the threshold, and predict the nodes whose model prediction probability is greater than 0.5 as positive samples. As shown in Figure 9b, learned models predict only about 6% of the samples to be positive on the Sin circuit. This implies that our learned models suffer from serious negative bias problem due to the class imbalance problem. Moreover, the results in Figure 9b show that learned models perform poorly in terms of the precision and recall, which suggests that 0.5 is an inappropriate threshold.

B.9. Discussion on the Model Inference Time

We use multiple representative circuits with node sizes ranging from $20,000$ to $780,000$ to test the bipartite graph construction time and model inference time of our COG. As shown in Table 30, the ratio of the total model time to the runtime of the Mfs2 heuristic (i.e., Percent) is very low, with an average of 2.66%. Note that as the size of the circuit increases, the total model time also increases significantly. For example, on the f0089 circuit with $788,288$ nodes, the total model time takes 36.41 seconds, which is much longer than that taken on a smaller circuit. The major reason is that our GPU (i.e., a NVidia GeForce GTX 3090 Ti) memory is limited, and we have not implemented multi-card parallel inference. That is, our model inference is performed serially and in batches on large-scale circuits, which could greatly increase the model inference time. In contrast, we can implement multi-card parallel inference to further reduce the total model time, especially on large-scale circuits.

C. Details of Datasets Used in This Paper

C.1. Description of Three Used Benchmarks

The industrial benchmark includes 27 circuits. We present detailed statistics of the circuits from the EPFL and IWLS in Tables 31 and 32. Moreover, we present detailed statistics of the industrial circuits and very large-scale circuits in Tables 34 and 33. In general, a circuit is modeled as a directed acyclic graph (DAG), where nodes represent logic gates and directed edges represent wires connecting the gates. The fanins of a node are nodes that are driving this node. The fanouts of a node are nodes driven by the node. The primary inputs (PIs) denote the nodes without fanins. The primary outputs (POs) denote a subset of the nodes of the network. Latches can be considered as specialized nodes within sequential circuits. Cubes refer to logical expressions that represent subsets of input variables in Boolean functions. Lev denotes the depth of the DAG, i.e., the maximum number of edges between PIs and POs.

C.2. Datasets for Evaluation on Open-Source Benchmarks

For each circuit and a given X heuristic, we collect the circuit dataset by applying the X heuristic to optimizing the circuit and collecting the node features $\{\mathbf{x}_i\}_{i=1}^n$ and labels $\{y_i\}_{i=1}^n$. We found that there are a small number of circuits with no effective nodes. We discard these circuits, as we can directly avoid applying transformation to these circuits without learning a model.

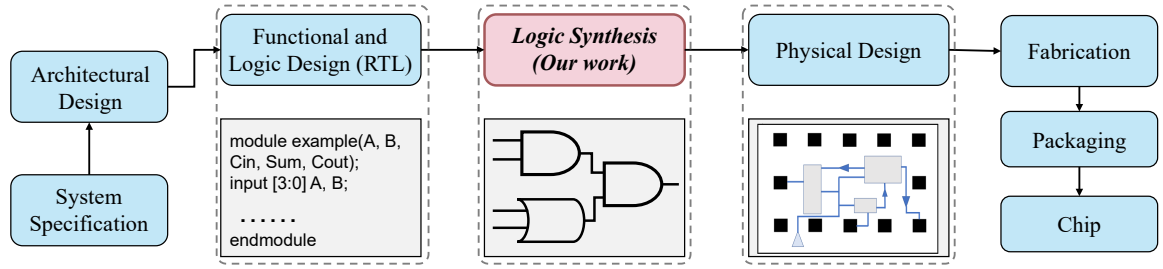
Specifically, under Evaluation Strategy 1 using the EPFL benchmark, we construct five datasets for evaluation. We use one of the five circuit datasets, i.e., circuit datasets collected from Log2, Hyp, Multiplier, Sin, Square, as the testing dataset, and the other circuit datasets as the training dataset. Furthermore, under Evaluation Strategy 1 using the IWLS benchmark, we construct four datasets for evaluation. We use one of the four circuit datasets, i.e., circuit datasets collected from Vga lcd, Ethernet, Wb conmax, and Des perf, as the testing dataset, and the other circuit datasets as the training dataset.

Moreover, under Evaluation Strategy 2, we construct a dataset for evaluation. Specifically, we use the five circuit datasets from Log2, Hyp, Multiplier, Sin, Square as the testing dataset, and the other circuit datasets from the EPFL and IWLS as the training dataset. To promote the community of machine learning for LS, we will release these datasets once the paper is accepted to be published.

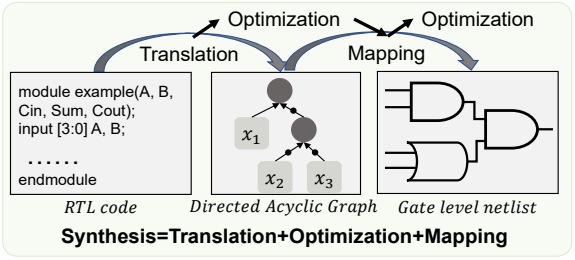
C.3. Datasets for Evaluation on Industrial Circuits and Very Large-Scale Circuits

In terms of the industrial circuits, we report a statistical description of the training and testing circuits in Table 34. As shown in Tables 34, 31, and 32, the industrial circuits consist of 27 industrial circuits, where the circuit sizes range from $2,775$ to $788,288$, which are much larger in size than open-source circuits. Note that we evaluate our method using Evaluation Strategy 2, with 21 circuits for training and 6 circuits for testing.

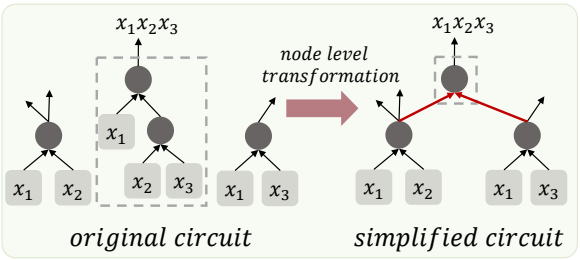
In terms of the very large-scale circuits (scale up to twenty million nodes), we report a detailed description of them in Table 33. Due to the small number of very large-scale circuits, we evaluate our method using Evaluation Strategy 1 mentioned in Experiment 1 in Section 5. Specifically, we use circuits from the EPFL mentioned in Table 31 and the two very large-scale circuits to construct datasets. (1) We set the Sixteen circuit as the testing dataset, and the rest as the training dataset. (2) We set the Twenty circuit as the testing dataset, and the rest as the training dataset. To promote the community of machine learning for LS, we will release the datasets once the paper is accepted to be published.



(a) The chip design workflow



(b) Logic synthesis



(c) Circuit optimization

Figure 10. (a) We illustrate the chip design workflow, and we focus on Logic Synthesis (LS) in this paper. (b) LS converts the behavioral description of a circuit to a transistor-level implementation, which generally consists of circuit optimization and technology mapping (Mishchenko et al., 2007; Ren & Hu, 2023). (c) Circuit optimization aims to simplify an input circuit without changing its functionality.

D. Details of Related Work and Background

D.1. Related Work

Machine Learning for Logic Synthesis As chip complexity has grown exponentially with the development of semiconductor technology, using machine learning (ML) to assist the automated chip design workflow has been an active topic of significant interest in recent years (Mirhoseini et al., 2021; Huang et al., 2021; Sánchez et al., 2023; Neto et al., 2021a; Lai et al., 2022; 2023). As shown in Figure 10a, the chip design workflow consists of many stages, such as high-level synthesis (Yao et al., 2024; Liu et al., 2022), logic synthesis (Li et al., 2023a; Zhu et al., 2023; Li et al., 2024a; Liu et al., 2023a;b), placement (Lai et al., 2022; 2023; Geng et al., 2024; Chen et al., 2023a), design space exploration (Chen et al., 2024; Zuo et al., 2023), etc (Huang et al., 2021; Ren & Hu, 2023). In most of the stages in the workflow, recent studies have demonstrated significant improvement by using ML methods compared with traditional methods, including high-level synthesis (Makrani et al., 2019; Kim et al., 2018; Liu & Carloni, 2013), logic synthesis (Neto et al., 2021a; Berndt et al., 2022; Neto et al., 2019a), and placement (Mirhoseini et al., 2021; Lai et al., 2022; 2023; Agnesina et al., 2023; Cheng et al., 2022). In this paper, we focus on using machine learning to promote efficient logic synthesis (LS), which plays a vital role in efficient chip design and can yield substantial economic value (Fawcett, 1994; Neto et al., 2021a). Existing research on machine learning for LS can be roughly divided into three categories (Berndt et al., 2022; Ren & Hu, 2023). First, (Synopsys, 2020; Cadence, 2021; Hosny et al., 2020b; Grosnit et al., 2022) use machine learning to tune the optimization flow of LS operators. Second, (Neto et al., 2021a; Kirby et al., 2019b; Zhou et al., 2019) use machine learning to predict key metrics after physical design and leverage the prediction to guide LS optimization. Third, (Neto et al., 2021b; 2019b) use machine learning to improve decision-making in traditional LS methods. Different from existing work, our work uses machine learning to improve the paradigm of traditional LS operators to promote efficient LS. An appealing feature of our PruneX is that it is applicable to many LS operators—which follow the paradigm illustrated in Figure 1—to significantly improve their efficiency.

