ArtNet: Hierarchical Clustering-Based Artificial Netlist Generator for ML and DTCO Applications

Andrew B. Kahng, *Fellow, IEEE*, Seokhyeong Kang, *Senior Member, IEEE*,
Seonghyeon Park, *Student Member, IEEE*, and Dooseok Yoon, *Student Member, IEEE*

***Abstract*—In advanced nodes, optimization of power, performance and area (PPA) has become highly complex and challenging. Machine learning (ML) and design-technology co-optimization (DTCO) provide promising mitigations, but face limitations due to a lack of diverse training data as well as long design flow turnaround times (TAT). We propose *ArtNet*, a novel artificial netlist generator designed to tackle these issues. Unlike previous methods, ArtNet replicates key topological characteristics, enhancing ML model generalization and supporting broader design space exploration for DTCO. By producing realistic artificial datasets that more closely match given target parameters, ArtNet enables more efficient PPA optimization and exploration of flows and design enablements. In the context of CNN-based DRV prediction, ArtNet's data augmentation improves F1 score by 0.16 compared to using only the original (real) dataset. In the DTCO context, ArtNet-generated *mini-brains* achieve a PPA match up to 97.94%, demonstrating close alignment with design metrics of targeted full-scale block designs.**

I. Introduction

As modern designs increase in complexity and scale, improvement of power, performance, and area (PPA) has become more challenging. Place-and-route (P&R) tools rely heavily on heuristics, but struggle with problem scale and the need to balance turnaround time (TAT) against quality of results (QoR). Machine learning (ML) offers the promise of TAT reduction through prediction and optimization of design processes to avoid iterative design loops [26]. However, data requirements of ML are difficult to satisfy, and obtaining high-quality, sharable design datasets remains a key challenge. Restrictions on sharing of proprietary designs and EDA tool outputs hinder creation of comprehensive datasets, limiting the effectiveness of ML models and underlying research efforts.

At the same time, the slowdown of Moore's Law has made design-technology co-optimization (DTCO) essential to PPA improvement in advanced nodes [4] [5]. However, co-exploration of the broad solution space for design and technology is gated by large tool and flow TAT on real designs.

In this work, we address the above challenges with ***ArtNet***, an artificial netlist generator that supplements limited real-world datasets to enhance the performance of ML models. By producing novel *mini-brains*, ArtNet also accelerates solution space exploration for design methodologies and DTCO. Ultimately, these benefits enable more effective PPA optimization in complex, large-scale designs.

A. Artificial Data: What and Why

Artificial data is increasingly attractive for its ability to mitigate high cost, limited availability, and security risks of real-world data [17]. Generated artificial data helps ML models and optimization heuristics comprehend rare corner cases that are otherwise difficult to capture with real-world data [21]. Diversity from artificial datasets can reduce data bias and enhance ML model robustness to input variations, alongside privacy-preserving benefits [15].

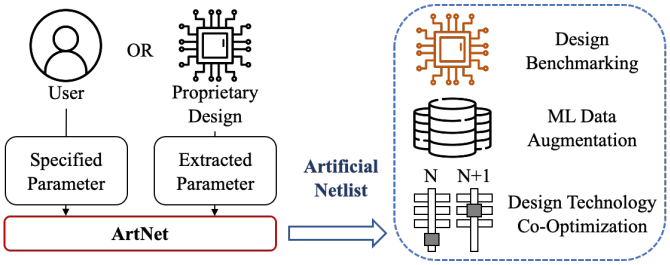


Fig. 1: Use cases of artificial netlist generation. ArtNet enables netlist generation from user-specified input parameters, and from parameters of a given target design. The output netlist can be used for design benchmarking, ML data augmentation, DTCO and other applications.

In VLSI physical design, a range of previous works have demonstrated the usefulness of artificial designs. In ML contexts, artificial netlists have been used to augment training datasets, ultimately improving model performance. For example, [26] [20] apply ML-based techniques for early-stage prediction and generation of routing blockages to minimize design rule violations (DRVs). Artificial designs can also help reduce biases and prevent faulty decisions during DTCO exploration: [18] uses a generic mesh-like netlist topology as the basis of routability assessment, while [4] analogously employs a knight's tour-based topology; [5] leverages artificial netlists to account for metrics such as routed wirelength, timing, and power. Fig. 1 depicts use cases for artificial netlist generation, spanning from types of inputs to diverse applications.

B. Related Work

Previous artificial netlist generators have sought to generate realistic netlists as benchmarks for evaluation of VLSI physical design algorithms. They focus on replication of classical netlist attributes such as Rent's rule, net degree distribution, and logic depth. **RMC** [7] uses a top-down partitioning strategy, but produces overly connected designs that mismatch realistic net degree distributions and Rent's rule parameter values. **GNL** [29] improves on this with bottom-up clustering that merges gate clusters while maintaining Rent's-rule scaling; however, its lack of logic depth control leads to excessive wirelengths and path delays. An enhanced version [31] builds on GNL by adding features for delay control and prevention of combinational loops, but its insertion of extra flip-flops causes inaccurate power estimates and large wirelengths. **Circ/Gen** [13] [22] proposes a hybrid approach that combines real circuit analysis with artificial netlist generation. However, its dependence on availability of input netlists and its excessive runtime reduce usability. Last, **ANG** [21] avoids dependence

on real circuits through parameter-based netlist generation, making it suitable for ML data augmentation. However, ANG offers limited design coverage as it cannot handle macros and suffers from high runtime complexity. This impacts usability in modern contexts where designs can be macro-heavy and have huge scale. Additionally, ANG's probabilistic graph generation can result in repetitive netlist structures.

Recently, ML-based DAG generation methods have been proposed [33] [1] [24]. For example, **LayerDAG** [24] proposes a layerwise autoregressive diffusion model that captures both directional and logical dependencies in DAGs, enabling the conditional generation of synthetic task graphs for hardware benchmarking. By conditioning on hardware metrics (e.g., runtime, resource usage), LayerDAG generates task graphs that can be used to train surrogate models for performance prediction without requiring actual hardware execution. However, since LayerDAG can only generate relatively small graphs (up to 400 nodes), it has limitations in modeling modern VLSI gate-level netlist hypergraphs that have thousands to millions of nodes. **LLM4Netlist** [32] proposes a step-wise generation framework that translates natural language descriptions into Berkeley logic interchange format (BLIF) netlists using a fine-tuned LLM. While it achieves state-of-the-art functional correctness on benchmark datasets, it still faces challenges in scaling to highly complex or large-scale circuits, and the inference time grows significantly with circuit size due to iterative decoding. Furthermore, since detailed functional descriptions are required for training, large-scale data generation remains difficult. **HYGENE** [8] enables the generation of hyperedges using a hypergraph generation method. However, it also has limitations in generating large graphs with multi-million nodes. Overall, ML-based methods tend to rely heavily on reference circuits and incur high training costs, making them resource-intensive; moreover, they have super-linear time complexity with the number of nodes [9].

Our proposed **ArtNet** overcomes the above challenges through highly efficient generation of artificial netlists that accurately match user-defined topological parameters and logical hierarchies. As detailed in Section III, ArtNet leverages Rent's rule to achieve realistic interconnect complexity, and considers sequential ratios and logic depth to produce netlists with realistic timing properties. Table I compares previous works and ArtNet.

TABLE I: Comparison of previous works with ArtNet.

Capability	**ANG** [21]	**Circ/Gen** [22]	**GNL** [29]	**ArtNet**
Runtime Efficiency	-	- -	+	++
Heterogeneous Modules	X	X	✓	✓
Interconnect Complexity	△	△	✓	✓
Combinational Loop Prevention	X	✓	✓	✓
Sequential Cell Ratio Matching	△	✓	X	✓
Logic Depth Matching	△	✓	✓	✓
Unconnected Cell Prevention	X	X	✓	✓
Dangling Net Prevention	X	X	✓	✓
#PI/PO Matching	✓	✓	X	✓
Macro Cell Insertion	X	X	✓	✓
Tech Mapping	✓	X	✓	✓
Real Design-based	✓	✓	△	✓
User-Specified Input-based	✓	X	X	✓

* △ denotes poor matching quality or incomplete flow support. '+' indicates fast runtime, and '-' indicates slow runtime. '✓' and 'X' respectively indicate presence and absence of a capability or attribute.

C. Our Contributions

Following are the key contributions (capabilities) of ArtNet.

- **Heterogeneity.** ArtNet can follow given hierarchical structure and submodule information, achieving netlist heterogeneity by creating clusters with different submodule characteristics and merging them into higher-level clusters. Balancing of terminals across cluster groups is used to control local interconnect complexity.
- **Interconnect complexity.** Given a set of input clusters, ArtNet generates interconnections to meet target input parameters such as Rent's exponent. ArtNet also ensures matching of specified numbers of primary inputs (PIs) and primary outputs (POs), and prevents dangling nets and floating pins.
- **Timing paths.** ArtNet performs net generation and timing path construction concurrently. For specified minimum and maximum logic depths, ArtNet seeks to keep all paths within the designated logic depth range while meeting other target attributes, such as the sequential ratio. In case of a conflict between the logic depth range and the sequential ratio, ArtNet gives higher priority to the logic depth range.
- **Experimental validations in ML context.** We validate ArtNet in an ML context (CNN-based DRV prediction), where ArtNet's augmentation of the original (real) dataset improves F1 score by 0.16.
- **Experimental validations in DTCO context.** In a DTCO context, *mini-brains*, i.e., ArtNet-generated netlists scaled to 10% of a full-scale design's size, achieve a strong PPA match (MAPE with respect to power per unit area and effective clock period (= target clock period minus worst endpoint slack), down to 2.06%) relative to full-scale designs.

II. Preliminaries

In this section, we begin by introducing netlist parameters, key notations, and ArtNet input methods.

A. Netlist Parameter Configuration

We now establish definitions and notation used in the paper. A *netlist* comprises *instances*, *nets*, *PIs* and *POs*. These elements define the (hyper)graph topology of the netlist, including overall connectivity and structure. Parameters that affect circuit size include number of instances (N_{inst}), number of macros (N_{macro}), and numbers of primary inputs (N_{PI}) and primary outputs (N_{PO}). Additionally, characteristics such as *sequential ratio* and *logic depth* determine the timing behavior of the circuit. We list these characteristics in Table II.

TABLE II: Description of netlist parameters.

Parameter	**Description**
N_{inst}	#instances
N_{PI}, N_{PO}	#primary input and #primary output pins, respectively
N_{macro}	#macros
R_{ratio}	Ratio of *Region 1* [23] max block size to circuit size
p	Rent's exponent (fitted using RentCon)
T_{avg}	Average #pins per the instances
S_{ratio}	Ratio of #sequential instances to the N_{inst}
D_{max}	Maximum depth of any timing path
D_{min}	Minimum depth of any timing path
MD_{max}	Maximum depth of macro timing path
MD_{min}	Minimum depth of macro timing path

*Standard deviations of p and T_{avg} are used as hyperparameters.

Interconnect Complexity. We parameterize interconnect complexity based on Rent's rule [23], the well-known empirical

power-law relationship between T, the total number of PIs and POs in a logic block, and B, the number of gates within that block. This relationship is $T = kB^p$, where k and p are Rent's constant and Rent's exponent, respectively. The Rent's exponent p ranges between 0 and 1; higher values indicate higher interconnect complexity. In ArtNet, we parameterize interconnect complexity of the circuit using Rent's exponent (p), and average number of terminals per instance (T_{avg}).[1]

Timing Characteristics. A netlist contains timing paths, i.e., purely combinational paths between sequentially-adjacent launch and capture flip-flops. We use logic depth as a proxy for signal propagation delay, and use minimum (D_{min}) and maximum (D_{max}) path depths as parameters to capture timing behavior. To manage timing constraints in detail, we also define minimum (MD_{min}) and maximum (MD_{max}) path depths for paths whose endpoints are macros. Another important parameter is S_{ratio}, the fraction of instances that are sequential cells. S_{ratio} impacts not only the average depth of timing paths, but pin density and clock tree wirelength as well [21].

B. Notations

Table III summarizes additional notation used in the paper.

TABLE III: Notation and description.

Notation	Description
X_{in}	Vector of input parameters $\{N_{inst}, N_{PI}, N_{PO}, ...\}$
$SpecFile$	Input file to ArtNet, generated from X_{in} and $.lef$
$clust$	Instances and submodules directly below the top module in the logical hierarchy
Q_{size}	A tree-based min-priority queue sorted by cluster size with randomized (reproducible for given seed) tie-breaks
Q_{seq}	A (FIFO) queue consisting of flip-flops
τ	Combining constraints for hierarchical clustering
I	Input terminals of cluster
O	Output terminals of cluster
$I(A)$	List of input nets of cluster A
$O(A)$	List of output nets of cluster A
$\mathcal{I}_{net}(A)$	Nets connecting to the inside of the cluster A
$\mathcal{E}_{net}(A)$	Nets connecting to the outside of the cluster A
σ_T	Standard deviation of input parameter T_{avg}
σ_p	Standard deviation of input parameter p

C. Artificial Netlist Generation Methods

ArtNet enables netlist generation from (1) user-specified parameters, and (2) from parameters of a given target design (Fig. 1).

User-specified parameters-based. ArtNet can generate netlists based on user-specified structural parameters, without requiring a reference design. In contrast to prior approaches [29] [22] which rely on fixed templates or seed circuits, ArtNet allows direct specification of key structural attributes. This enables generation of netlists that reflect the intended characteristics of a given target domain. Given a set of parameters, ArtNet constructs a corresponding *SpecFile*, which guides the netlist generation process (Sections IV-A, IV-B and IV-D).

Target design-based. ArtNet also supports generation of artificial netlists from real designs by extracting structural and timing parameters directly from the netlist. This allows the generated netlist to reflect the key characteristics of the original design. As a result, unsharable industrial designs can be abstracted and shared with the academic community, enabling design benchmarking and EDA tool evaluation. When a real netlist is provided, we extract the parameters listed in Table II using OpenDB (N_{inst}, N_{PI}, N_{PO}, N_{macro}, T_{avg}, S_{ratio}), OpenSTA (D_{min}, D_{max}), and RentCon (p, R_{ratio}). These parameters are used to generate a *SpecFile*, which is then used by ArtNet to produce an artificial netlist that statistically resembles the original design (Sections IV-C and IV-E). To make this functionality accessible to a wide range of users, this *SpecFile* generation flow is implemented in the OpenROAD project [36] as the `write_artnet_spec` command in the OpenROAD application, and a more detailed *SpecFile* description is provided in [41].

III. Our Approach

ArtNet performs hierarchical clustering-based netlist generation as shown in Fig. 2. Our approach consists of three main steps: (1) hierarchical clustering, (2) net generation, and (3) PI/PO matching. The input to this process includes the *SpecFile*, generated from a vector of input parameters, and the .lef file corresponding to the target technology and design enablement. The .lef file is used to generate the cell list in *SpecFile*. The *SpecFile* also includes logical hierarchy information. To reflect the logical hierarchy, prevent combinational loops, and adhere to Rent's exponent during netlist generation, we employ a bottom-up clustering approach as in GNL [31]. Our approach is detailed in Algorithm 1.

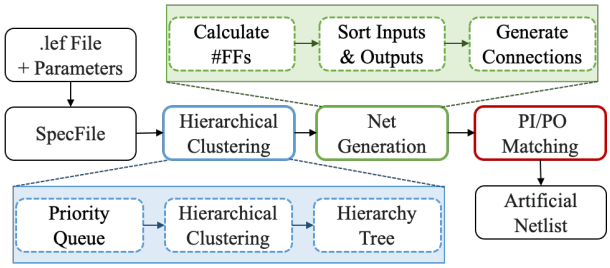


Fig. 2: Overall ArtNet framework.

Lines 1-6: If the *SpecFile* contains logical hierarchy information, we extract the hierarchy and input parameters ($X_{in}^{sub_i}$) for each submodule. Next, we store the input parameter details for the top module (X_{in}^{top}). We use OpenDB from the OpenROAD project [36] to parse *.lef* and map instances to their corresponding masters.

Lines 7-15: From the top module's perspective, we create two queues: (1) a min-priority queue (Q_{size}), which contains each combinational gate, macro, and submodule as a cluster, and (2) a sequential queue (Q_{seq}), which contains all flip-flops (FFs)[2]. In Q_{size}, the priority of each cluster is determined by the number of instances it contains, in ascending order, with clusters of the same size arranged in a random (but, reproducible according to initial seed) order. We define the combining constraints (τ) based on X_{in}^{top} and execute hierarchical clustering using Algorithm 2. Once clustering is

[1]To evaluate the Rent's exponent of a circuit, we use the RentCon methodology and code from [34].

[2]Macro instances are treated as high-pin-count instances and, despite having a clock pin, are included in Q_{size}. However, they are regarded as sequential cells during timing path construction.

Algorithm 1: Overall flow.

inputs: *SpecFile*, *.lef*
output: *Netlist* (.v)
1 /* Read input parameters */
2 **if** Logical hierarchy is present **then**
3 **for** $i \leftarrow 0; i < \#submodules_{top}; i \leftarrow i+1$ **do**
4 $X_{in}^{sub_i} \leftarrow$ Read parameters of submodules from *SpecFile*
5 $X_{in}^{top} \leftarrow$ Read parameters of top module from *SpecFile*
6 Read *.lef* using OpenDB and map instances to masters
7 **for** *each instance or submodule in SpecFile* **do**
8 **if** *item is combinational, macro or a submodule* **then**
9 $Q_{size} \leftarrow$ Enqueue item
10 **else**
11 $Q_{seq} \leftarrow$ Enqueue item (used in Algorithm 4)
12 /* Hierarchical clustering */
13 $\tau \leftarrow$ Generate combining constraints from X_{in}^{top}, as defined in Algorithm 2
14 $clust_{root} \leftarrow$ Get root cluster from clustering using Algorithm 2
15 $H(V, E) \leftarrow$ Generate hierarchy tree from $clust_{root}$
16 /* Net generation */
17 $level_{\max} \leftarrow H(V, E)$
18 **for** $i \leftarrow level_{max} - 1; i > level_0; i \leftarrow i-1$ **do**
19 $clust_i \leftarrow$ cluster corresponding to level i
20 **if** $clust_i$ == *submodule* **then**
21 Hierarchical clustering using Algorithm 2
22 Generate nets between $clust_i.leftChild$ and $clust_i.rightChild$ using Algorithm 3
23 /* PI/PO matching */
24 **if** $I_{clust_{root}} \neq N_{PI} || O_{clust_{root}} \neq N_{PO}$ **then**
25 Match $I_{clust_{root}}$ and $O_{clust_{root}}$ to N_{PI} and N_{PO}
26 *Netlist* $\leftarrow$ Write netlist
27 **return** *Netlist*

complete, leaving only one element in the queue, we construct a hierarchy tree H with the final cluster $clust_{root}$ as its root. ***Lines 16-22:*** We levelize H so that all leaf nodes in H are at the same level. Each leaf node represents either a submodule or a single instance. Moving from the leaf nodes up to the root node, we create net connections between clusters. If a cluster corresponds to a submodule, we recursively perform hierarchical clustering and net generation within that submodule. ***Lines 23-26:*** If #PIs and #POs of the root cluster $clust_{root}$ do not match the input parameters N_{PI} and N_{PO}, we perform *PI/PO matching* to align them by adding or deleting PIs and POs, as described in Fig. 5. Adding PIs stops if every FF has a single fanout, and deleting POs stops if every PO driver has a single fanout.

A. Hierarchical Clustering

Hierarchical clustering in ArtNet considers the design's complexity by applying constraints based on the average number of pins per module, the Rent's exponent, and its standard deviation. This ensures that the resulting hierarchy remains structurally balanced and design-aware. Fig. 3 illustrates the construction process of the hierarchy tree (Algorithm 2). In the initial priority queue, A, B, C and D denote individual instances, while EFG represents a predefined submodule of the top module. At each step, the clustering algorithm selects and merges groups of instances or clusters according to the priority queue, progressively building up higher-level modules. This process continues iteratively until only a single cluster remains, which becomes the top of the hierarchy. The algorithm continues clustering until only a single cluster remains in the queue. By combining predefined submodules with dynamically clustered instances, ArtNet can construct a hierarchical structure that closely follows the submodule organization of real designs; this information from the real design is written in the *SpecFile* and applied during submodule generation as described in Algorithm 1, Line 4.

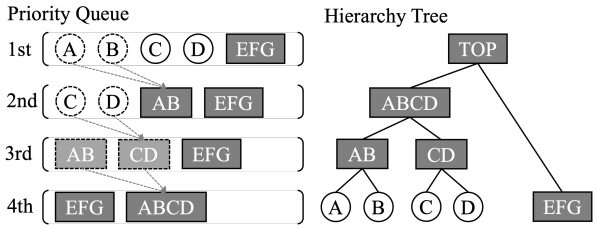


Fig. 3: Hierarchical clustering from a given priority queue.

Algorithm 2: Hierarchical clustering.

inputs: Priority Queue Q_{size}
output: Root Cluster $clust_{root}$
1 $fail\_count \leftarrow 0$
2 $fail\_limit \leftarrow |Q_{size}|$
3 **while** $|Q_{size}| \geq 1$ **do**
4 $clust_A, clust_B \leftarrow$ Dequeue from Q_{size}
5 $\tau \leftarrow (T(clust_A) > MaxT(|clust_B|)) \vee (T(clust_B) > MaxT(|clust_A|))$
6 **if** τ **then**
7 $Q_{size} \leftarrow$ Enqueue $clust_A$ and $clust_B$
8 $fail\_count \leftarrow fail\_count + 1$
9 **if** $fail\_count \geq fail\_limit$ **then**
10 $\sigma_T \leftarrow \sigma_T * 1.2$
11 $fail\_count \leftarrow 0$
12 $fail\_limit \leftarrow |Q_{size}|$
13 **else**
14 $clust_{AB} \leftarrow$ Merge $clust_A$ and $clust_B$
15 $clust_{AB}.rightChild \leftarrow clust_A$
16 $clust_{AB}.leftChild \leftarrow clust_B$
17 $Q_{size} \leftarrow$ Enqueue $clust_{AB}$
18 $fail\_count \leftarrow 0$
19 $fail\_limit \leftarrow |Q_{size}|$
20 $clust_{root} \leftarrow$ Dequeue from Q_{size}
21 **return** $clust_{root}$

Lines 1-12: We start with Q_{size} that contains initial clusters, and initialize a failure counter along with a threshold based on the queue size. We dequeue two clusters $clust_A$ and $clust_B$. The combining condition (τ) determines whether it is feasible to merge the two clusters, with $T(clust_A)$ and $T(clust_B)$ denoting the number of terminals associated with clusters $clust_A$ and $clust_B$, respectively. If τ holds, we re-enqueue $clust_A$ and $clust_B$ and increment $fail\_count$; when it reaches $fail\_limit$, we relax the merge criterion by updating σ_T. The thresholds $MaxT(|clust_B|)$ and $MaxT(|clust_A|)$ indicate the maximum allowable number of terminals for merging the clusters, and are determined from the Rent's rule expression:

$$MaxT(|clust|) = T_{avg}|clust|^p + \alpha\sigma_T|clust|^{\sigma_p} \quad (1)$$

The second term on the right-hand side is included to ensure diversity in the terminal count distribution of the cluster. ***Lines 13-21:*** If the combining condition is satisfied, $clust_A$ and $clust_B$ are merged into a new cluster, $clust_{AB}$. We designate $clust_A$ as the right child and $clust_B$ as the left child of $clust_{AB}$, capturing the hierarchical relationship between the clusters. Finally, the merged cluster $clust_{AB}$ is enqueued back

into Q_{size} for the next iteration. After the clustering is done, the root cluster $clust_{root}$ is dequeued from Q_{size} and returned.