Generalizable Prediction in Chip Design Workflow Prior research has investigated the utilization of machine learning (ML) techniques to develop generalizable congestion prediction models within the chip design workflow (Kirby et al., 2019a; Wang et al., 2022a; Ghose et al., 2021; Yang et al., 2022). Nevertheless, our work differs from previous studies in two fundamental aspects. **First**, we address dissimilar input data. Prior research mainly focuses on the physical design stage with circuits represented by gate-level netlists or designed layouts, while we focus on the logic synthesis stage with circuits represented by Boolean networks. The dissimilarity in input data poses significant challenges in directly

applying their methods to our setting. To the best of our knowledge, our work is the first data-driven method to well tackle the out-of-distribution (OOD) generalization problem across circuits in LS operators, which is critical for the success of data-driven LS algorithms. **Second**, we employ different methodologies for learning models. Generally, they propose problem-specific graph neural network architectures and learn the models via supervised learning. In contrast, our PruneX formulates the OOD generalization problem across circuits as a novel circuit domain generalization task, which offers promising avenues for future research on prediction tasks in chip design. Moreover, based on a key observation called the transformation-invariant domain knowledge, our PruneX further proposes to learn domain-invariant representations for enhanced generalization capabilities. Moreover, our work is also related to studies on out-of-distribution generalization (Wang et al., 2022b; Xie et al., 2024b;a; Chen et al., 2023b; 2022) and graph neural networks (Hamilton, 2020; Shi et al., 2023; Li et al., 2023c;b; Geng et al., 2022).

Machine Learning for Combinatorial Optimization Logic optimization, which is a key task in logic synthesis, is also essentially a combinatorial optimization problem. The use of machine learning to tackle combinatorial optimization problems has been an active topic of significant interest in recent years (Bengio et al., 2021; Gasse et al., 2019; Geng et al., 2023; Wang et al., 2023b; 2024b; Ling et al., 2024; Li et al., 2024b; Wang et al., 2021; 2024a; 2023a; Wang & Yu, 2023).

D.2. Background

Logic Synthesis (LS) Driven by Moore's law, the chip design complexity has grown exponentially (Khailany, 2020; Lopera et al., 2021; Huang et al., 2021; Mirhoseini et al., 2021; Ren & Hu, 2023). Thus, the chip design workflow has incorporated multiple Electronic Design Automation (EDA) tools to synthesize, simulate, test, and verify different circuit designs efficiently and reliably. These EDA tools automatize the chip design workflow as shown in Figure 10a. A LS tool—which aims to transform a behavioral description of a design into an optimized gate-level circuit implementation—is one of the most important modules in the EDA tools. In general, LS consists of pre-mapping optimization, technology mapping, and post-mapping optimization (Hosny et al., 2020a; Ren & Hu, 2023). **In this paper, we define Circuit Optimization (CO) by both pre-mapping optimization and post-mapping optimization.** First, in the pre-mapping optimization phase, logic optimization operators, such as Rewrite (Bertacco & Damiani, 1997), Resub (Brayton, 2006), and Refactor (Brayton, 1982), are applied to an input circuit to optimize the circuit. Then, in the technology mapping phase, the optimized logic circuit is mapped to the available technology library, e.g., a standard-cell netlist (Brayton & Kam) or k-input lookup-tables (Mishchenko et al., 2007). Finally, post-mapping optimization operators, such as Mfs2 (Mishchenko et al., 2011), are applied to the mapped circuit to further optimize it.

Circuit Optimization In the LS stage, a circuit is usually modeled by a DAG. Common types of DAGs for CO include And-Inverter Graphs (AIGs) for pre-mapping optimization (Bertacco & Damiani, 1997; Mishchenko et al., 2006) and K-Input Look-Up Tables (K-LUTs) for post-mapping optimization (Mishchenko et al., 2007; 2011). In the pre-mapping optimization phase, an AIG is a DAG containing four types of nodes: the constant, PIs, POs, and two-input And (And2) nodes. A graph edge is either complemented or not. A complemented edge indicates that the signal is complemented. In the post-mapping optimization phase, a K-LUT is a DAG with nodes corresponding to Look-Up Tables and directed edges corresponding to wires. A Look-Up Table in a K-LUT is a digital memory that implements the Boolean function of the node.

Commonly Used LS Heuristics A rich set of LS heuristics have been developed to tackle the CO task in the pre-mapping and post-mapping optimization phases (Brayton & Mishchenko, 2010). In this paper, we focus on commonly used LS heuristics on large-scale industrial circuits—that is, the Resub (Brayton, 2006), Mfs2 (Mishchenko et al., 2011), Rewrite (Bertacco & Damiani, 1997), and Refactor (Brayton, 1982) heuristics—which often act as a bottleneck to the efficiency of the LS optimization processes. We notice that these LS heuristics follow the same paradigm as shown in Fig. 1. Specifically, they apply transformations to subgraphs rooted at each node—that is, the node-level transformation—sequentially for all nodes on an input DAG. Note that the major differences among these operators lie in the node-level transformation mechanism.

E. Details of Methods and Experimental Settings

E.1. Details of Experimental Setup

E.1.1. Optimization Sequence Flows for Collecting Data and Evaluation

In the industrial setting, we usually apply a sequence of Logic Synthesis (LS) heuristics to optimizing an input circuit. Thus, we follow the setting throughout all experiments unless mentioned otherwise. Specifically, in terms of the Mfs2 heuristic, we

apply the sequence of heuristics, i.e., *strash; dch; if -C 12; mfs2 -W 4 -M 5000*, to collect data and evaluate the performance of the Default Mfs2 heuristic and our PruneX. Note that the optimization sequence flow is a standard academic flow for evaluating the Default Mfs2 heuristic, which follows previous work (Mishchenko et al., 2011). Moreover, in terms of the Resub heuristic, we apply a common commercial optimization sequence flow widely used in the industrial setting, i.e., *strash; resyn2; resub -K 16 -N 3 -z*, to collect data and evaluate the performance of the Default Resub heuristic and our PruneX. We train our method with ADAM (Kingma & Ba, 2014) using the PyTorch.

E.1.2. Optimization Sequence Flows for Evaluating 2PruneX-COG

To apply our PruneX-COG twice, we apply the sequence of heuristics, i.e., *strash; dch; if -C 12; mfs2 -W 4 -M 5000; strash; if -C 12; mfs2 -W 4 -M 5000*, to evaluate the performance of 2PruneX-COG. Note that the mfs2 heuristic is a post-mapping optimization heuristic, whose input DAG is a k-input look-up table graph (K-LUTs). Moreover, the strash heuristic transforms the current circuit representation into an And-Inverter Graph (AIG) by one-level structural hashing. Then, the if (Mishchenko et al., 2007) heuristic maps an AIG into a K-LUTs. Finally, the Mfs2 heuristic optimizes the input K-LUTs.

E.1.3. Top k Accuracy Metric

In our implemented PruneX, we sort all the nodes according to the prediction scores given by our learned classifier, and select the top k nodes—that is, the nodes with top k scores. That is, the top k nodes are predicted positive, and the other nodes are predicted negative. Then, the top k accuracy metric is defined by the recall, i.e., the fraction of true positive nodes that are predicted to be positive.

E.2. Implementation Details of the Focal Loss

The Focal Loss is first introduced to address the class imbalance problem in the object detection scenario (Lin et al., 2017). In this paper, we leverage the Focal Loss to alleviate the class imbalance problem in learning classifiers for logic synthesis. Specifically, the loss function for each data pair $(\mathbf{x}, y)$ takes the form of

$$\begin{aligned} l(f(\mathbf{x}), y) = &- \alpha y(1 - f(\mathbf{x}))^{\gamma} \log(f(\mathbf{x})) \\ &- (1 - \alpha) f(\mathbf{x})^{\gamma} \log(1 - f(\mathbf{x})), \end{aligned}$$

where $\alpha \in [0, 1]$ denotes a weighting factor, and $\gamma \geq 0$ denotes a tunable focusing parameter. In this paper, we set α by inverse class frequency, and γ by 2, which follows (Lin et al., 2017).

E.3. Implementation Details of EnsembleMLP

EnsembleMLP aims to learn a classifier to predict the probability (i.e., score) of a node that it is effective. For a fair comparison, EnsembleMLP uses the same node features as COG in Table 35, and we train EnsembleMLP via the Focal Loss as well. Specifically, we implement the EnsembleMLP with a multi-layer perceptron containing three hidden layers with 1024 units. To enhance the generalization ability, we train the EnsembleMLP with the ensemble learning trick (Zhou & Zhou, 2021). Specifically, we train the EnsembleMLP with 15 ensembles.

E.4. Implementation Details of GCNN

Our GCNN model takes as input the bipartite graph and performs a single graph convolution, in the form of two interleaved half-convolutions. Specifically, we break down our graph convolution into two successive passes, i.e., one from the root node to the non-root nodes and one from the non-root nodes to the root node. The passes take the form of

$$\begin{aligned} \mathbf{h}(\mathbf{c}_i) &\leftarrow \mathbf{f}_{\mathcal{C}}(\mathbf{c}_i, \mathbf{g}_{\mathcal{C}}(\mathbf{c}_i, \mathbf{t}_1)) \\ \mathbf{h}(\mathbf{t}_1) &\leftarrow \mathbf{f}_{\mathcal{T}}(\mathbf{t}_1, \frac{1}{m} \sum_{i=1}^{m} \mathbf{g}_{\mathcal{T}}(\mathbf{h}(\mathbf{c}_i), \mathbf{t}_1)) \end{aligned}$$

for all $i \in \{1, \ldots, m\}$, where $\mathbf{f}_{\mathcal{C}}, \mathbf{g}_{\mathcal{C}}, \mathbf{f}_{\mathcal{T}}$, and $\mathbf{g}_{\mathcal{T}}$ are two-layer perceptrons with relu activation functions, $\mathbf{t}_1$ denotes the root node feature, and $\mathbf{c}_i$ denotes the i-th non-root node feature.

E.5. Discussion on Assumption 4.1

With our circuit aggregation mechanism, circuits can be categorized in various ways. For instance, in the EPFL dataset, circuits can be aggregated into arithmetic and control domains based on their respective functionalities. Additionally, there is an option to treat each circuit within the EPFL dataset as an individual domain, which is similar to the natural division of domains in computer vision (Shen et al., 2021). Furthermore, the flexibility extends to the number of circuits in each domain. For example, an Arithmetic Logic Unit (ALU) circuit is proficient in both arithmetic computations and control operations, allowing its placement in either the arithmetic or control domain without violating the i.i.d assumption.

In our setting, for each of N circuits, a set of nodes are obtained by applying an X heuristic to the circuit. The nodes are then labeled based on whether the corresponding node-level transformation is effective. Thus, the training set consists of heterogeneous samples from several distributions, i.e., nodes from several different circuits. That is, we are given N similar and related distributions, and aim to generalize to a similar but different test distribution. Regarding the source of these circuits, each circuit may be a sample of a complex integrated circuit, such as an addition circuit, storage circuit, control circuit, etc. In addition, aggregating circuits with similar functionality is similar to sampling some small building blocks from a complex integrated circuit. Therefore, it is reasonable to assume that the training Circuit Domains and testing Circuit Domains come from the same hyper-distribution. That is, Assumption 4.1 holds.

Moreover, we empirically show that the data distributions from N different circuits are N similar but different distributions in Figure 9a, which suggests that Assumption 4.1 holds in practice as well.

E.6. Details on Our Circuit Aggregation Mechanism

In this paper, we aggregate circuits based on their sample sizes and functionality. In general, we divide the real industry setting into two categories. In the first category, we are given N circuits with their functionality information. In the second category, we are given N circuits without their functionality information for privacy protection. In the first category, we aggregate those circuits with the same high-level functionality. Take the EPFL benchmark as an example, we aggregate circuits with the arithmetic functionality and control functionality into an arithmetic Circuit Domain and a control Circuit Domain, respectively. Thus, the two Circuit Domains contain two large datasets with similar sample sizes, which is beneficial for achieving a small generalization error bound. Moreover, the two Circuit Domains may come from small integrated units from a complex integrated circuit, which follows Assumption 4.1. In the second category, we aggregate circuits to make the sample sizes across domains nearly uniform. Specifically, we sort the circuits according to their sample sizes, and divide the odd-numbered circuits into one domain, and the even-numbered circuits into another domain. In this paper, we assume that the IWLS and industrial benchmarks fall into the second category.

E.7. Details of the Transformation-Invariant Domain Knowledge

Here we take the node-level transformation in the Resub and Mfs2 heuristics, i.e., the resubstitution transformation (Brayton, 2006), as an example. The node-level transformations in the Rewrite (Bertacco & Damiani, 1997) and Refactor (Brayton, 1982) heuristics are different from but similar to the resubstitution transformation. Please refer to (Bertacco & Damiani, 1997) for details. In terms of the resubstitution transformation, we are given a current node called the root node. The resubstitution transformation first uses a heuristic rule to collect a limited number of *candidate input nodes* of the root node. In general, the candidate input nodes include two types of nodes: (a) the nodes that are on the input direction of the root node and distance-m or less from the root node, and (b) the nodes that are on the output direction of the type (a) nodes and with level not exceeding the level of the root node. The resubstitution transformation focuses on the subgraph constructed by the root node and candidate input nodes. Next, the resubstitution transformation aims to find a new set of input nodes from *candidate input nodes* to replace existing input nodes of the current node *without changing its functionality*, thereby reducing the size and/or depth of the circuit.

Thus, the resubstitution transformation mechanism is invariant across different circuits. Moreover, whether a resubstitution transformation can be effective is highly related to the focused subgraph regardless of what the global graph is. Based on the transformation-invariant domain knowledge, we propose to extract the subgraph rooted at a node to learn its node embedding for discriminative classification, which carries the potential to learn domain-invariant representations and thus well generalize to unseen circuits. Please see (Brayton, 2006) for details on the resubstitution.

E.8. Details of PruneX and COG

E.8.1. Data Collection and Model Learning

In the offline phase, we aim to collect a training dataset $\mathcal{D}$ from multiple existing logic circuits $\{\mathcal{C}_i\}_{i=1}^N$ given a LS operator. In general, a logic circuit is modeled as a directed acyclic graph (DAG), where nodes represent logic gates and directed edges represent wires connecting the gates. Let X denote a LS operator. To generate the Circuit Dataset, we apply the X operator to optimizing the circuit, which applies node-level transformations to each node on the DAG. For each node in the circuit, we generate a data pair $(\mathbf{x}, y)$, where $\mathbf{x}$ denotes the node, and y denotes the label. Specifically, if the node-level transformation is effective at the node $\mathbf{x}$, then $y = 1$. Otherwise, $y = 0$. Given the X operator and N circuits, we generate a dataset $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^N = \{\{\mathbf{x}_j^i, y_j^i\}_{j=1}^{n_i}\}_{i=1}^N$.

Given the generated dataset $\mathcal{D}$, we learn a binary classifier $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ denotes the input space of nodes, and $\mathcal{Y}$ denotes $\{0, 1\}$. Specifically, we formulate the prediction task as a binary classification task. The optimization objective for each data pair $(\mathbf{x}, y)$ takes the form of $l(f(\mathbf{x}), y) = -y \log(f(\mathbf{x})) - (1-y) \log(1 - f(\mathbf{x}))$. That is, the output of our learned model for an input node represents the probability that the node is an effective node. Note that the Ineffective Node-Level Transformations (INT) problem leads to an extreme *class imbalance* in the training dataset. This severe imbalance poses a substantial challenge to the classification task (Lin et al., 2017; Rota Bulo et al., 2017). To tackle this problem, we leverage the Focal Loss (Lin et al., 2017), which has been shown successful in addressing class imbalance for object detection tasks. We provide more details on the Focal Loss in Appendix E.2.

E.8.2. Designed Node Features

As shown in Table 35, we design features for each node to contain the functionality information, structure information, and other possible useful features. As the Resub heuristic and the Mfs2 heuristic are logic optimization and post-mapping optimization heuristics respectively, their corresponding input directed acyclic graphs are different. For the logic optimization heuristic Resub, the input circuit is usually represented as an And-Inverter Graph (AIG). For the post-mapping optimization heuristic Mfs2, the input circuit is represented as a k-inputs Look up tables (k-LUTs). Please refer to Section D.2 for details on the AIG and k-LUTs.

Therefore, for the two different types of heuristics, the specific designed node features are slightly different. Specifically, for the Mfs2 heuristic, we design node features with the truth table of the node (i.e., a 6-input look up table) and other possible useful features, such as the fanin/fanout number and level of the node. Overall, the node feature for the Mfs2 heuristic is a 69-dimensional vector. Moreover, for the Resub heuristic, we consider features from the node itself and from its two children to capture structural information, which follows (Neto et al., 2021b). We use the two features inv0 and inv1 to capture the functionality information, where inv0/inv1$= 1$ denotes that the left/right input edge is inverted. Thus, the node feature consists of the root node feature (8-dimensional vector) and its two children features (two 6-dimensional vectors). Note that its children features do not contain the cut information, i.e., the Leaves number and Cuts number. Overall, the node feature for the Resub heuristic is a 20-dimensional vector.

E.8.3. Implementation Details of PruneX

As shown in Fig. 1, our proposed PruneX first extract node features, i.e., constructing bipartite graphs for all nodes as mentioned in Section 4.2. PruneX then call the learned classifier to predict the score that a node is effective for all nodes. The learned classifier is heuristic-specific. That is, we learn an individual classifier for each given heuristic, as the input circuit representation for different heuristics are different and thus the designed node features are different. Note that an input circuit is modeled by a directed acylic graph (DAG), and each node on the DAG has a unique identification number (ID). Thus, PruneX sort the node IDs in descending order according to the scores, and select the top $k\%$ node IDs. Finally, PruneX only applies node-level transformations in sequence to nodes in the selected node ID set.

The model inference time in PruneX is low, as PruneX only needs to call the inference model once. However, there may be a marginal shift between the predicted node IDs and the true node IDs. The reason is that when applying node-level transformations in sequence, the structure of the graph may change. This causes the true node IDs to change, while the predicted node IDs are fixed. Fortunately, we found that the shift between the predicted node IDs and the true node IDs has marginal impact on the optimization performance of the heuristics for the following two reasons. First, the ID set of effective nodes is quite sparse due to the ineffective node-level transformations problem. Therefore, the influence between consecutive effective node-level transformations is slight. Second, the results in Table 14 show that only applying transformations to true

effective nodes achieves the same optimization performance as the default Mfs2 heuristic (see Appendix B.3 for details).

E.8.4. Discussion on How to Apply PruneX to the Rewrite and Refactor heuristics

The Rewrite and Refactor heuristics follow the same paradigm as the Resub and Mfs2 heuristics. We illustrate the unified paradigm in Figure 1 (Left) in the main text. Note that the major difference between these heuristics lies in the node-level transformation. That is, the specific methods for node-level transformation are different between these heuristics. For example, the Resub heuristic applies the resubstitution transformation. We provide details on the resubstitution transformation in Appendix E.7. As shown in Figure 1 (Right), our PruneX uses a learned classifier to predict those nodes with effective node-level transformations in advance, and applies node-level transformations to these predicted effective nodes sequentially. Note that the paradigm of our PruneX is independent from the node-level transformations. Thus, our PruneX can also be applied to the Rewrite and Refactor heuristics.