B. Net Generation

Fig. 4 illustrates the net generation process. We recall the example from hierarchical clustering, where cluster AB comprises child clusters A and B. Net generation creates nets between A and B to connect them within cluster AB. There are three main steps: (1) calculate the number of input/output terminals and the number of nets to be created; (2) determine the required number of additional FFs from Q_{seq} to maintain the S_{ratio} of the cluster; and (3) connect the nets in a manner that ensures the logic depth constraints are satisfied.

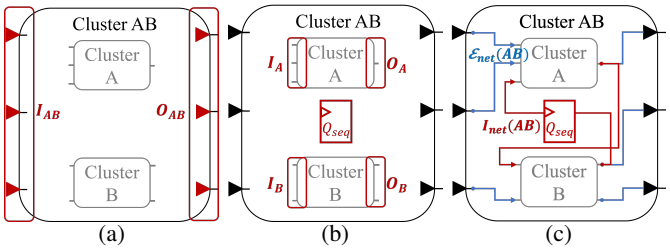


Fig. 4: Net generation flow: (a) calculate In/Out terminals to be created; (b) determine additional FFs from Q_{seq}; and (c) make connections. (Refer to Table III for notation.)

Interconnect complexity. Algorithm 3 describes our net generation approach that addresses both interconnect complexity and logic depth constraints.

Lines 1-2: We define the mean number of terminals (μ_T) and its standard deviation (σ_T) based on the given input parameters T_{avg}, p and the cluster size $|clust|$ as follows:

$$\mu_T = T_{avg}|clust|^p, \quad \sigma_T = |clust|^{\sigma_p}. \tag{2}$$

Similarly, using the hyperparameter G_{avg}[3] and its standard deviation σ_G, the total number of terminals T, and the ratio of the number of outputs to the number of terminals G, are defined by the following distributions:

$$T \sim \mathcal{N}(\mu_T, \sigma_T), \quad G \sim \mathcal{N}(G_{avg}, \sigma_G) \tag{3}$$

Then, we get the number of input terminals I_{AB} by $T*(1-G)$ and the number of output terminals O_{AB} by $T*G$.

Lines 3-9: Let clusters A and B have sets of input/output nets I_A/O_A and I_B/O_B, respectively. If the number of output nets exceeds the number of input nets, then the internal net count ($\mathcal{I}_{net}$) is set to the average of these values, balancing connections between inputs and outputs. Otherwise, it is set to the output net count. The external net count is then calculated by subtracting the input net count from the total number of input nets, ensuring that the sum of internal and external net counts matches the module's connection capacity. This approach maintains a balanced connection distribution, and multi-sink nets naturally arise when terminal counts are matched, since a parent-level cluster's terminal aggregates multiple underlying sinks.

Lines 10-32: We sort output nets by logic depth, the logic cell hop count to the farthest adjacent sequential cell, in descending order, and input nets by logic depth in ascending order. $I(A)$ and $O(A)$ denote the input/output terminal lists of cluster A. $I(A)_i$ denotes the i^{th} element of $I(A)$ and $I(A)_0$ refers to the first element of $I(A)$. We maintain indices i and j as pointers into $I(B)$ and $I(A)$, respectively, and use k as a counter for the number of inter-cluster connections. We initialize i, j, k to 0. When connecting these nets, we ensure that no resulting path depth exceeds D_{max} (if a combined path length exceeds D_{max}, the connection is skipped). We check whether connecting $O(A)_0$ to $I(B)_i$ or $O(B)_0$ to $I(A)_j$ satisfies the depth constraint. When it is satisfied, the corresponding input net is removed and the resulting output net is appended to the external net set $\mathcal{E}_{net}$. When the constraint is not satisfied, the corresponding input net is connected to the resulting output net using Algorithm 4. In the final clustering, all unresolved connections are completed using Algorithm 4. This structurally guarantees that the maximum depth constraint is satisfied and that no dangling nets remain. If neither of these connections is possible, we exit the loop (Line 32). This approach prioritizes the longest paths first, addressing the most critical paths early in the process.

[3] G_{avg} means the average ratio of the number of outputs to the number of terminals of a cluster.

Algorithm 3: Net generation.

inputs : Cluster $clust_{AB}$, Input parameters X_{in}^{top}
output: Net connections in Cluster $clust_{AB}$

1 /* Calculate input and output counts */
2 $I_{AB}, O_{AB} \leftarrow getIO(|clust_{AB}|)$
3 /* Calculate number of nets */
4 $clust_A \leftarrow clust_{AB}.leftChild$ and $clust_B \leftarrow clust_{AB}.rightChild$
5 **if** $(O_A + O_B - O_{AB}) > (I_A + I_B - I_{AB})$ **then**
6 $\quad |\mathcal{I}_{net}(AB)| \leftarrow (O_A + O_B - O_{AB} + I_A + I_B - I_{AB})/2$
7 **else**
8 $\quad |\mathcal{I}_{net}(AB)| \leftarrow (O_A + O_B - O_{AB})$
9 $|\mathcal{E}_{net}(AB)| \leftarrow I_A + I_B - I_{AB} - |\mathcal{I}_{net}(AB)|$
10 Sort($I(A), I(B)$) and Sort($O(A), O(B)$)
11 Initialize indices $i \leftarrow 0$, $j \leftarrow 0$, and counter $k \leftarrow 0$
12 /* Make external (inter-cluster) nets */
13 **while** $k < |\mathcal{I}_{net}(AB)| + |\mathcal{E}_{net}(AB)|$ **do**
14 $\quad Cond_{A\to B} \leftarrow (Depth(O(A)_0) + Depth(I(B)_i) \le D_{max})$
15 $\quad Cond_{B\to A} \leftarrow (Depth(O(B)_0) + Depth(I(A)_j) \le D_{max})$
16 $\quad$ **if** $Cond_{A\to B}$ **then**
17 $\quad\quad O(A)_0 \leftarrow Connect(O(A)_0, I(B)_i)$
18 $\quad\quad$ Delete $I(B)_i$, Append $O(A)_0$ to $\mathcal{E}_{net}(AB)$, and k++
19 $\quad$ **else if** $FF_{new} > 0$ *or Final clustering* **then**
20 $\quad\quad O(A)_0 \leftarrow Connect(O(A)_0, I(B)_i)$ using Algorithm 4
21 $\quad$ **else**
22 $\quad\quad i \leftarrow i + 1$
23 $\quad$ **if** $Cond_{B\to A}$ **then**
24 $\quad\quad O(B)_0 \leftarrow Connect(O(B)_0, I(A)_j)$
25 $\quad\quad$ Delete $I(A)_j$, Append $O(B)_0$ to $\mathcal{E}_{net}(AB)$, and k++
26 $\quad$ **else if** $FF_{new} > 0$ *or Final clustering* **then**
27 $\quad\quad O(B)_0 \leftarrow Connect(O(B)_0, I(A)_j)$ using Algorithm 4
28 $\quad$ **else**
29 $\quad\quad j \leftarrow j + 1$
30 $\quad$ **if** *no connection available* **then**
31 $\quad\quad$ /* Promote unresolved nets to $clust_{AB}$ */
32 $\quad\quad$ break
33 /* Convert external nets to internal nets */
34 Sort($\mathcal{E}_{net}(AB)$)
35 **while** $|\mathcal{I}_{net}(AB)| > 0$ **do**
36 $\quad$ Append $\mathcal{E}_{net}(AB)_0$ to $\mathcal{I}_{net}(AB)$
37 $\quad$ Remove $\mathcal{E}_{net}(AB)_0$ from $\mathcal{E}_{net}(AB)$ and $|\mathcal{I}_{net}(AB)|$--
38 Delete $clust_A$ and $clust_B$
39 **return** $clust_{AB}$

Lines 33-39: External nets are sorted in descending order of their corresponding input nets' depth to prevent large-depth paths from persisting through subsequent clustering stages. After sorting, they are converted to internal nets until the required number of internal nets is reached. Last, we remove $clust_A$ and $clust_B$ and return the merged cluster $clust_{AB}$.

Timing path constraints. To handle cases where the path length constraints are not satisfied (Lines 19-20 and 26-27 in Algorithm 3), we introduce flip-flops (FF_{new}) as needed. We first determine the number of additional flip-flops required during net generation:

$$FF_{new} = \lfloor S_{ratio} \times |clust_{AB}| - (FF_{clust_A} + FF_{clust_B}) \rfloor. \quad (4)$$

We then insert FFs according to the following two cases:

Case 1. If the output net's path length exceeds the input net's maximum allowable length, a FF is inserted from the sequential queue (Q_{seq}) to adjust the path length and satisfy the constraints, using the allocated FF_{new}. This aims to meet the S_{ratio} initially.

Case 2. During the final clustering stage, if additional FFs are required to meet logic depth requirements, they are inserted even if this exceeds the prescribed sequential ratio (S_{ratio}). This ensures that required logic depth constraints are met across the entire design.

Algorithm 4 describes our FF insertion approach. As noted above, FF insertion is performed during the process of connecting nets between two clusters. The algorithm specifically describes the insertion of a FF into the output net $O(A)$ of $clust_A$ during the net generation between $clust_A$ and $clust_B$.

Algorithm 4: Insert a Flip-Flop.

inputs : Cluster $clust_{AB}$
Sequential queue Q_{seq}
An output net $O(A)$
output: The net updated with a flip-flop between its source and sinks

1 /* Prepare a flip-flop instance */
2 **if** Q_{seq} *is not empty* **then**
3 $FF \leftarrow$ Dequeue from Q_{seq}
4 **else**
5 $FF \leftarrow$ default flip-flop FF
6 Update S_{ratio}, N_{inst} of $clust_{AB}$
7 /* Insert FF between the net's source and sinks */
8 Create a net Net_D to drive the data input D of FF
9 Move the source of $O(A)$ to Net_D
10 Connect the data output Q of FF to the source of $O(A)$
11 Add Net_D to $\mathcal{I}_{net}(A)$
12 **return** *Updated output net* $O(A)$

Lines 1-5: We prepare a flip-flop instance FF by either dequeuing from the sequential queue Q_{seq} or instantiating a default flip-flop when the queue is empty. The default flip-flop is determined either by a user-specified flip-flop or by selecting the most frequently used flip-flop in the design.

Line 6: We then update the S_{ratio} and N_{inst} of the merged cluster $clust_{AB}$. This update contributes to maintaining an appropriate S_{ratio} and target interconnect complexity during remaining net generation.

Lines 7-12: We insert the FF between the source and sinks of the original output net $O(A)$. A new net Net_D is created to connect the original source to the data input pin (D) of the flip-flop, while output net $O(A)$ connects the data output pin (Q) of the flip-flop to the sink. This insertion restructures the netlist to control the timing path depths while preserving the original connectivity.

C. PI/PO Matching

PI/PO matching adjusts PIs and POs so that the generated netlist better reflects specified attributes even if this affects the Ratio of *Region 1* (R_{ratio}). Fig. 5 illustrates four main operations for PI and PO manipulation.