E.8.5. Implementation Details of COG

Hardware Specification Throughout all experiments, we use a single machine that contains eight GPU devices (Nvidia GeForce GTX 3090 Ti) and two Intel Gold 6246R CPUs.

Neural Network Architecture The GCNN model encodes node features into embeddings with dimension 128. The multi-head neural network contains a shared multi-layer perceptron containing three hidden layers with 1024 units with M heads branching off independently. To output classification probabilities for values between 0 and 1, the multi-head neural network uses a sigmoid activation function at the last layer.

Training Details Throughout all experiments, we apply Adam optimizer with learning rate $\alpha = 1e - 4$ to optimize our models. Moreover, for training stability, we apply linear learning rate decay with step size 100 and decay rate 0.96. For generalization to very large-scale circuits, we apply Min-Max Normalization to data in training and testing datasets. Due to limited video memory, we set the batch size to 10240. For simplicity, we train models for 3000 epochs, and save the three models at 1000, 2000, and 3000 epochs. Furthermore, we choose the model that performs best on the training set from the three models to evaluate on the testing circuits. To further enhance our method, we can leverage the model selection strategies investigated in (Gulrajani & Lopez-Paz), which we leave as future work.

Table 8. We analyze the runtime of commonly used LS heuristics on open-source circuits. Ratio denotes the ratio of the runtime to that of the Rewrite heuristic.

	Log2		Hyp		Multiplier	
heuristics	RunTime (s)	Ratio	RunTime (s)	Ratio	RunTime (s)	Ratio
Rewrite	1.73 (0.09)	1.00	29.97 (0.33)	1.00	0.74 (0.014)	1.00
Balance	0.04 (0.0)	0.02	0.36 (0.035)	0.01	0.03 (0.0)	0.04
Refactor	2.12 (0.04)	1.23	24.01 (0.07)	0.80	1.72 (0.02)	2.32
Resub (K=12)	4.33 (0.042)	2.50	9.17 (0.04)	0.31	1.81 (0.02)	2.45
Resub (K=16)	76.44 (0.55)	44.18	116.14 (0.10)	3.88	23.33 (0.05)	31.53
Mfs2	128.25 (0.33)	74.13	259.07 (0.33)	8.64	13.70 (0.07)	18.51
	Sin		Square		Vga lcd	
heuristics	RunTime (s)	Ratio	RunTime (s)	Ratio	RunTime (s)	Ratio
Rewrite	0.23 (0.0)	1.00	0.50 (0.04)	1.00	6.99 (0.22)	1.00
Balance	0.01 (0.0)	0.04	0.02 (0.0)	0.04	0.38 (0.022)	0.05
Refactor	0.21 (0.016)	0.91	1.19 (0.02)	2.38	7.83 (0.12)	1.12
Resub (K=12)	0.90 (0.0)	3.91	1.03 (0.005)	2.06	97.32 (0.06)	13.92
Resub (K=16)	17.05 (0.016)	74.13	11.87 (0.08)	23.74	649.19 (1.16)	92.87
Mfs2	10.15 (0.06)	44.13	25.33 (0.41)	50.66	128.55 (1.08)	18.39
	Ethernet		Wb conmax		Des perf	
heuristics	RunTime (s)	Ratio	RunTime (s)	Ratio	RunTime (s)	Ratio
Rewrite	1.51 (0.03)	1.00	0.86 (0.03)	1.00	1.86 (0.04)	1.00
Balance	0.13 (0.005)	0.09	0.07 (0.009)	0.08	0.16 (0.016)	0.09
Refactor	1.06 (0.009)	0.70	0.59 (0.016)	0.69	1.37 (0.09)	0.74
Resub (K=12)	10.72 (0.02)	7.10	7.96 (0.04)	9.26	23.37 (0.27)	12.56
Resub (K=16)	145.77 (0.19)	96.54	149.80 (0.38)	174.19	223.09 (1.44)	119.94
Mfs2	27.53 (0.15)	18.23	25.40 (0.04)	29.53	30.11 (0.065)	16.19

Avg Time Ratio to Rewrite						
heuristics	Rewrite	Balance	Refactor	Resub (K=12)	Resub (K=16)	Mfs2
Time Ratio	1.00	0.05	1.21	**6.01**	**73.44**	**30.94**

Table 9. We analyze the runtime of commonly used LS heuristics on industrial circuits. Ratio denotes the ratio of the runtime to that of the Rewrite heuristic.

	f5022		f8272		c5088	
heuristics	RunTime (s)	Ratio	RunTime (s)	Ratio	RunTime (s)	Ratio
Rewrite	36.06 (0.40)	1.00	29.41 (0.15)	1.00	25.05 (0.10)	1.00
Balance	0.75 (0.005)	0.02	0.74 (0.04)	0.03	0.76 (0.005)	0.03
Refactor	42.24 (0.10)	1.17	24.65 (0.08)	0.84	31.51 (0.33)	1.26
Resub (K=12)	228.92 (2.11)	6.35	517.74 (27.07)	17.60	336.13 (22.22)	13.42
Resub (K=16)	1144.56 (10.57)	31.74	743.51 (4.53)	25.28	1660.70 (7.03)	66.30
Mfs2	177.93 (0.69)	4.93	76.97 (0.13)	2.62	487.51 (51.68)	19.46
	d3151		c8449		d4067	
heuristics	RunTime (s)	Ratio	RunTime (s)	Ratio	RunTime (s)	Ratio
Rewrite	0.94 (0.012)	1.00	1.44 (0.02)	1.00	33.52 (0.15)	1.00
Balance	0.03 (0.0)	0.03	0.04 (0.0)	0.03	1.16 (0.008)	0.03
Refactor	0.80 (0.008)	0.85	1.43 (0.008)	0.99	44.53 (0.12)	1.33
Resub (K=12)	6.13 (0.12)	6.52	8.03 (0.49)	5.58	1061.05 (6.96)	31.65
Resub (K=16)	61.22 (0.03)	65.13	72.80 (0.008)	50.56	5300.23 (20.44)	158.12
Mfs2	66.17 (5.25)	70.39	234.58 (45.54)	162.90	201.43 (12.17)	6.01

Avg Time Ratio to Rewrite						
heuristics	Rewrite	Balance	Refactor	Resub (K=12)	Resub (K=16)	Mfs2
Time Ratio	1.00	0.03	1.07	**13.52**	**66.19**	**44.39**

Table 10. We analyze the runtime of the Rewrite and Refactor heuristics on very large-scale circuits.

Sixteen		Twenty	
heuristics	RunTime (s)	heuristics	RunTime (s)
Rewrite	8502.77 (297.2)	Rewrite	12715.52 (52.83)
Refactor	7486.48 (212.83)	Refactor	10297.4 (42.0)

Table 11. We analyze the runtime percentage of the Mfs2 and Resub heuristics over common LS optimization sequences.

Mfs2 heuristic			
Circuit	RunTime of Mfs2	Total RunTime	Percent (%)
Log2	155.33 (1.31)	175.46 (2.26)	88.53
Hyp	263.91 (0.36)	272.52 (0.38)	96.84
Multiplier	16.69 (0.07)	23.27 (0.03)	71.72
Sin	12.64 (0.27)	15.34 (0.29)	82.40
Square	21.41 (0.07)	26.6 (0.11)	80.49
Des perf	29.51 (0.28)	51.54 (0.3)	57.26
Ethernet	27.73 (0.46)	49.12 (0.85)	56.45
Wb conmax	21.24 (0.54)	31.32 (0.57)	67.82
Vga lcd	189.1 (3.37)	261.58 (4.42)	72.29
Average	-	-	**74.87**
Resub heuristic			
Circuit	RunTime of Resub	Total RunTime	Percent (%)
Log2	65.87 (1.44)	73.82 (2.05)	89.23
Hyp	115.86 (0.17)	235.3 (2.49)	49.24
Multiplier	19.91 (0.06)	25.25 (0.02)	78.85
Sin	12.74 (0.02)	13.73 (0.01)	92.79
Square	9.37 (0.13)	13.93 (0.2)	67.26
Des perf	228.12 (13.89)	237.71 (14.09)	95.97
Ethernet	83.54 (0.27)	88.45 (0.37)	94.45
Wb conmax	102.31 (0.25)	106.14 (0.47)	96.39
Vga lcd	403.3 (1.87)	415.16 (2.4)	97.14
Average	-	-	**84.59**

Table 12. We report the recall and optimization performance of the Resub heuristic incorporated with Random models. And Reduction denotes the reduced number of nodes, i.e., optimization performance. Percent denotes the hyperparameter k, i.e., the percent of nodes to apply transformations. Normalized AR denotes the ratio of the AR to that of the default heuristic.

Log2				Hyp			
Percent	Recall	And Reduction (AR)	Normalized AR	Percent	Recall	And Reduction (AR)	Normalized AR
0.10	0.06	8.00 (2.0)	0.07	0.10	0.11	702.00 (16.31)	0.10
0.20	0.15	16.00 (0.0)	0.15	0.20	0.20	1362.67 (21.23)	0.20
0.30	0.25	37.50 (0.5)	0.34	0.30	0.30	2027.00 (8.52)	0.30
0.40	0.31	35.00 (2.0)	0.32	0.40	0.39	2667.33 (30.65)	0.39
0.50	0.48	52.00 (1.0)	0.47	0.50	0.50	3407.33 (38.76)	0.50
0.60	0.60	64.50 (2.5)	0.59	0.60	0.59	4053.33 (25.72)	0.60
0.70	0.72	80.50 (2.5)	0.73	0.70	0.69	4733.33 (53.32)	0.70
0.80	0.81	85.00 (1.0)	0.77	0.80	0.81	5423.33 (21.31)	0.80
0.90	0.87	99.50 (2.5)	0.90	0.90	0.91	6108.33 (20.00)	0.90
1.00	1.00	110.00 (0.0)	1.00	1.00	1.00	6797.00 (0.0)	1.00
Multiplier				Square			
Percent	Recall	And Reduction (AR)	Normalized AR	Percent	Recall	And Reduction (AR)	Normalized AR
0.10	0.13	8.00 (0.81)	0.07	0.10	0.12	84.33 (6.23)	0.11
0.20	0.28	22.00 (1.41)	0.18	0.20	0.15	154.66 (9.39)	0.20
0.30	0.40	36.00 (4.32)	0.30	0.30	0.25	239.00 (15.12)	0.31
0.40	0.47	45.00 (2.44)	0.38	0.40	0.37	311.00 (8.28)	0.40
0.50	0.58	58.33 (1.24)	0.49	0.50	0.47	404.33 (14.88)	0.52
0.60	0.69	72.66 (2.35)	0.61	0.60	0.55	463.00 (4.96)	0.60
0.70	0.78	81.66 (0.94)	0.68	0.70	0.66	550.00 (11.04)	0.71
0.80	0.84	94.33 (2.86)	0.79	0.80	0.76	629.00 (1.63)	0.81
0.90	0.97	107.66 (2.49)	0.90	0.90	0.86	696.33 (5.31)	0.90
1.00	1.00	120.00 (0.0)	1.00	1.00	1.00	778.00 (0.0)	1.00
Des perf				Vga lcd			
Percent	Recall	And Reduction (AR)	Normalized AR	Percent	Recall	And Reduction (AR)	Normalized AR
0.10	0.10	292.00 (8.98)	0.13	0.10	0.10	7.66 (3.68)	0.09
0.20	0.21	505.00 (3.56)	0.22	0.20	0.15	22.33 (2.49)	0.26
0.30	0.30	775.00 (6.53)	0.34	0.30	0.20	32.00 (3.74)	0.38
0.40	0.41	1031.33 (27.34)	0.45	0.40	0.34	37.00 (6.97)	0.44
0.50	0.51	1281.00 (18.02)	0.56	0.50	0.46	50.66 (5.18)	0.60
0.60	0.60	1467.67 (22.23)	0.64	0.60	0.54	60.33 (3.09)	0.71
0.70	0.71	1738.67 (22.16)	0.75	0.70	0.61	69.00 (4.54)	0.81
0.80	0.81	1913.33 (12.92)	0.83	0.80	0.80	77.00 (3.74)	0.91
0.90	0.90	2124.00 (4.55)	0.92	0.90	0.90	79.33 (3.85)	0.93
1.00	1.00	2307.00 (0.0)	1.00	1.00	1.00	85.00 (0.0)	1.00
Ethernet				Wb conmax			
Percent	Recall	And Reduction (AR)	Normalized AR	Percent	Recall	And Reduction (AR)	Normalized AR
0.10	0.13	26.33 (1.69)	0.13	0.10	0.11	325.00 (17.8)	0.16
0.20	0.26	52.66 (9.80)	0.26	0.20	0.20	596.67 (28.52)	0.29
0.30	0.34	70.33 (16.54)	0.34	0.30	0.30	879.67 (27.01)	0.43
0.40	0.47	110.66 (15.54)	0.54	0.40	0.39	1042.33 (21.36)	0.51
0.50	0.58	125.00 (9.09)	0.61	0.50	0.49	1310.33 (20.89)	0.64
0.60	0.67	138.33 (2.05)	0.68	0.60	0.60	1502.33 (3.09)	0.74
0.70	0.77	161.00 (9.89)	0.79	0.70	0.70	1660.00 (53.39)	0.81
0.80	0.82	180.33 (3.39)	0.88	0.80	0.79	1789.33 (8.38)	0.88
0.90	0.92	191.00 (8.64)	0.94	0.90	0.90	1946.33 (8.26)	0.96
1.00	1.00	204.00 (0.0)	1.00	1.00	1.00	2037.00 (0.0)	1.00

Table 13. We report the number of applied transformations and runtime of the Resub heuristic incorporated with a Random model. Percent denotes the hyperparameter k, i.e., the percent of nodes to apply transformations. Normalized RT denotes the ratio of the runtime to that of the default heuristic.

Log2			Hyp		
Percent	Run Time (RT, s)	Normalized RT	Percent	Run Time (RT, s)	Normalized RT
0.10	6.99 (0.24)	0.11	0.10	14.64 (0.03)	0.13
0.20	13.56 (0.07)	0.21	0.20	26.73 (0.07)	0.23
0.30	19.24 (0.36)	0.30	0.30	37.93 (0.30)	0.33
0.40	26.06 (0.17)	0.41	0.40	49.30 (0.21)	0.43
0.50	31.40 (0.13)	0.50	0.50	60.50 (0.11)	0.53
0.60	37.80 (0.15)	0.60	0.60	71.82 (0.19)	0.63
0.70	44.42 (0.42)	0.70	0.70	82.42 (0.43)	0.72
0.80	50.84 (0.42)	0.81	0.80	92.93 (0.81)	0.82
0.90	57.32 (0.62)	0.91	0.90	103.25 (0.65)	0.91
1.00	63.13 (0.5)	1.00	1.00	113.98 (0.71)	1.00
Multiplier			Square		
Percent	Run Time (RT, s)	Normalized RT	Percent	Run Time (RT, s)	Normalized RT
0.10	2.36 (0.04)	0.12	0.10	1.15 (0.071)	0.13
0.20	4.32 (0.04)	0.22	0.20	2.15 (0.06)	0.23
0.30	6.35 (0.06)	0.32	0.30	2.86 (0.03)	0.31
0.40	8.06 (0.10)	0.41	0.40	3.89 (0.11)	0.43
0.50	10.41 (0.23)	0.53	0.50	4.74 (0.05)	0.52
0.60	12.12 (0.15)	0.61	0.60	5.58 (0.05)	0.61
0.70	13.99 (0.04)	0.71	0.70	6.50 (0.04)	0.71
0.80	15.92 (0.07)	0.80	0.80	7.39 (0.04)	0.81
0.90	17.84 (0.13)	0.90	0.90	8.25 (0.05)	0.90
1.00	19.79 (0.02)	1.00	1.00	9.15 (0.005)	1.00
Des perf			Vga lcd		
Percent	Run Time (RT, s)	Normalized RT	Percent	Run Time (RT, s)	Normalized RT
0.10	37.01 (0.41)	0.17	0.10	67.04 (1.38)	0.17
0.20	57.51 (0.77)	0.27	0.20	103.03 (1.72)	0.26
0.30	76.87 (0.64)	0.36	0.30	138.10 (0.51)	0.35
0.40	97.43 (0.15)	0.46	0.40	173.13 (0.89)	0.44
0.50	116.11 (0.31)	0.55	0.50	211.32 (1.82)	0.54
0.60	135.97 (0.88)	0.64	0.60	246.12 (3.85)	0.63
0.70	154.60 (1.53)	0.73	0.70	283.42 (2.39)	0.73
0.80	174.10 (1.04)	0.82	0.80	318.32 (2.08)	0.82
0.90	193.21 (1.37)	0.91	0.90	352.70 (1.53)	0.91
1.00	211.95 (1.39)	1.00	1.00	389.20 (2.91)	1.00
Ethernet			Wb conmax		
Percent	Run Time (RT, s)	Normalized RT	Percent	Run Time (RT, s)	Normalized RT
0.10	16.72 (0.40)	0.21	0.10	12.03 (0.4)	0.12
0.20	24.12 (0.36)	0.30	0.20	22.30 (0.4)	0.22
0.30	31.21 (0.53)	0.38	0.30	33.03 (0.26)	0.33
0.40	39.16 (1.04)	0.48	0.40	42.97 (0.62)	0.42
0.50	45.29 (1.09)	0.56	0.50	53.14 (0.4)	0.52
0.60	52.99 (0.45)	0.65	0.60	62.52 (0.35)	0.62
0.70	59.35 (0.19)	0.73	0.70	72.15 (0.13)	0.71
0.80	66.56 (0.97)	0.82	0.80	82.14 (0.21)	0.81
0.90	73.85 (0.35)	0.91	0.90	91.82 (0.25)	0.91
1.00	81.34 (0.78)	1.00	1.00	101.43 (0.16)	1.00

Table 14. The results show that only applying transformations on effective nodes (i.e., Orcale) significantly improves the runtime while keeping the same size and depth of the Log2 circuit. Note that Nd denotes the number of nodes on the circuit, i.e., the size of the circuit. Lev denotes the depth of the circuit.

Log2					
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
Oracle	10621.00	**0.00**	104.00	5.40	**96.54**
Default (Mfs2)	10621.00 (0.0)	NA	104.00 (0.0)	156.19 (1.66)	NA

Table 15. Detailed offline evaluation results using Evaluation Strategy 1 on the Mfs2 heuristic and open-source circuits.