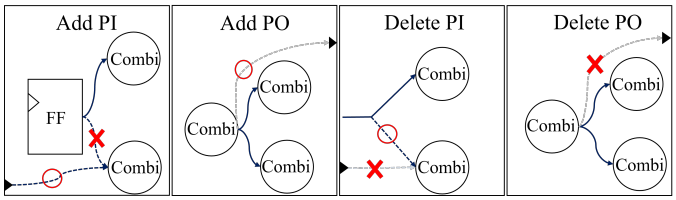


Fig. 5: PI/PO matching methods.

Add PI. When adding a PI by converting the input of a combinational gate that is a fanout of a FF, it is essential that the FF has additional sinks other than this specific combinational gate. This ensures that disconnecting the FF from this gate does not leave it without connections, preserving overall connectivity and helping to maintain realistic logic depths.

Delete PI. Removing a PI can lead to floating inputs on the connected logic, and reconnecting these inputs arbitrarily to other gates' outputs can induce combinational loops and timing path distortions. To prevent these issues, ArtNet increases the sink count of an existing PI, keeping stable connectivity and maintaining accuracy of timing paths.

Add PO. A new PO is added by selecting an output net from combinational logic that previously did not have a PO as a sink. This enables the creation of additional output paths without disrupting existing data flows or introducing redundant connections – enabling more flexible and diverse output configurations.

Delete PO. When deleting a PO, adjustments are made only to output nets from source gates that have additional sinks besides the PO. This approach ensures that no floating pins are created and that continuity in timing paths is preserved.

IV. Experimental Evaluation

Our netlist generation framework is implemented in C++ and built on the OpenROAD infrastructure [36]. For our experiments, we use OpenROAD for Section IV-B, while Innovus *v21.1* [37] is used for Sections IV-C, IV-D, and IV-E. The ASAP7 [6] open enablement is employed throughout the experiments. For Section IV-C, Voltus *v21.1* [38] is used. All experiments are conducted on a server equipped with four 2.4 GHz Intel Xeon(R) Gold 6148 processors and 384 GB of RAM. For the experiments in Section IV-D, we train the model using an Nvidia 3090-Ti GPU. All codes and scripts used in this work are posted for review at [40].

A. Runtime Efficiency

As described above, ArtNet's hierarchical clustering starts with N individual instances, and iteratively merges pairs of modules until a single root module is obtained. Fig. 6 plots

runtime versus log of number of instances, and also shows a runtime breakdown across main steps. *Initialization* refers to the stage where the priority queue (Q_{size}) and sequential queue (Q_{seq}) are created from the *SpecFile*. *Clustering* is the process of generating a hierarchical tree from these queues. Finally, *Net Generation* includes both making connections between clusters and inserting flip-flops.

To obtain the data in the figure, we sweep instance counts from 500,000 to 100M with step size 500,000, and sweep p from 0.45 to 0.55 with step size 0.05. Other parameters are fixed: S_{ratio} is set to 0.2, G_{avg} is set to 0.3, and minimum and maximum logic depths are fixed at 1 and 40, respectively, as their impact on complexity is negligible. We cap the maximum runtime at 3 hours; under this constraint, ANG generates netlists up to 250K instances, while GNL succeeds up to 50.5M instances. ArtNet requires between 980 and 3,212 seconds of runtime to generate 100M-instance netlists with these parameter settings, while GNL requires a minimum of 29,000 seconds of runtime to generate 100M-instance netlists and ANG requires a minimum of 38,000 seconds of runtime to generate 500K-instance netlists. This runtime gap is primarily due to ANG's flat, distribution-matching–based graph construction: edges are inserted one by one to reduce mismatches to target statistics (e.g., fan-in/fan-out distributions), and each insertion requires a global search over candidate node pairs. As design size increases, the candidate space grows rapidly, resulting in strongly superlinear runtime scaling.

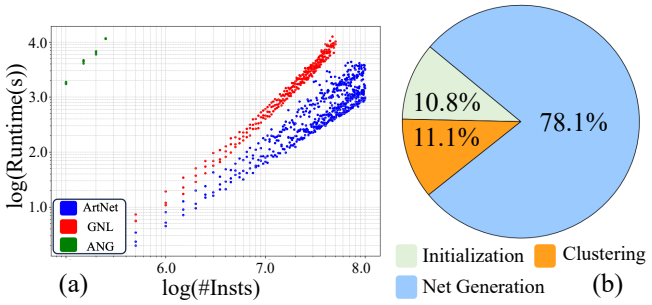


Fig. 6: Runtime analysis: (a) Runtime vs. #Insts; and (b) ArtNet Runtime breakdown.

B. Assessments of Convergence (Stability) and Coverage

We develop new types of analyses to assess artificial netlist generators according to a convergence (stability) criterion, as well as ability to cover combinations of input parameters.

1) Convergence: A fundamental question that has not been well-answered in the artificial netlist generation literature is: What combinations of input parameters should be realizable in artificial netlists? Here, we do not provide any theoretical analysis to advance understanding of this question. However, we empirically assess the *stability* of the netlist generator – specifically, its ability to handle input parameters extracted from artificial netlists *that the generator has itself produced*. Our experiment proceeds in three steps. **(1)** A netlist is generated based on given input parameters. **(2)** From the generated netlist, topological parameters are extracted. **(3)** The extracted parameters are compared against the input parameters to determine if they match, i.e., whether given convergence criteria are met. If the criteria are not met, the extracted parameters are used as new input parameters, and the process loops back to step **(1)**. This iterative cycle continues until the artificial netlist parameters converge according to the desired criteria, or until a maximum limit of 50 iterations have been performed. We define convergence as all parameters of the generated netlist matching input parameters to within 1%.

We highlight both maximum (blue) and minimum (red) values for the target parameter, while tracking the evolution of all parameters throughout the convergence process from start (dashed lines) to end (solid lines). The three main columns in Fig. 7 show how extreme values in one parameter (e.g., maximum $p = 0.80$ in blue, minimum $p = 0.30$ in red, and purple overlap in the left column) affect values taken by other parameters during the convergence process. The narrower bands for ArtNet show its superior convergence behavior as well as smaller impact of any given parameter on other parameters, relative to GNL and ANG. We see that ArtNet not only converges more reliably toward the requested parameter values, but also minimizes cross-parameter interference, which is critical for predictable netlist generation.

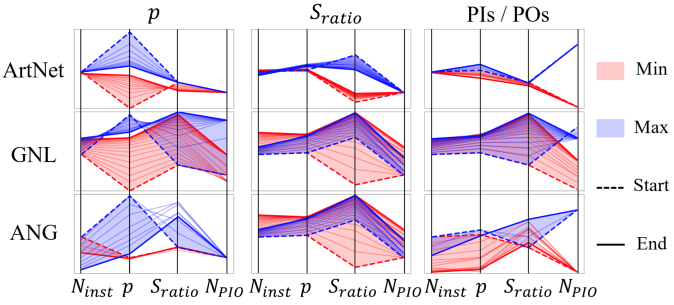


Fig. 7: Convergence plot. The ranges of N_{inst}, p, S_{ratio}, and N_{PIO} (= N_{PI} + N_{PO}) are [0, 20,000], [0.30, 0.80], [0, 0.40], and [0, 1000], respectively (Refer to Table III for notation.)

2) Coverage: We also study the space of parameter combinations for which the netlist generator can successfully produce a well-formed artificial netlist, as well as the space of parameter combinations for which the generator fails.

TABLE IV: Netlist parameter space coverage data.

	Data Points	Excluded	S_{ratio}		p		#Insts	
			< Min	> Max	< Min	> Max	< Min	> Max
ANG	1280	766	0	0	0	725	64	0
GNL	1120	421	0	271	0	100	0	68
ArtNet	1120	145	29	67	3	0	42	8

*The "< Min" and "> Max" counts are calculated independently for each axis.

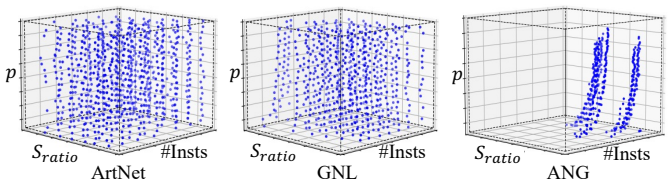


Fig. 8: Coverage plot. The number of instances, S_{ratio}, and p have ranges of [10,000, 200,000], [0.05, 0.30], and [0.35, 0.70], respectively. (Refer to Table III for notation.)

We evaluate the ability of ANG, GNL, and ArtNet to achieve broad parameter space coverage through exploration

of diverse design configurations. Key parameters include #Insts (design space), S_{ratio} (timing path), and p (interconnect complexity). For ArtNet and GNL, we sweep #Insts within [10,000, 200,000] with step size 10,000; we sweep S_{ratio} within [0.05, 0.30] with step size 0.05; and we sweep p within [0.35, 0.70] with step size 0.05. Since ANG does not use p as an input, we sweep alternative parameters related to routing complexity: average net degree within [2.0, 3.0] with step size 0.5, and average bounding box size within [0.1, 1.0] with step size 0.3. This setup results in 1120 data points each for ArtNet and GNL, and 1280 data points for ANG. Table IV shows the netlist parameter coverage data and Fig. 8 visualizes the coverages across the parameter space, where parameter combinations that induce artificial netlists outside of the space are considered to be outliers and excluded from the figure. The excluded outliers comprise 766 points from ANG (mostly due to high p values), 421 points from GNL (due to high S_{ratio}), and 145 points from ArtNet (due to S_{ratio} deviations and low p). As the design size increases, ANG shows increasing p, and struggles to maintain uniform S_{ratio}. For GNL, additional FFs frequently push S_{ratio} beyond the limit, and also cause instance count limits to be exceeded. In contrast, ArtNet shows more stable performance across the range of input parameter combinations, and demonstrates superior coverage. This broad coverage reflects diversity with reduced bias obtained by sweeping input parameters, while accuracy is preserved during construction.

C. Evaluation for Design Benchmarking

1) Cell Type Distribution Analysis: We also assess the robustness of ArtNet with respect to user-defined cell type distributions. To do this, we modify the mix of standard cells in the original design and analyze how well the generated netlists preserve the intended distribution. This lets us determine whether ArtNet is more responsive than other methods in aligning with user-specified cell type constraints.

Specifically, we extract the distribution of cells from the real designs and analyze how well the generated netlists preserve the intended distribution. The similarity between the target and resulting cell type distributions is quantified using cosine similarity, which is defined as: $\textit{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$, where $\mathbf{A}$ and $\mathbf{B}$ are the vectors representing the target and generated cell type distributions, respectively. Table V presents the results of this analysis. Across all designs, ArtNet achieves the highest cosine similarity, demonstrating its ability to closely match the user-defined cell type distributions.

TABLE V: Cell distribution matching evaluation results based on cosine similarity.

	ANG	GNL	ArtNet
jpeg	0.9309	0.8858	**0.9999**
nova	0.7103	0.9984	**1.0000**
netcard	0.9730	0.9967	**0.9999**
tate_pairing	0.9847	0.9778	**1.0000**
CA53	N/A	0.9595	**1.0000**
ariane-133	N/A	**1.0000**	0.9999

2) Rent's Analysis: We also evaluate the similarity of interconnect complexity between artificial netlists and real designs by measuring Rent's parameter using 24 methods in RentCon [34]:

TABLE VI: Methods for Rent's parameter measurement.

Evaluation Method	Pin Counting Method	Averaging Method
Circuit Partitioning	Type1 (Directed Edge)	Arithmetic Mean
Graph Traversal	Type2 (Hyperedge)	
Rect Sampling I		Geometric Mean
Rect Sampling II	Type3 (Undirected Edge)	

We use four different evaluation methods to analyze Rent's parameter:

***Circuit Partitioning-based* [23]:** Recursively partition the netlist with min-cut bisection until each partition contains at least two cells, then compute average cell and pin counts at each level. The classic multilevel circuit partitioner MLPart [3] is incorporated in the program to recursively partition the circuit netlist.

***Graph Traversal-based* [10]:** Perform breadth-first search (BFS) from randomly selected cells to form clusters of varying sizes, and calculate average pin counts per cluster.

Rectangle Sampling I: Randomly sample rectangles of varying sizes with uniformly distributed centers on the chip, ranging from the size of two cell instances to the full chip, and compute average cell and pin counts for each size.

Rectangle Sampling II: Generate multiple random samples per rectangle size on the chip, and compute average pin counts across samples.

Each evaluation method uses different pin counting approaches to analyze the connectivity of cells. We use three different pin counting methods:

Pin Type1 (Directed graph edge model): Graph edge model with signal direction. Each individual source-to-sink connection crossing the boundary is counted as one pin.

Pin Type2 (Hyperedge model): Hyperedge model with or without signal direction. All connections of the same net crossing the boundary are counted as one pin, regardless of how many edges there are.

Pin Type3 (Undirected graph edge model): Graph edge model without signal direction. Each connection crossing the boundary is counted as one pin, but direction is ignored.

Last, to aggregate the results from different pin counting methods, we apply two types of averaging methods:

Arithmetic Mean: The arithmetic mean method simply computes the average of all values obtained across all partitions, clusters, or sampled rectangles.

Geometric Mean: The geometric mean method calculates the mean in a multiplicative sense, which is often more suitable for data with a wide range of values or skewed distributions.

We evaluate Rent's parameter using these 24 different methods, combining four evaluation methods, three pin counting methods, and two averaging methods. This comprehensive study provides a thorough analysis across structural and placement patterns, giving a clear picture of how artificial netlists compare to real designs. We evaluate Rent's exponent on six designs listed in Table VIII. Table VII summarizes the errors between Rent's exponent values of real and artificial designs using mean absolute percentage error (MAPE), mean absolute error (MAE), and median absolute error (MedAE). ArtNet demonstrates consistently low error across all evaluation methods, with particularly strong performance for Type2 pin counting, as it adopts a hyperedge-based formulation that directly models net-level connectivity. Type1 and Type3 results are reported as complementary graph-edge views. In

circuit partitioning method, ArtNet achieves a MAPE of 1.43%, MAE of 0.01, and MedAE of 0.01, significantly outperforming ANG and GNL. In graph traversal, ArtNet maintains superior accuracy with a MAPE of 3.43%, MAE of 0.026, and MedAE of 0.02. Similarly, in rectangle sampling I, ArtNet achieves a MAPE of 5.24%, MAE of 0.026, and MedAE of 0.02, while in rectangle sampling II, it achieves a MAPE of 3.64%, MAE of 0.02, and MedAE of 0.01, demonstrating consistent and precise estimation across evaluation methods. In particular, ArtNet shows significantly lower errors in the circuit partitioning-based method because its recursive clustering closely aligns with the partitioning approach. We see that the netlists generated by ArtNet closely match not only the structural characteristics but also the placement patterns of the real designs.

Differences across pin types are largely attributable to the generators' construction assumptions. ANG builds nets from graph-edge–level connections, which tends to favor Type1 metrics, whereas ArtNet and GNL generate nets directly at the hyperedge level, aligning more closely with Type2 evaluation. Consistent with this, ArtNet attains the lowest error in all 24 Type2 settings and remains competitive under graph-edge metrics. Overall, these results indicate that the netlists generated by ArtNet closely match the net-level interconnect structure of real netlists, while also showing robust behavior under alternative graph-edge–based evaluations.

TABLE VII: Rent's parameter error between real and artificial netlists.

Method	Pin type	Netlist	Arithmetic Mean			Geometric Mean		
			MAPE (%)	MAE	MedAE	MAPE (%)	MAE	MedAE
Circuit Partitioning	Type1	ANG	**5.65**	**0.05**	**0.04**	**6.24**	**0.05**	**0.06**
		GNL	13.2	0.12	0.14	11.77	0.10	0.11
		ArtNet	11.7	0.10	0.11	10.49	0.09	0.08
	Type2	ANG	39.07	0.21	0.22	41.41	0.22	0.22
		GNL	3.79	0.02	**0.01**	6.04	0.03	0.03
		ArtNet	**1.43**	**0.01**	**0.01**	**2.55**	**0.01**	**0.01**
	Type3	ANG	6.62	0.13	0.15	10.55	0.20	0.20
		GNL	5.05	0.10	0.09	8.18	0.15	0.14
		ArtNet	**3.60**	**0.07**	**0.02**	**5.59**	**0.10**	**0.04**
Graph Traversal	Type1	ANG	**3.48**	**0.03**	**0.04**	**3.72**	**0.04**	**0.04**
		GNL	7.11	0.07	0.06	7.08	0.07	0.06
		ArtNet	6.91	0.06	0.06	6.88	0.06	0.06
	Type2	ANG	5.45	0.04	0.04	5.46	0.04	0.04
		GNL	4.00	**0.03**	**0.02**	3.47	0.03	**0.02**
		ArtNet	**3.43**	**0.03**	**0.02**	**2.91**	**0.02**	**0.02**
	Type3	ANG	6.19	0.09	0.09	5.97	0.09	0.09
		GNL	5.19	0.08	0.06	4.90	0.07	0.06
		ArtNet	**2.97**	**0.04**	**0.02**	**3.22**	**0.05**	**0.02**
Rect Sampling I	Type1	ANG	**2.86**	**0.02**	**0.02**	3.83	0.03	0.03
		GNL	3.24	0.03	**0.02**	**2.75**	**0.02**	**0.02**
		ArtNet	4.02	0.03	0.03	3.53	0.03	0.03
	Type2	ANG	22.35	0.12	0.12	24.01	0.12	0.12
		GNL	9.59	0.05	0.03	11.05	0.05	0.04
		ArtNet	**5.24**	**0.03**	**0.02**	**5.88**	**0.03**	**0.02**
	Type3	ANG	**2.61**	**0.04**	**0.03**	**5.92**	**0.08**	0.09
		GNL	7.27	0.10	0.09	7.68	0.10	**0.04**
		ArtNet	5.27	0.08	0.04	6.06	**0.08**	0.05
Rect Sampling II	Type1	ANG	5.26	**0.04**	**0.04**	5.23	**0.04**	**0.04**
		GNL	**5.08**	**0.04**	0.05	**5.06**	**0.04**	0.05
		ArtNet	5.57	0.05	0.05	5.31	**0.04**	0.05
	Type2	ANG	29.94	0.17	0.17	29.79	0.17	0.17
		GNL	6.49	0.04	0.03	7.54	0.04	0.03
		ArtNet	**3.64**	**0.02**	**0.01**	**3.61**	**0.02**	**0.01**
	Type3	ANG	8.18	0.13	0.10	8.13	0.13	0.10
		GNL	**4.52**	**0.07**	**0.04**	**4.49**	**0.07**	**0.04**
		ArtNet	4.89	**0.07**	**0.04**	5.00	**0.07**	0.05

*The value closest to Real (target) is highlighted in **Bold**. ArtNet "wins" $36\frac{1}{3}$ of 72 (4 methods $\times$ 3 types $\times$ 6 metrics) head-to-head comparisons against ANG ($21\frac{5}{6}$) and GNL ($13\frac{5}{6}$), where k-way ties are counted as $\frac{1}{k}$ fractional wins.

3) P&R Evaluation: Topological features such as circuit size, interconnect complexity, and timing depth significantly influence P&R behavior. We experimentally confirm that ArtNet's artificial netlists designed to mimic real circuit topologies can achieve comparable P&R QoR metrics, validating their use as reliable proxies for design benchmarking. For our experiments, we use jpeg, nova, netcard, tate_pairing, Arm Cortex-A53 (CA53), and ariane-133 [40]. Table VIII shows the QoR metrics for each circuit from real, ANG, GNL, and ArtNet versions after running commercial P&R flow. All evaluations use identical design parameters and the same tool flow for consistency.

We compare artificial netlists generated by ANG, GNL and ArtNet across three design stages. At pre-placement, we evaluate the number of instances (#Insts), number of nets (#Nets), number of flip-flops (#FFs) and Rent's exponent (p), directly extracted from the netlists. At post-placement, we evaluate half-perimeter wire length (HPWL) and pin density[4] (PinD). At the post-routing stage, we evaluate routed wirelength (rWL), worst negative slack (WNS), total negative slack (TNS), total power (Power), and the number of DRVs (#DRVs). We note that ANG fails to generate designs for CA53 and ariane-133 due to its lack of macro insertion support. Also, while ANG can process netcard and tate_pairing parameters, the ANG-produced netlists fail routing due to excessive interconnect complexity ($>$ 8.5M DRVs and $>$ 3M DRVs, respectively).

QoR metrics in Table VIII are presented using various comparison metrics in Tables IX and X for additional analysis and insight. Table IX presents average errors of ANG, GNL, and ArtNet compared to real circuits, highlighting the gap between the generated and real designs in terms of design features. ANG exhibits average errors of 7.76% in #Insts, 7.72% in #Nets, 40.91% in #FFs, and 40.21% in p compared to real designs. GNL shows slightly better accuracy, with average errors of 4.11% in #Insts, 4.36% in #Nets, 32.00% in #FFs, and 4.02% in p. ArtNet has the best performance, with average errors of 0.21% in #Insts, 0.38% in #Nets, 1.19% in #FFs, and 1.43% in p. ArtNet outperforms both ANG and GNL in other accuracy of reproduction metrics as well, e.g., HPWL (13.90%), pin density (2.66%), rWL (9.68%), power (7.49%), area (7.34%), VT Ratio (2.03%), and CTS leakage power (9.80%). We conclude that ArtNet has stronger correlations than ANG and GNL with the real circuits for every QoR metric. Furthermore, it also demonstrates superior runtime, with 4793$\times$ speedup over ANG and 2.30$\times$ speedup over GNL.

Table X compares the changes in area and #Insts during the downstream optimization design process, assuming that a well-replicated netlist should exhibit similar design feature changes to those in real designs, as well as similar design feature values in Table VIII. Each delta value is calculated by subtracting the pre-place stage value from the post-route optimization stage value and dividing the result by the pre-place stage value. On average across six designs, ArtNet undergoes a 4.30% change in area and a 5.90% change in #Insts, showing the closest alignment with the changes observed in the real design. Although ANG achieves better results than ArtNet for jpeg,

[4]Pin density is defined as the number of pins divided by the core area.

TABLE VIII: P&R evaluation.