	Log2			Hyp			Multiplier		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.89 (0.04)**	**0.97 (0.01)**	**0.99 (0.01)**	**0.79 (0.04)**	**0.90 (0.03)**	**0.95 (0.008**	**0.91 (0.03)**	**0.96 (0.03)**	**0.98 (0.01)**
EnsembleMLP	0.61 (0.0)	0.65 (0.007)	0.85 (0.03)	0.62 (0.005)	0.73 (0.004)	0.82 (0.003)	0.63 (0.0)	0.71 (0.0)	0.75 (0.0)
Random	0.31	0.48	0.60	0.39	0.50	0.59	0.47	0.58	0.69
	Sin			Square			Vga lcd		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.71 (0.06)**	**0.87 (0.04)**	**0.95 (0.02)**	**0.90 (0.01)**	**0.94 (0.002)**	**0.95 (0.0)**	**0.93 (0.007)**	**0.95 (0.01)**	**0.98 (0.007)**
EnsembleMLP	0.56 (0.0)	0.58 (0.0)	0.67 (0.0)	0.85 (0.02)	0.91 (0.004)	0.92 (0.0)	0.19 (0.04)	0.29 (0.02)	0.53 (0.02)
Random	0.47	0.64	0.72	0.37	0.47	0.55	0.34	0.46	0.54
	Ethernet			Wb conmax			Des perf		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.91 (0.09)**	**0.96 (0.04)**	**0.97 (0.03)**	**0.69 (0.04)**	**0.81 (0.05)**	**0.89 (0.04)**	**0.53 (0.0006)**	**0.68 (0.0007)**	**0.82 (0.0004)**
EnsembleMLP	0.53 (0.004)	0.62 (0.01)	0.63 (0.03)	0.43 (0.07)	0.57 (0.05)	0.68 (0.03)	0.44 (0.07)	0.49 (0.06)	0.58 (0.03)
Random	0.47	0.58	0.67	0.39	0.49	0.60	0.41	0.51	0.60

Table 16. Detailed offline evaluation results using Evaluation Strategy 1 on the Resub heuristic and open-source circuits.

	Log2			Hyp			Multiplier		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.66 (0.02)**	**0.88 (0.02)**	**0.97 (0.004)**	**0.88 (0.008)**	**0.91 (0.02)**	0.94 (0.03)	0.74 (0.17)	**0.97 (0.02)**	**0.99 (0.003)**
EnsembleMLP	0.39 (0.1)	0.57 (0.2)	0.62 (0.20)	0.72 (0.02)	0.86 (0.005)	**0.97 (0.01)**	**0.88 (0.12)**	0.93 (0.08)	0.99 (0.01)
Random	0.39	0.48	0.63	0.40	0.50	0.60	0.34	0.43	0.58
	Sin			Square			Vga lcd		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	0.46 (0.17)	**0.69 (0.07)**	**0.76 (0.07)**	**0.60 (0.11)**	**0.73 (0.08)**	**0.81 (0.07)**	0.47 (0.34)	**0.77 (0.04)**	**0.94 (0.02)**
EnsembleMLP	**0.48 (0.03)**	0.63 (0.1)	0.76 (0.07)	0.39 (0.09)	0.52 (0.12)	0.63 (0.08)	**0.68 (0.03)**	0.71 (0.01)	0.87 (0.01)
Random	0.42	0.54	0.57	0.38	0.48	0.59	0.34	0.46	0.54
	Ethernet			Wb conmax			Des perf		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG	**0.82 (0.02)**	**0.98 (0.009)**	**0.99 (0.0)**	**0.66 (0.07)**	**0.88 (0.08)**	**0.96 (0.01)**	**0.55 (0.07)**	**0.72 (0.05)**	**0.89 (0.009)**
EnsembleMLP	0.61 (0.01)	0.82 (0.04)	0.87 (0.05)	0.18 (0.26)	0.18 (0.26)	0.40 (0.11)	0.36 (0.04)	0.46 (0.03)	0.56 (0.03)
Random	0.47	0.58	0.67	0.39	0.49	0.60	0.41	0.51	0.60

Table 17. Detailed offline evaluation results using Evaluation Strategy 2 on the Mfs2 and Resub heuristics and open-source circuits.

		Log2	Hyp	Multiplier	Sin	Square
heuristic	Method	top 50% acc	top 50% acc	top 50% acc	top 50% acc	top 50% acc
mfs2	COG	**0.88 (0.02)**	**0.85 (0.06)**	**0.87 (0.04)**	**0.79 (0.12)**	**0.58 (0.05)**
	EnsembleMLP	0.08 (0.007)	0.78 (0.01)	0.33 (0.01)	0.65 (0.08)	0.45 (0.007)
	Random	0.48	0.50	0.58	0.54	0.47
resub	COG	**0.80 (0.005)**	**0.87 (0.03)**	**0.87 (0.0)**	**0.81 (0.03)**	0.89 (0.01)
	EnsembleMLP	0.21 (0.02)	0.67 (0.12)	0.71 (0.06)	0.28 (0.0)	**0.92 (0.008)**
	Random	0.48	0.50	0.43	0.54	0.48

Table 18. Detailed online evaluation results using Evaluation Strategy 1 on the Mfs2 heuristic and open-source circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

	Log2					Hyp					Multiplier				
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	10621.00 (0.0)	0.00	104.00 (0.0)	137.87 (3.96)	**11.73**	63819.00 (58.65)	-0.37	8259.00 (0.0)	149.97 (19.89)	**45.30**	7800.33 (1.24)	-0.02	87.00 (0.0)	9.04 (0.37)	**45.24**
Default (Mfs2)	10621.00 (0.0)	NA	104.00 (0.0)	156.19 (1.66)	NA	63581.00 (0.0)	NA	8259.00 (0.0)	274.13 (9.90)	NA	7799.00 (0.0)	NA	87.00 (0.0)	16.69 (0.07)	NA
	Sin					Square					Vga lcd				
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	1991.00 (0.81)	-0.05	53.00 (0.0)	10.52 (0.41)	**15.11**	5701.67 (0.47)	-0.01	83.00 (0.0)	9.30 (0.43)	**56.61**	38328.00 (0.0)	-0.01	7.00 (0.0)	96.16 (3.02)	**49.15**
Default (Mfs2)	1990.00 (0.0)	NA	53.00 (0.0)	12.38 (0.016)	NA	5701.00 (0.0)	NA	83.00 (0.0)	21.41 (0.071)	NA	38326.00 (0.0)	NA	7.00 (0.0)	189.10 (3.37)	NA
	Ethernet					Wb conmax					Des perf				
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	13639.33 (0.47)	-0.01	9.00 (0.0)	11.42 (0.48)	**58.82**	16813.33 (87.64)	-1.84	9.00 (0.0)	10.41 (0.25)	**51.04**	31139.00 (7.78)	-0.93	6.00 (0.0)	18.98 (0.25)	**35.68**
Default (Mfs2)	13638.00 (0.0)	NA	9.00 (0.0)	27.73 (0.45)	NA	16509.00 (0.0)	NA	9.00 (0.0)	21.24 (0.53)	NA	30853.00 (0.0)	NA	6.00 (0.0)	29.51 (0.28)	NA

Table 19. Detailed online evaluation results using Evaluation Strategy 1 on the Resub heuristic and open-source circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

	Log2					Hyp					Multiplier				
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	29271.00 (3.74)	-0.04	376.00 (0.0)	35.73 (0.72)	**44.92**	205183.33 (177.48)	-0.32	24794.00 (0.0)	37.34 (0.79)	**67.77**	24440.00 (3.26)	-0.02	262.00 (0.0)	11.07 (0.68)	**44.90**
Default (Resub)	29260.00 (0.0)	NA	376.00 (0.0)	64.87 (0.86)	NA	204533.00 (0.0)	NA	24792.00 (0.0)	115.86 (0.16)	NA	24436.00 (0.0)	NA	262.00 (0.0)	20.09 (0.10)	NA
	Sin					Square					Vga lcd				
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	5025.67 (1.69)	-0.15	177.00 (0.0)	8.15 (0.67)	**37.88**	16053.67 (73.05)	-1.32	248.00 (0.0)	4.92 (0.22)	**47.49**	90870.67 (11.78)	-0.08	19.00 (0.0)	242.76 (83.81)	**39.81**
Default (Resub)	5018.00 (0.0)	NA	177.00 (0.0)	13.12 (0.12)	NA	15845.00 (0.0)	NA	248.00 (0.0)	9.37 (0.13)	NA	90795.00 (0.0)	NA	19.00 (0.0)	403.30 (1.86)	NA
	Ethernet					Wb conmax					Des perf				
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	43415.00 (64.65)	-0.16	26.00 (0.0)	55.27 (10.02)	**33.84**	39277.00 (120.14)	-0.39	18.00 (0.0)	85.87 (1.12)	**16.07**	68458.00 (114.77)	-1.88	16.00 (0.0)	110.89 (10.31)	**51.39**
Default (Resub)	43345.00 (0.0)	NA	26.00 (0.0)	83.54 (0.27)	NA	39126.00 (0.0)	NA	18.00 (0.0)	102.31 (0.24)	NA	67193.00 (0.0)	NA	16.00 (0.0)	228.12 (13.89)	NA

Table 20. Detailed online evaluation results using Evaluation Strategy 2 on the Mfs2 and Resub heuristics and open-source circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