Design		Pre-Place				Post-Place		Post-Route							Runtime
		#Insts	#Nets	#FFs	p	HPWL	PinD	Area	CTS Leak	rWL	WNS	TNS	Power	#DRVs	(s)
jpeg	Real	46,593	46,612	4,392	0.49	1.000	0.314	5,456	7.87	1.000	-37	-5.61	0.071	0	-
	ANG	45,627	45,645	8,741	0.73	4.089	0.236	6,353	12.68	4.176	-114	-157.2	0.109	1,028	257.3
	GNL	50,569	50,571	8,368	0.54	1.327	0.272	7,307	**9.96**	1.366	-16	**-1.175**	0.104	**0**	0.41
	ArtNet	**46,682**	**46,703**	**4,482**	**0.50**	**1.054**	**0.309**	**5,620**	5.29	**1.013**	**-39**	-0.816	**0.069**	**0**	**0.23**
nova	Real	142,863	143,261	29,131	0.57	1.000	0.306	19,037	31.81	1.000	-122	-24.64	0.059	6	-
	ANG	120,704	121,169	25,890	0.76	2.062	0.209	15,019	39.40	2.121	**-22**	-1.087	0.092	24	3,160
	GNL	145,720	146,409	31,988	0.59	1.297	0.293	24,204	37.32	1.347	-1466	-27.7k	0.155	0	1.82
	ArtNet	**142,547**	**143,236**	**28,815**	**0.56**	**1.088**	**0.299**	**17,257**	**32.58**	**1.015**	-367	**-42.03**	**0.071**	**0**	**0.76**
netcard	Real	261,591	263,416	67,440	0.58	1.000	0.251	42,488	75.04	1.000	-28	-0.574	0.341	5	-
	ANG	238,861	240,700	59,926	0.81	4.290	0.209	N/A	N/A	N/A	N/A	N/A	N/A	$>$ 8.5M	18,427
	GNL	270,180	272,024	76,029	0.60	1.133	0.227	47,447	80.81	1.037	0.00	0.00	0.575	**0**	3.95
	ArtNet	**260,113**	**261,957**	**65,962**	**0.57**	**1.113**	**0.242**	**38,073**	**71.34**	**0.983**	**-4**	**-0.01**	**0.359**	**0**	**1.61**
tate_pairing	Real	211,468	212,246	31,416	0.54	1.000	0.348	23,772	36.97	1.000	-20	-1.451	0.303	0	-
	ANG	201,401	202,179	44,716	0.75	5.987	0.265	N/A	N/A	N/A	N/A	N/A	N/A	$>$ 3M	3525
	GNL	223,663	224,442	43,611	0.56	1.763	0.305	34,149	49.19	1.779	-161	-1419.5	0.543	**1**	3.14
	ArtNet	**211,177**	**211,956**	**31,125**	**0.54**	**1.453**	**0.338**	**24,655**	**35.77**	**1.406**	**-11**	**-0.265**	**0.322**	**1**	**1.47**
CA53	Real	303,242	305,294	35,234	0.65	1.000	0.098	39,746	48.37	1.000	-104	-94.26	0.392	202	-
	ANG	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	-
	GNL	317,373	320,006	49,365	**0.66**	1.785	0.092	63,630	73.02	1.727	-610	-13k	0.827	9,734	20.34
	ArtNet	**303,251**	**306,235**	**35,243**	0.63	**0.985**	**0.096**	**35,974**	**51.85**	**0.961**	**-290**	**-317.3**	**0.431**	**649**	**13.49**
ariane-133	Real	97,329	100,000	19,807	0.58	1.000	0.056	15,909	0.38	1.000	-21	-2.54	0.341	3	-
	ANG	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	-
	GNL	97,755	101,625	**19,807**	0.59	1.376	**0.058**	19,457	**0.35**	1.385	**-7**	**-0.015**	0.405	14	2.06
	ArtNet	**97,496**	**101,051**	19,975	**0.58**	**1.111**	**0.058**	**14,618**	**0.35**	**1.091**	0.00	0.00	**0.342**	**0**	**0.69**

*Area is in units of μm^2; CTS Leak is in units of μW; WNS is in units of ps; TNS is in units of ns; and Power is in units of W.
The value closest to Real (target) is highlighted in **Bold**.

TABLE IX: Mean absolute percentage errors (MAPEs) of ANG, GNL and ArtNet compared to the real circuits.

MAPE (%)	ANG	GNL	ArtNet
#Insts	7.76	4.11	**0.21**
#Nets	7.72	4.36	**0.38**
#FF	40.91	32.00	**1.19**
p	40.21	4.02	**1.43**
HPWL	310.70	44.68	**13.90**
Pin Density	24.28	8.21	**2.66**
rWL	214.85	44.02	**9.68**
Power	54.73	81.13	**7.49**
Area	18.77	33.13	**7.34**
VT Ratio	4.26	2.60	**2.03**
CTS Leak	42.47	20.95	**9.80**

TABLE X: Impact of downstream optimization steps on Area and #Insts.

Design	Real		ANG		GNL		ArtNet	
	ΔArea	Δ#Insts	ΔArea	Δ#Insts	ΔArea	Δ#Insts	ΔArea	Δ#Insts
jpeg	8.04	9.96	8.15	4.59	17.67	32.34	10.68	20.48
nova	5.08	8.31	-9.98	-20.52	29.14	42.05	-4.23	-6.24
netcard	9.99	10.91	N/A	N/A	13.62	16.27	2.36	7.62
tate	3.67	2.34	N/A	N/A	28.07	36.19	7.92	7.89
CA53	10.91	16.45	N/A	N/A	58.01	77.40	7.92	5.47
ariane	8.92	14.31	N/A	N/A	33.48	60.48	1.13	0.18
Avg.	7.77	10.38	-0.92	-7.97	30.00	44.12	**4.30**	**5.90**

*Δ(%) = (Value@post-route-opt - Value@pre-place) / (Value@pre-place).
The value closest to Real (target) is highlighted in **Bold**.

Table VIII shows that ArtNet, overall, generates netlists more similar to the real design.

4) Design Locality: Fig. 9 illustrates ArtNet's ability to reflect both structural and timing-related locality of real design. While logical hierarchy-aware generation improves locality over flat generation, differences from real designs remain. We use ArtNet to generate two netlists: (1) ArtNet-generated nova with logical hierarchy, and (2) ArtNet-generated nova without logical hierarchy. Then, we compare these netlists on four different aspects: clustering results; timing critical paths; routing congestion map; and static IR drop.

First, we evaluate the structural locality of three netlists using the Leiden community detection algorithm [30]. Each detected cluster is visualized with a unique color after placement to highlight spatial coherence (Fig. 9(a)). Table XI summarizes the clustering metrics. The real netlist forms 62 clusters with a modularity [25] of 0.94 and an average conductance [27] of 0.02. The hierarchy-aware netlist yields 52 clusters with similar modularity (0.95) and even lower conductance (0.01). In contrast, the flat netlist produces fewer clusters (45) with comparable modularity (0.95) but higher conductance (0.05), suggesting blurred boundaries and degraded locality. We observe that incorporating logical hierarchy in ArtNet helps to preserve the modular structure seen in real designs.

TABLE XI: Clustering characteristics comparison data.

Design	#Clusters	Modularity	Conductance
nova-real	62	0.94	0.02
nova-flat	45	0.95	0.05
nova-hier	52	0.95	0.01

Second, we examine the distribution of timing-critical paths. Fig. 9(b) shows the spatial distribution of timing-critical paths for three netlists. In both the real and hierarchy-aware netlists, critical paths are localized within compact regions, while in the flat netlist, they are scattered across the layout. As summarized in Table XII, the flat netlist suffers from significantly worse timing, with a WNS of -303ps and TNS of -24.88ns, compared to -28ps and -1.99ns in real and -154ps and -1.35ns in hierarchy-aware netlists, respectively. These results confirm that preserving logical hierarchy not only enhances structural locality but also leads to more realistic timing behavior.

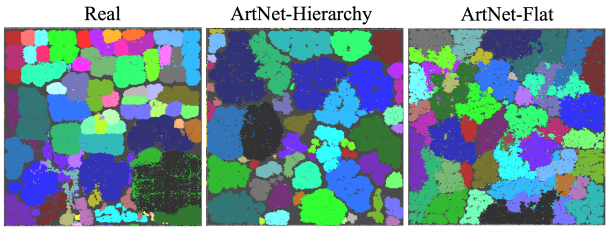
TABLE XII: Timing characteristics comparison data.

Design	WNS (ps)	TNS (ns)	#FEPs
nova-real	-28	-1.99	140
nova-flat	-303	-24.88	192
nova-hier	-154	-1.35	24

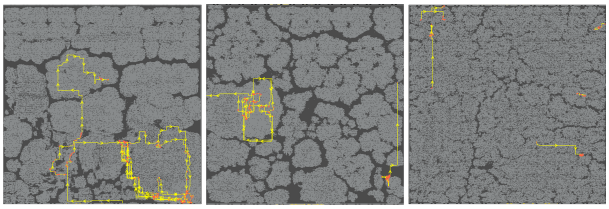
*#FEPs = number of failing endpoints.

Next, we analyze the routing congestion map. Fig. 9(c) shows the congestion distribution across metal layers M1 to

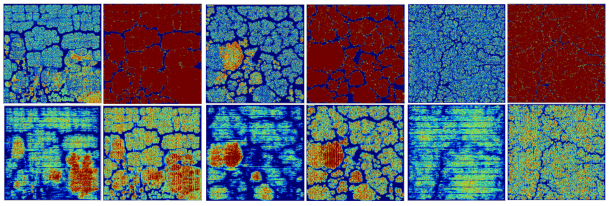
M4. In both the real and hierarchy-aware netlists, congestion is concentrated in specific regions. In contrast, the flat netlist shows a more uniform congestion spread, indicating a loss of spatial locality. Last, we examine the IR drop behavior after routing. Fig. 9(d) shows the IR drop distribution. In both the real and hierarchy-aware netlists, IR drop hotspots can be observed. The flat netlist, however, shows more diffuse IR drop and a loss of locality. While ArtNet captures several key aspects of design locality, we acknowledge that noticeable differences remain compared to real designs.



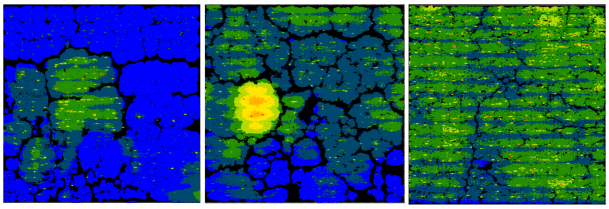
(a) Clustering results.



(b) Timing-critical paths.



(c) Routing congestion map (M1-M4, shown clockwise from top-left).



(d) Static IR drop.

Fig. 9: ArtNet achieves realistic design locality. Real Design nova (left); ArtNet-generated nova with logical hierarchy (center); and ArtNet-generated nova without logical hierarchy (right).

D. Assessment in an ML Framework

We demonstrate a practical use case of artificial netlists in the ML context. Fig. 10 illustrates the data augmentation flow with netlist generator. We generate augmented datasets using ANG and ArtNet, apply these datasets in the training of ML models, evaluate model performance on unseen circuits, and compare against a baseline trained on a real dataset.

Design of Experiments. We train three CNN-based routability prediction models [26], [39] respectively using three datasets:[5] (1) M_{real} with real netlist dataset, (2) M_{ANG} with augmented dataset from ANG, and (3) M_{ArtNet} with augmented dataset from ArtNet. For the real design dataset, we generate 64 layouts per design, varying utilization, top routing layer, clock period, and aspect ratio. The real designs [35] used are as follows:

- **Training dataset.** ldpc_decoder_802_3an, eth_top, ibex_core, mpeg2_top, point_scalar_mult, tate_pairing, vga_enh_top, des3, fpu, keccak, pci_bridge32, sasc_top, tv80s, wb_conmax_top
- **Test dataset.** aes_cipher_top, netcard, nova

To create the augmented datasets, we use the Sobol sequence [28] for efficient parameter sampling with broad coverage of the parameter space. Table XIII provides the parameter space, determined based on the real design training dataset. We sample 64 configurations and generate 64 netlists, each with 18 layouts that vary in utilization, top routing layer, clock period, and aspect ratio, for a total of 1,152 layouts. The final augmented dataset combines both real and artificial datasets. Due to the tile-and-crop training pipeline of the CNN-based model, each layout is decomposed into many tiles, resulting in an effective training set that is much larger than the number of layouts and sufficient for mini-batch training.

TABLE XIII: Parameter space of training dataset.

	ArtNet / ANG			ArtNet		ANG		
	N_{PI}	N_{PO}	S_{ratio}	T_{avg}	p	D_{avg}	B_{avg}	O_{avg}
Min	14	32	0.00	2.01	0.495	2.10	0.09	3.51
Max	1131	1416	0.29	3.41	0.651	2.94	0.59	130.73
Avg.	297	350	0.13	2.58	0.57	2.41	0.28	18.15
Std.	345.5	465.6	0.10	0.36	0.06	0.23	0.18	36.03

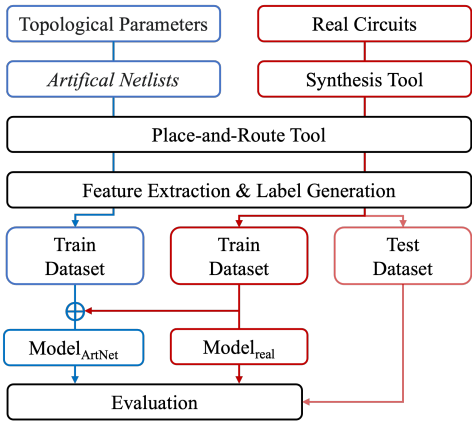


Fig. 10: Data augmentation flow with netlist generator.

Model Performance Evaluation. Table XIV compares the models using accuracy, recall, precision, and F1 score. The M_{real} model achieves 0.93 accuracy, 0.55 recall, 0.67 precision, and F1 score of 0.61. With the benefit of data augmentation, the M_{ANG} model improves all four metrics. The M_{ArtNet} model achieves the best results, with 0.96 accuracy, 0.72 recall, 0.76 precision, and F1 score of 0.74. For completeness, we further conduct a data-ablation study by training with 25%, 50%, 75%, and 100% of the ArtNet-augmented dataset, and with an ArtNet-only setting in which training uses only ArtNet-generated layouts (i.e., excluding all real-layout samples). We observe that performance largely saturates once 50% of the ArtNet-augmented data is used, and the ArtNet-only model remains comparable to the real-data baseline.

Dataset Analysis. Table XV presents a statistical comparison between the ANG and ArtNet datasets based on four key

[5]We adopt the same ML model used in ANG [21] for a fair comparison.

TABLE XIV: Comparison of CNN model performance.

Model	Accuracy	Recall	Precision	F1
M_{real}	0.93	0.55	0.67	0.61
M_{ANG}	0.94	0.68	0.74	0.71
M_{ArtNet}	0.96	0.72	0.76	0.74
$M_{ArtNet\_75\%}$	0.96	0.71	0.77	0.73
$M_{ArtNet\_50\%}$	0.94	0.75	0.70	0.71
$M_{ArtNet\_25\%}$	0.93	0.60	0.69	0.64
M_{ArtNet_only}	0.92	0.58	0.60	0.59

metrics: variance (σ^2) [12], mean pairwise distance ($\bar{d}$) [2], and mean ($\bar{\lambda}$) and standard deviation of eigenvalues (STD(λ)) of the CNN feature map (treated as a matrix, following [14]). We calculate these metrics from data points within a single layout to reflect the feature diversity. Larger variance and higher mean pairwise distance indicate greater sample diversity. Higher mean eigenvalues suggest more variability captured by principal components, as the sum of the eigenvalues equals the total variance in the dataset. A lower standard deviation implies a more even distribution, as eigenvalues of roughly the same size mean the variance is spread out evenly.

TABLE XV: Statistical comparison of datasets.

	σ^2	d	λ	STD(λ)
ANG	0.80	8.04	0.80	0.16
ArtNet	**0.83**	**8.32**	**0.83**	**0.15**

For the ANG dataset, the variance is 0.80, the mean pairwise distance is 8.04, the mean of eigenvalues is 0.80, and the standard deviation of eigenvalues is 0.16. By contrast, the ArtNet dataset shows a higher variance of 0.83, a greater mean distance of 8.32, an eigenvalue mean of 0.83, and a lower standard deviation of 0.15. These results indicate that the ArtNet dataset has a broader data spread and more consistent eigenvalue distribution.

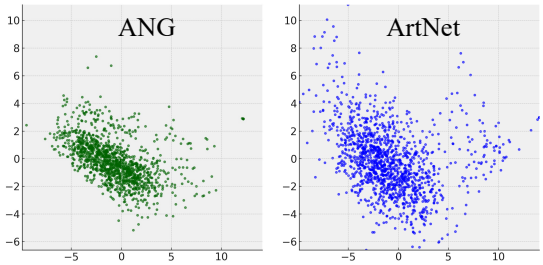


Fig. 11: PCA projection of the ANG (left) and ArtNet (right) datasets.

Fig. 11 illustrates feature diversity for both datasets using a principal component analysis (PCA) plot [14]. The ANG dataset exhibits a clustered and skewed distribution, indicating limited variability and potential feature redundancy. In contrast, ArtNet dataset presents a more uniformly dispersed distribution that spans a wider area of the feature space, although some skewness remains due to the inherent correlations among layout metrics. Such a broader distribution implies richer structural information, which is consistent with extraction of more effective features that ultimately result in better model performance [11].

E. Assessment in DTCO Context: Mini-Brains

We explore a novel *mini-brains* approach [16] that enables more efficient design-technology co-optimization, helping to mitigate the increasing complexity and scale of real modern designs. Our *mini-brain* methodology generates artificial netlists scaled to 10% of a full-scale design's size, by reducing only circuit-size–related parameters (N_{insts}, N_{PI}, N_{PO} and N_{macro}), with the goal of replicating the PPA characteristics of full-scale designs while significantly reducing P&R runtimes. The reduction in implementation flow runtime is primarily due to the reduced netlist size.

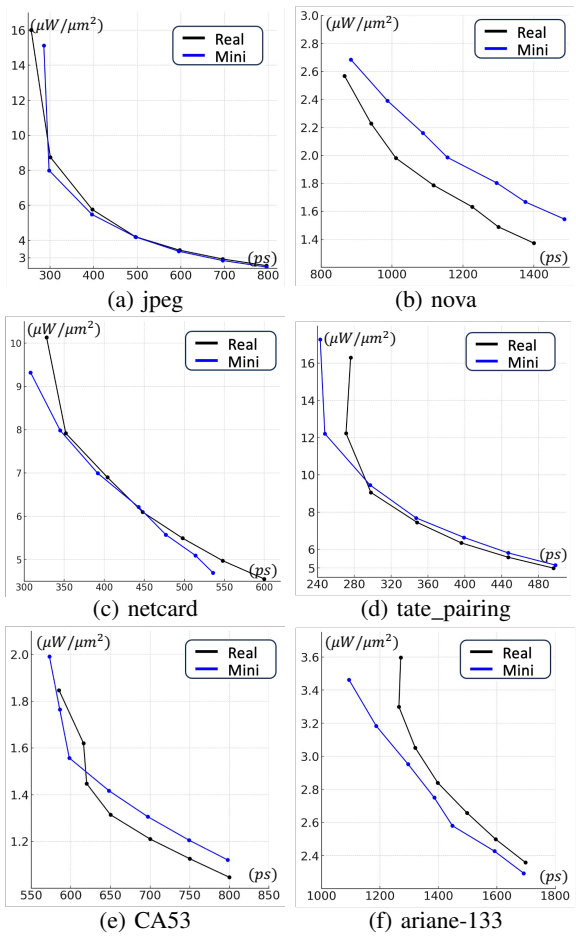


Fig. 12: DTCO *mini-brain* evaluation. Plot of power per unit area (μW/μm^2) vs. effective clock period (ps).

TABLE XVI: MAPE loss and runtime reduction results.

Design	MAPE Loss (%)			P&R Runtime Reduction
	ANG	GNL	ArtNet	
jpeg	2.66	18.17	**2.06**	-79.26%
nova	27.55	133.37	**7.33**	-91.45%
netcard	31.37	14.19	**3.74**	-93.74%
tate_pairing	11.58	34.44	**3.43**	-88.58%
CA53	N/A	8.26	**4.70**	-87.50%
ariane-133	N/A	169.61	**3.50**	-95.53%

1) PPA-Matching Evaluation: To assess reliability, we conduct robustness tests by sweeping target clock periods and calculating the average of two metrics; the MAPE of the effective clock period and the MAPE of total power per area. Fig. 12 shows that the mini-brain closely tracks the PPA trends of the real design across a wide range of target clock periods. On the y-axis, power values are divided by area to enable comparison between designs of different sizes. The overall loss is defined as:

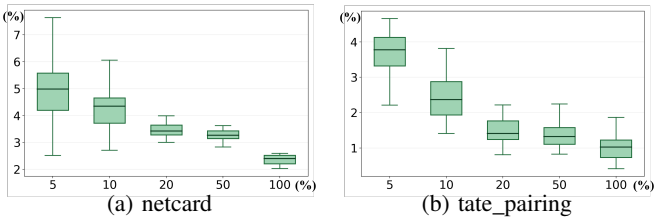
$$\mathcal{L} = \frac{1}{2}\left(\text{MAPE}_{\text{clock}} + \text{MAPE}_{\text{power}}\right) \tag{5}$$

where

$$\text{MAPE}_{\text{clock}} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{T_{\text{real}}^{(i)} - T_{\text{mini}}^{(i)}}{T_{\text{real}}^{(i)}}\right| \times 100, \quad (6)$$

$$\text{MAPE}_{\text{power}} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{P_{\text{real}}^{(i)} - P_{\text{mini}}^{(i)}}{P_{\text{real}}^{(i)}}\right| \times 100. \quad (7)$$

Here, $T_{\text{real}}^{(i)}$ and $T_{\text{mini}}^{(i)}$ respectively denote the effective clock periods of the real design and its *mini-brain* for the i^{th} sample. Likewise, $P_{\text{real}}^{(i)}$ and $P_{\text{mini}}^{(i)}$ respectively denote the total power per area of the real design and its *mini-brain*. Table XVI shows the potential of *mini-brains* to accurately replicate PPA metrics while significantly reducing P&R runtime compared to full-scale designs, particularly in the DTCO context. We also study how the MAPE loss changes with the mini-brain scale factor. Fig. 13 shows box plots of distributions of MAPE loss over 50 runs with random seeds (1, 2, ..., 50). As the mini-brain scale becomes closer to the real design (i.e., 100%), both the MAPE loss and the variance of its distribution decrease.



(a) netcard (b) tate_pairing

Fig. 13: Tukey box plots (quartiles, with (min, max) whiskers truncated at 1.5 × IQR (inter-quartile range)) of MAPE loss (%) distribution for 50 random seeds, across mini-brain scale factors (%).

2) BEOL Parameter Evaluation: We demonstrate a practical use case of ArtNet-generated *mini-brains* in the DTCO context by evaluating design sensitivity to various BEOL parameter configurations. As illustrated in Fig. 14, *mini-brains* enable a faster DTCO loop by serving as lightweight surrogates for real designs.

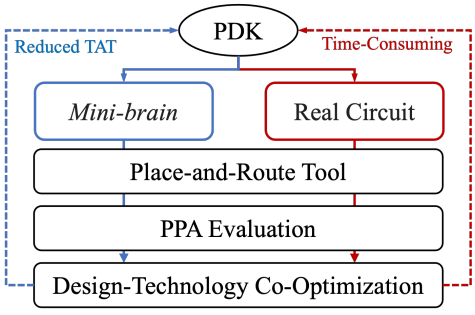


Fig. 14: DTCO exploration flow with *mini-brains*.