Log2

Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	10621.67 (0.47)	-0.01	104.00 (0.0)	88.70 (17.85)	**42.90**	PruneX (COG)	29284.00 (2.0)	-0.08	376.00 (0.0)	40.03 (1.57)	**39.23**
Default (Mfs2)	10621.00 (0.0)	NA	104.00 (0.0)	155.33 (1.31)	NA	Default (Resub)	29260.00 (0.0)	NA	376.00 (0.0)	65.87 (1.43)	NA

Hyp

Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	63682.00 (18.23)	-0.16	8259.00 (0.0)	210.52 (7.99)	**20.23**	PruneX (COG)	205436.50 (216.5)	-0.44	24793.00 (0.0)	43.01 (5.01)	**62.61**
Default (Mfs2)	63581.00 (0.0)	NA	8259.00 (0.0)	263.91 (0.36)	NA	Default (Resub)	204533.00 (0.0)	NA	24792.00 (0.0)	115.07 (0.009)	NA

Multiplier

Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	7799.00 (0.0)	0.00	87.00 (0.0)	14.67 (0.43)	**16.41**	PruneX (COG)	24452.50 (0.5)	-0.07	262.00 (0.0)	12.35 (0.14)	**37.97**
Default (Mfs2)	7799.00 (0.0)	NA	87.00 (0.0)	17.55 (0.16)	NA	Default (Resub)	24436.00 (0.0)	NA	262.00 (0.0)	19.91 (0.06)	NA

Sin

Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	1992.33 (0.94)	-0.12	53.00 (0.0)	9.66 (1.75)	**23.58**	PruneX (COG)	5019.50 (0.5)	-0.03	177.00 (0.0)	7.85 (0.64)	**38.38**
Default (Mfs2)	1990.00 (0.0)	NA	53.00 (0.0)	12.64 (0.27)	NA	Default (Resub)	5018.00 (0.0)	NA	177.00 (0.0)	12.74 (0.024)	NA

Square

Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	5703.33 (1.24)	-0.04	83.00 (0.0)	15.15 (0.96)	**30.18**	PruneX (COG)	15918.50 (12.5)	-0.46	248.00 (0.0)	5.14 (0.1)	**43.89**
Default (Mfs2)	5701.00 (0.0)	NA	83.00 (0.0)	21.70 (0.32)	NA	Default (Resub)	15845.00 (0.0)	NA	248.00 (0.0)	9.16 (0.009)	NA

Table 21. Detailed offline evaluation results on the Mfs2 heuristic and industrial circuits.

	ci3			ci1		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG (Ours)	**0.78 (0.01)**	**0.85 (0.006)**	**0.90 (0.006)**	**1.00 (0.0001)**	**1.00 (0.0)**	**1.00 (0.0)**
EnsembleMLP (Ours)	0.43 (0.002)	0.54 (0.008)	0.69 (0.008)	1.00 (0.00006)	1.00 (0.00006)	1.00 (0.00006)
Random	0.40	0.50	0.60	0.40	0.50	0.60
	ci4			ci6		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG (Ours)	**0.78 (0.01)**	**0.85 (0.01)**	**0.90 (0.02)**	**0.71 (0.003)**	**0.84 (0.02)**	**0.91 (0.02)**
EnsembleMLP (Ours)	0.50 (0.003)	0.61 (0.009)	0.68 (0.004)	0.48 (0.00003)	0.66 (0.0004)	0.75 (0.00003)
Random	0.42	0.52	0.61	0.40	0.50	0.60
	ci5			ci2		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG (Ours)	**0.82 (0.003)**	**0.83 (0.002)**	**0.84 (0.01)**	**0.89 (0.008)**	**0.89 (0.01)**	**0.92 (0.02)**
EnsembleMLP (Ours)	**0.86 (0.007)**	**0.94 (0.0004)**	**0.95 (0.0008)**	**0.98 (0.0)**	**0.98 (0.0)**	**0.99 (0.004)**
Random	0.36	0.48	0.58	0.33	0.47	0.61

Table 22. Detailed offline evaluation results on the Resub heuristic and industrial circuits.

	ci3			ci1		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG (Ours)	**0.91 (0.02)**	**0.94 (0.008)**	**0.96 (0.01)**	**0.89 (0.14)**	**1.00 (0.0)**	**1.00 (0.0)**
EnsembleMLP (Ours)	0.39 (0.03)	0.43 (0.04)	0.53 (0.07)	0.31 (0.22)	0.67 (0.47)	0.68 (0.45)
Random	0.40	0.50	0.60	0.40	0.50	0.60
	ci4			ci6		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG (Ours)	**0.63 (0.01)**	**0.76 (0.02)**	**0.79 (0.002)**	**0.84 (0.02)**	**0.85 (0.03)**	**0.88 (0.02)**
EnsembleMLP (Ours)	0.41 (0.02)	0.50 (0.02)	0.59 (0.013)	0.25 (0.01)	0.33 (0.02)	0.45 (0.016)
Random	0.42	0.52	0.61	0.40	0.50	0.60
	ci5			ci2		
Method	top 40% acc	top 50% acc	top 60% acc	top 40% acc	top 50% acc	top 60% acc
COG (Ours)	**0.99 (0.001)**	**1.00 (0.001)**	**1.00 (0.0005)**	**0.98 (0.0004)**	**0.98 (0.0008)**	**0.98 (0.001)**
EnsembleMLP (Ours)	0.56 (0.22)	0.61 (0.19)	0.63 (0.18)	0.17 (0.09)	0.20 (0.11)	0.33 (0.20)
Random	0.36	0.48	0.58	0.33	0.47	0.61

Table 23. Detailed online evaluation results on the Mfs2 heuristic and industrial circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

	ci1					ci6				
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	165538.00 (0.0)	0.00	47.00 (0.0)	93.58 (0.49)	**47.41**	99258.00 (0.0)	-0.01	12.00 (0.0)	31.71 (1.17)	**58.80**
Default (Mfs2)	165538.00 (0.0)	NA	47.00 (0.0)	177.93 (0.69)	NA	99245.00 (0.0)	NA	12.00 (0.0)	76.97 (0.13)	NA
	ci2					ci3				
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	195670.33 (1.24)	-0.003	35.00 (0.0)	212.10 (5.13)	**56.49**	6648.33 (2.49)	-0.22	15.00 (0.0)	51.58 (2.55)	**22.05**
Default (Mfs2)	195665.00 (0.0)	NA	35.00 (0.0)	487.51 (51.68)	NA	6634.00 (0.0)	NA	15.00 (0.0)	66.17 (5.25)	NA
	ci4					ci5				
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	9612.66 (17.55)	-0.78	17.00 (0.0)	169.65 (40.92)	**27.68**	215714.00 (2.82)	-0.003	26.00 (0.0)	162.59 (16.74)	**19.28**
Default (Mfs2)	9538.00 (0.0)	NA	17.00 (0.0)	234.58 (45.54)	NA	215708.00 (0.0)	NA	26.00 (0.0)	201.43 (12.17)	NA

Table 24. Detailed online evaluation results on the Resub heuristic and industrial circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

	ci1					ci6				
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	474604.66 (6.5)	-0.001	141.0 (0.0)	793.10 (110.67)	**30.71**	278680.33 (1.88)	-0.015	31.0 (0.0)	370.89 (4.00)	**50.12**
Default (Resub)	474600.0 (0.0)	NA	141.0 (0.0)	1144.56 (10.57)	NA	278637.0 (0.0)	NA	31.0 (0.0)	743.51 (4.53)	NA
	ci2					ci3				
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	506986.66 (15.36)	-0.003	173.0 (0.0)	1908.63 (692.66)	**24.09**	18531.33 (29.16)	-0.46	57.0 (0.0)	13.51 (0.33)	**46.81**
Default (Resub)	506972.0 (0.0)	NA	173.0 (0.0)	2514.46 (377.94)	NA	18447.0 (0.0)	NA	57.0 (0.0)	25.40 (2.94)	NA
	ci4					ci5				
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	27834.33 (20.53)	-0.50	65.0 (0.0)	17.33 (2.84)	**50.74**	643664.66 (76.85)	-0.22	100.0 (0.0)	1638.94 (255.20)	**67.60**
Default (Resub)	27695.0 (0.0)	NA	65.0 (0.0)	35.18 (0.15)	NA	642243.0 (0.0)	NA	95.0 (0.0)	5058.70 (51.38)	NA

Table 25. Detailed offline evaluation results on the Mfs2 heuristic and very large-scale circuits.

	Sixteen		
Method	top 40% acc	top 50% acc	top 60% acc
COG	**1.00 (9e-05)**	**1.00 (4e-05)**	**1.00 (8e-05)**
EnsembleMLP	0.34 (0.03)	0.44 (0.03)	0.53 (0.02)
Random	0.31	0.48	0.60
	Twenty		
Method	top 40% acc	top 50% acc	top 60% acc
COG	**1.00 (4e-05)**	**1.00 (0.0001)**	**1.00 (0.0001)**
EnsembleMLP	0.54 (0.17)	0.61 (0.15)	0.68 (0.11)
Random	0.40	0.50	0.60

Table 26. Detailed offline evaluation results on the Resub heuristic and very large-scale circuits.

	Sixteen		
Method	top 40% acc	top 50% acc	top 60% acc
COG	**0.89 (0.008)**	**0.94 (0.007)**	**0.97 (0.004)**
EnsembleMLP	0.42 (0.06)	0.54 (0.09)	0.67 (0.13)
Random	0.31	0.48	0.60
	Twenty		
Method	top 40% acc	top 50% acc	top 60% acc
COG	**0.91 (0.006)**	**0.95 (0.003)**	**0.98 (0.001)**
EnsembleMLP	0.41 (0.0)	0.52 (0.0)	0.61 (0.0)
Random	0.42	0.48	0.62

Table 27. Detailed online evaluation results on the Mfs2 heuristic and very large-scale circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

Sixteen					
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	6017637.0 (0.81)	-9.9E-05	48.0 (0.0)	38581.40 (933.59)	**27.43**
Default (Mfs2)	6017631.0 (0.0)	NA	48.0 (0.0)	53162.36 (337.79)	NA
Twenty					
Method	Nd	Decrease (Nd, %)	Lev	Time (s)	Improvement (Time, %)
PruneX (COG)	7693099.0 (0.0)	-0.0001	54.0 (0.0)	54046.50 (851.80)	**42.00**
Default (Mfs2)	7693089.0 (0.0)	NA	54.0 (0.0)	93189.12 (8806.55)	NA

Table 28. Detailed online evaluation results on the Resub heuristic and very large-scale circuits. Improvement denotes the improvement of our PruneX compared to the default heuristic.

Sixteen					
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	11972490.66 (267.26)	-0.005	99.0 (0.0)	11166.83 (374.72)	**16.35**
Default (Resub)	11971930.0 (0.0)	NA	99.0 (0.0)	13349.62 (175.76)	NA
Twenty					
Method	And	Decrease (And, %)	Lev	Time (s)	Improvement (Time, %)
PruneX-COG (Ours)	15310801.33 (214.67)	-0.0001	86.0 (0.0)	14114.51 (881.11)	**18.17**
Default (Resub)	15310242.0 (0.0)	NA	86.0 (0.0)	17249.11 (90.69)	NA

Table 29. We provide more online evaluation results of improving QoR with our proposed PruneX-COG. Improvement denotes the improvement of our PruneX compared to the default heuristic.

	Multiplier				Log2			
Method	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)
Default (Mfs2)	7799.00 (0.0)	NA	16.69 (0.07)	NA	10621 (0.0)	NA	156.19(1.66)	NA
2PruneX-COG (0.3, Ours)	7658.00 (4.32)	**1.81**	8.86 (0.87)	**46.91**	10556.00 (3.74)	**0.61**	179.04 (6.16)	**-14.63**
2PruneX-COG (0.4, Ours)	7655.00 (4.54)	**1.85**	14.09 (1.07)	**15.58**	5523.33 (5.43)	**3.12**	237.79 (8.28)	**-52.24**
	Sin				Ethernet			
Method	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)
Default (Mfs2)	1990.00 (0.0)	NA	12.38 (0.016)	NA	13638.00 (0.0)	NA	27.73 (0.45)	NA
2PruneX-COG (0.3, Ours)	1985.00 (0.0)	**0.25**	11.65 (0.06)	**5.90**	13514.33 (4.71)	**0.91**	8.87 (1.47)	**68.01**
2PruneX-COG (0.4, Ours)	1980.66.00 (0.94)	**0.47**	16.23 (0.06)	**-31.10**	13511.00 (0.81)	**0.93**	15.44 (0.76)	**44.32**
	Square				ci1			
Method	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Lev ↓	Improvement ↑ (Lev, %)	Time (s) ↓	Improvement ↑ (Time, %)
Default (Mfs2)	5701.00 (0.0)	NA	21.41 (0.071)	NA	47.00 (0.0)	NA	177.93 (0.69)	NA
2PruneX-COG (0.3, Ours)	5550.33 (11.67)	**2.64**	9.94 (0.55)	**53.57**	45 (0.0)	**4.26**	148.87 (19.79)	**16.33**
2PruneX-COG (0.4, Ours)	5523.33 (5.43)	**3.12**	14.23 (0.66)	**33.54**	45.00 (0.0)	**4.26**	182.35 (25.45)	-2.48
	ci4				ci3			
Method	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)	Nd ↓	Improvement ↑ (Nd, %)	Time (s) ↓	Improvement ↑ (Time, %)
Default (Mfs2)	6634 (0.0)	NA	66.17 (5.25)	NA	215708.00 (0.0)	NA	201.43 (12.17)	NA
2PruneX (COG, 0.3)	9496.66 (16.99)	**0.43**	184.02 (9.03)	**21.55**	6538.33 (8.99)	**1.44**	52.69 (0.26)	**20.37**
2PruneX (COG, 0.4)	9452.66 (7.40)	**0.89**	292.90 (14.26)	-24.86	6516.66 (10.84)	**1.77**	85.80 (3.84)	-29.67

Table 30. The results show that the ratio of the total model time to the runtime of the Mfs2 heuristic (i.e., Percent) is very low, with an average of 2.66%.

Circuit	Nodes	Graph construction time (s)	Model inference time (s)	Total model time (s)	Runtime of heuristic (s)	Percent (%)
Log2	32060	0.48 (0.01)	2.01 (0.05)	2.49	156.19 (1.66)	1.59
Hyp	214335	2.56 (0.02)	6.87 (0.09)	9.43	274.13 (9.90)	3.44
Vga lcd	124050	1.52 (0.04)	5.02 (0.33)	6.54	189.10 (3.37)	3.46
ci6	788288	10.87 (0.01)	25.54 (0.26)	36.41	1436.53 (123.54)	2.53
ci4	37051	0.57 (0.10)	2.18 (0.22)	2.75	234.58 (45.54)	1.17
ci3	24778	0.41 (0.004)	2.10 (0.07)	2.51	66.17 (5.25)	3.79

Table 31. A detailed description of circuits from the EPFL benchmark. Nodes denotes the sizes of circuits, and Lev denotes the depths of circuits.

Circuit	PI	PO	Latch	Nodes	Edge	Cube	Lev
Adder	256	129	0	1020	2040	1020	255
Barrel shifter	135	128	0	3336	6672	3336	12
Divisor	128	128	0	57247	114494	57247	4372
Hypotenuse	256	128	0	214335	428670	214335	24801
Log2	32	32	0	32060	64120	323060	444
Max	512	130	0	2865	5730	2865	287
Multiplier	128	128	0	27062	54124	27062	274
Sin	24	25	0	5416	10832	5416	225
Square-root	128	64	0	24618	49236	24618	5058
Square	64	128	0	18486	36969	18485	250
Round-robin ariter	256	129	0	11839	23678	11839	87
Alu control unit	7	26	0	175	348	174	10
Coding-cavlc	10	11	0	693	1386	693	16
Decoder	8	256	0	304	608	304	3
i2c controller	147	142	0	1357	2698	1356	20
Int to float converter	11	7	0	260	520	260	16
Memory controller	1204	1230	0	47110	93945	47109	114
Priority encoder	128	8	0	978	1956	978	250
Lookahead XY router	60	30	0	284	514	257	54
Voter	1001	1	0	13758	27516	13758	70

Table 32. A detailed description of circuits from the IWLS benchmark. Nodes denotes the sizes of circuits, and Lev denotes the depths of circuits.

Circuit	PI	PO	latch	nodes	edge	cube	lev
aes_core	259	129	530	20797	40645	24444	28
des_area	240	64	128	5005	9882	5889	35
des_perf	234	64	8808	98463	180542	108666	28
ethernet	98	115	10544	46804	113378	72850	37
i2c	19	14	128	1147	2299	1375	15
mem_ctrl	115	152	1083	11508	26436	14603	31
pci_bridge32	162	207	3359	16897	34607	23130	29
pci_conf_cyc_addr_dec	32	32	0	109	212	128	6
pci_spoci_ctrl	25	13	60	1271	2637	1557	19
sasc	16	12	117	552	1148	766	10
simple_spi	16	12	132	823	1694	1089	14
spi	47	45	229	3230	6904	4054	32
steppermotordrive	4	4	25	228	397	253	11
systemcaes	260	129	670	7961	18236	11648	44
systemcdes	132	65	190	3324	6304	3791	33
tv80	14	32	359	7166	16280	9352	50
usb_funct	128	121	1746	12871	27102	16378	25
usb_phy	15	18	98	559	1001	638	12
vga_lcd	89	109	17079	124050	242332	146201	25
wb_conmax	1130	1416	770	29036	77185	39619	26
wb_dma	217	215	263	3495	7052	4496	26

Table 33. A detailed description of two very large-scale circuits from the EPFL benchmark. Nodes denotes the sizes of circuits, and Lev denotes the depths of circuits.

Circuit	PI	PO	Latch	Nodes	Lev
twenty	137	60	0	20732893	162
sixteen	117	50	0	16216836	140

Table 34. A statsical description of 27 industrial circuits (21 training circuits and 6 testing circuits) from Huawei HiSilicon. Nodes denotes the sizes of circuits, and Lev denotes the depths of circuits.

Traning Circuits	PI	PO	Latch	Nodes	Lev
mean	8410.5	5978.682	0	104229.4	55.95455
max	59974	29721	0	788288	104
min	41	107	0	2775	18
Testing Circuits	**PI**	**PO**	**latch**	**nodes**	**lev**
mean	18540.67	18015	0	356111.2	103.3333
max	42257	33849	0	655243	185
min	523	483	0	24778	40

Table 35. We provide our manually designed node features.

heuristic	Node features		
Mfs2	Functionality information	Structure information	Other features
Input: 6-LUTs	Truth table of the node the node is 6-input LUT (64-dimensional vector)	-	Fanin number Fanout number Level LevelR Node ID
Resub	Functionality information	Structure information	Other features
Input: AIG	inv0 inv1	Child1 features (6-dimensional vector) Child2 features (6-dimensional vector)	Fanout number Level Total Level Node ID Leaves number Cuts number