Design of Experiments. To assess the effectiveness of DTCO *mini-brains* in BEOL parameter exploration, we conduct a BEOL parameter sweep across multiple metal layer configurations. As shown in Table XVII, we sweep four attributes related to the M3 and M5 metal layers: pitch and capacitance, resulting in a total of 81 BEOL configurations per design. For each configuration, we perform independent P&R runs on both the full design and its corresponding *mini-brain*, allowing direct comparison of their sensitivity to BEOL variation.

TABLE XVII: BEOL parameter space.

Parameter	Description	Option
M3P	M3 layer (Mx) pitch	[0.85, 1.00, 1.30]
M5P	M5 layer (My) pitch	[0.75, 1.00, 1.25]
M3C	M3 layer (Mx) capacitance	[0.50, 1.00, 1.50]
M5C	M5 layer (My) capacitance	[0.50, 1.00, 1.50]

* Default value for each parameter is 1.00.

Ranking Consistency Evaluation. We evaluate whether *mini-brains* can reliably preserve the relative ranking of design outcomes under different BEOL settings. For each *mini-brain* and each real design, we rank all 81 BEOL configurations in ascending order of total power. For each full design and corresponding *mini-brain*, we evaluate rank correlation of outcomes using Kendall and Spearman correlation coefficients. Table XVIII shows consistently high rank correlation and low MAPE for power per unit area across diverse designs, indicating that *mini-brains* successfully capture the sensitivity of design quality to BEOL variation. This confirms that *mini-brains* are not only efficient to generate but also accurate enough to guide BEOL parameter exploration.

TABLE XVIII: DTCO mini-brain evaluation results. Design quality rank correlation between real and artificial designs, across 81 BEOL parameter combinations and MAPE for power per unit area (μW/μm^2).

Design	Kendall	Spearman	MAPE (%)
jpeg	0.621	0.823	2.37
nova	0.662	0.856	5.21
netcard	0.623	0.802	1.14
tate_pairing	0.760	0.931	4.98
CA53	0.752	0.915	7.26
ariane-133	0.646	0.851	2.99

V. Conclusion

The ***ArtNet*** artificial netlist generator is built on top of the OpenROAD infrastructure, and is designed to expand ML training datasets and accelerate DTCO exploration. By leveraging hierarchical clustering and incorporating timing path characteristics, ArtNet serves as a reliable proxy for real circuits in design benchmarking and performance evaluation. By matching key attributes spanning topology, timing, and interconnect complexity with high fidelity, ArtNet enables accurate P&R benchmarking while significantly reducing runtime compared to previous methods. Its ability to achieve broader parameter space coverage enables better exploration of diverse design configurations, enhancing both ML data augmentation and DTCO applications. In an ML context (CNN-based DRV prediction), ArtNet data augmentation improves F1 scores by 0.16 compared to models trained only on real datasets. In a DTCO application, ArtNet-generated *mini-brains* achieve a strong PPA match (MAPE loss as low as 2.06%) to full-scale designs, while reducing P&R runtime by up to 95.53%. Furthermore, *mini-brains* serve as a fast and scalable proxy in the DTCO loop, enabling rapid PPA evaluation across various BEOL configurations. These results establish ArtNet as a powerful tool for advancing innovation in circuit design, offering both accuracy and efficiency for next-generation design flow. In combination with open-sourcing and OpenROAD integration, we believe that this work will add to the foundations for new research on artificial data generation in the EDA field.

ACKNOWLEDGMENT

This work is partially supported by the Samsung AI Center. The authors acknowledge the use of ChatGPT for improving the English phrasing and grammatical quality of the manuscript.

REFERENCES

[1] S. An, H. Lee, J. Jo, S. Lee and S. J. Hwang, "DiffusionNAG: Predictor-guided Neural Architecture Generation with Diffusion Models", *Proc. ICLR*, 2023.
[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York: Springer, 2002.
[3] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. ASPDAC*, 2000.
[4] C.-K. Cheng, A. B. Kahng, H. Kim, M. Kim, D. Lee, D. Park and M. Woo, "PROBE2.0: A Systematic Framework for Routability Assessment from Technology to Design in Advanced Nodes", *IEEE TCAD* 41(5) (2022), pp. 1495-1508.
[5] S. Choi, J. Jung, A. B. Kahng, M. Kim, C.-H. Park, B. Pramanik and D. Yoon, "PROBE3.0: A Systematic Framework for Design-Technology Pathfinding with Improved Design Enablement", *IEEE TCAD* 43(4) (2024), pp. 1218-1231.
[6] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy and G. Yeric, "ASAP7: A 7nm FinFET Predictive Process Design Kit", *Microelectronics Journal* 53 (2016), pp. 105-115.
[7] J. Darnauer and W. W. Dai, "A Method for Generating Random Circuits and Its Application to Routability Measurement", *Proc. FPGA*, 1996.
[8] D. Gailhard, E. Tartaglione, L. Naviner and J. H. Giraldo, "HYGENE: A Diffusion-Based Hypergraph Generation Method", *Proc. AAAI*, 2025.
[9] X. Guo and L. Zhao, "A Systematic Survey on Deep Generative Models for Graph Generation", *IEEE TCAD*, 45(5) (2023), pp. 5370-5390.
[10] L. Hagen, A. B. Kahng, F. Kurdahi and C. Ramachandran, "On the Intrinsic Rent Parameter and New Spectra-Based Methods for Wireability Estimation", *IEEE TCAD*, 13(1) (1994), pp. 27-37.
[11] S. H. Hasanpour, M. Rouhani, M. Fayyaz and M. Sabokrou, "Let's Keep It Simple, Using Simple Architectures to Outperform Deeper and More Complex Architectures", *ArXiv 1608.06037*, https://arxiv.org/abs/1608.06037, 2016.
[12] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York: Springer, 2002.
[13] M. D. Hutton, J. Rose, J. P. Grossman and D. G. Corneil, "Characterization and Parameterized Generation of Synthetic Combinational Benchmark Circuits", *IEEE TCAD* 17(10) (1998), pp. 985-996.
[14] I. T. Jolliffe, *Principal Component Analysis*, 2nd edition, New York: Springer, 2002.
[15] J. Jordon, L. Szpruch, F. Houssiau, M. Bottarelli, G. Cherubin, C. Maple, S. N. Cohen and A. Weller, "Synthetic Data – What, Why and How?", *ArXiv 2205.03257*, https://arxiv.org/abs/2205.03257, 2022.
[16] A. B. Kahng, "AI, EDA and Disruptive Innovation", *IEEE CEDA DAC keynote*, 2024.
[17] A. B. Kahng, "Machine Learning Applications in Physical Design: Recent Results and Directions", *Proc. ISPD*, 2018.
[18] A. Kahng, A. B. Kahng, H. Lee and J. Li, "PROBE: Placement, Routing, Back-End-of-Line Measurement Utility", *IEEE TCAD* 37(7) (2018), pp. 1459–1472.
[19] A. B. Kahng and S. Kang, "Construction of Realistic Gate Sizing Benchmarks With Known Optimal Solutions", *Proc. ISPD*, 2012.
[20] A. B. Kahng, S. Kundu and D. Yoon, "Placement Tomography-Based Routing Blockage Generation for DRV Hotspot Mitigation", *Proc. ICCAD*, 2024.
[21] D. Kim, S.-Y. Lee, K. Min and S. Kang, "Construction of Realistic Place-and-Route Benchmarks for Machine Learning Applications", *IEEE TCAD* 42(6) (2023), pp. 2030-2042.
[22] P. D. Kundarewich and J. Rose, "Synthetic Circuit Generation Using Clustering and Iteration", *IEEE TCAD* 23(6), 2004, pp. 869-887.
[23] B. S. Landman and R. L. Russo, "On a Pin Versus Block Relationship for Partitions of Logic Graphs", *IEEE TC* 20(12) (1971), pp. 1469-1479.
[24] M. Li, V. Shitole, E. Chien, C. Man, Z. Wang, S. Sridharan, Y. Zhang, T. Krishna and P. Li, "LayerDAG: A Layerwise Autoregressive Diffusion Model for Directed Acyclic Graph Generation", *Proc. ICLR*, 2025.
[25] M. E. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks", *Phys. Rev. E* 69 (2004), pp. 1–15.
[26] S. Park, D. Kim, S. Kwon and S. Kang, "Routability Prediction and Optimization Using Explainable AI", *Proc. ICCAD*, 2023.
[27] S. Fortunato, "Community Detection in Graphs", *Physics Reports*, 486(3-5) (2010), pp. 75-174.
[28] I. M. Sobol, "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals", *USSR Comput. Math. Math. Phys.* 7(4) (1967), pp. 86–112.
[29] D. Stroobandt, P. Verplaetse and J. V. Campenhout, "Generating Synthetic Benchmark Circuits for Evaluating CAD Tools", *IEEE TCAD* 19(9) (2000), pp. 1011-1022.
[30] V. A. Traag, L. Waltman and N. J. van Eck, "From Louvain to Leiden: Guaranteeing Well-Connected Communities", *Scientific Reports* 9 (2019).
[31] P. Verplaetse, D. Stroobandt and J. V. Campenhout, "Synthetic Benchmark Circuits for Timing-driven Physical Design Applications", *Proc. ICVLSI*, 2002.
[32] K. Ye, Q. Yang, Z. Lu, H. Yu, T. Cui, R. Bai and L. Shen, "LLM4Netlist: LLM-enabled Step-based Netlist Generation from Natural Language Description", *IEEE JETCAS*, early access, 2025.
[33] M. Zhang, S. Jiang, Z. Cui, R. Garnett and Y. Chen, "D-VAE: A variational autoencoder for directed acyclic graphs", *Proc. NeurIPS*, 2019.
[34] K. Jeong, A. B. Kahng and H. Yao, "Rent Parameter Evaluation Using Different Methods", 2008, https://vlsicad.ucsd.edu/WLD/RentCon.pdf
[35] OpenCores: Open Source IP-Cores. http://www.opencores.org
[36] The OpenROAD Project (GitHub). https://github.com/The-OpenROAD-Project/OpenROAD
[37] Cadence Innovus Implementation System. version 21.1. www.cadence.com
[38] Cadence Voltus Implementation System. version 21.1. www.cadence.com
[39] XAI_RoutOpt. https://github.com/shypark98/XAI_RoutOpt
[40] ArtNet GitHub for review. https://github.com/shypark98/ArtNet
[41] ArtNet SpecFile Generation Flow. https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/par#artnet-spec-file-generation-flow

Andrew B. Kahng is a distinguished professor in the CSE and ECE Departments of the University of California at San Diego. His interests include IC physical design, the design-manufacturing interface, large-scale combinatorial optimization, AI/ML for EDA and IC design, and technology roadmapping. He received the Ph.D. degree in Computer Science from the University of California at San Diego.

Seokhyeong Kang received the B.S. and M.S. degrees in electrical engineering from Pohang University of Science and Technology (POSTECH), Korea, in 1999 and 2001, respectively, and the Ph.D. degree in Electrical and Computer Engineering from the University of California, San Diego, La Jolla in 2013. From 2001 to 2008, he was with the System-on-Chip Development Team at Samsung Electronics, Suwon, Korea. He is currently an Associate Professor in the Department of Electrical Engineering at POSTECH. His research interests include IC physical design and AI/ML-driven EDA.

Seonghyeon Park received the B.S. degree in electrical engineering from Pohang University of Science and Technology (POSTECH), Pohang, South Korea in 2021. He is currently pursuing the Ph.D. degree with POSTECH, Pohang, South Korea. His research interests include VLSI physical design and ML-based EDA.

Dooseok Yoon received the M.S. degree in electrical and computer engineering from the University of California at San Diego, La Jolla, CA, USA, in 2024. He is currently pursuing the Ph.D. degree at the University of California at San Diego, La Jolla. His research interests include VLSI physical design and design-technology co-optimization.