A Self-Supervised and Cross-Design Netlist Power Model for Time-Based Layout Power Analysis

Wenkai Li, Yao Lu, Wenji Fang, Yugao Zhu, Ziyan Guo, Jing Wang,
Mengming Li, Qijun Zhang, and Zhiyao Xie, *Member, IEEE*

***Abstract*—Accurate power prediction in VLSI design is crucial for effective power optimization, especially as designs get transformed from gate-level netlist to layout stages. However, traditional accurate power simulation requires time-consuming back-end processing and simulation steps, which significantly impede design optimization. To address this, we propose ATLAS, which can predict the ultimate time-based layout power for any new design in the gate-level netlist. In addition, we extend ATLAS to support power prediction using only RTL-stage toggle information, further increasing its applicability and efficiency. To the best of our knowledge, ATLAS is the first work that supports both time-based power simulation and general cross-design power modeling. It achieves such general time-based power modeling by proposing a new pre-training and fine-tuning paradigm customized for circuit power. Targeting golden per-cycle layout power from commercial tools, our ATLAS achieves the average mean absolute percentage error (MAPE) of only 5.41%, 3.79%, and 7.51% for the clock tree, register, and combinational power groups, respectively, without any layout information. Overall, the average MAPE for the total power of the entire design is 3.05%, and the inference speed of a workload is significantly faster than the standard flow of commercial tools. Furthermore, ATLAS can bypass the time-consuming signal propagation process, and when using only RTL-stage toggle information, achieves a total power MAPE as low as 5.00%.**

***Index Terms*—Power modeling, netlist, agile design method, self-supervised learning.**

I. Introduction

POWER is an increasingly important objective in modern chip design. As design complexity keeps scaling up, it is increasingly costly for chip designers to get accurate power values of their design. It is even more challenging to simulate time-based (e.g., per-cycle) power values, which enables the analysis of peak power and power fluctuations (Ldi/dt). As Fig. 1 shows, starting at a gate-level netlist, designers need to complete the design whole layout process and then employ target workloads (in .fsdb or .vcd format) to simulate time-based power consumption based on commercial EDA tools [1]. Both layout and power simulation can take days or even weeks for large designs or workloads. In summary, accurate, efficient, time-based power models are in high demand.

This work is supported by Hong Kong Research Grants Council (RGC) YCRG Grant C6003-24Y and GRF Grant 16216825. It was partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government. (Corresponding author: Zhiyao Xie)

Wenkai Li, Yao Lu, Wenji Fang, Yugao Zhu, Ziyan Guo, Jing Wang, Mengming Li, Qijun Zhang, and Zhiyao Xie are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong. Email: (wlidm, yludf, wfang838, yzhuel, zguoby, jwangjw, mengming.li, qzhangcs)@connect.ust.hk; eezhiyao@ust.hk. (Wenkai Li and Yao Lu contributed equally to this work.)

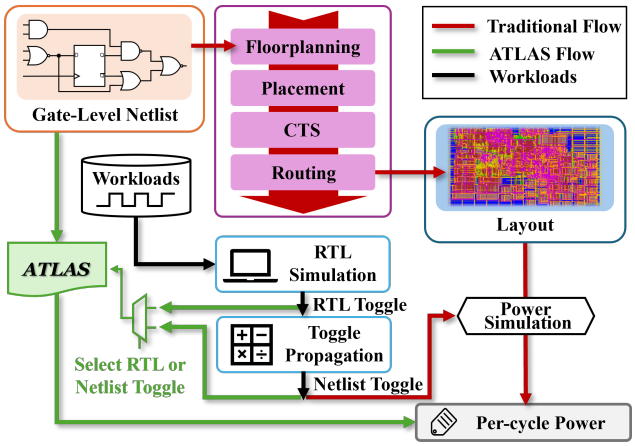

Fig. 1: Overview of ATLAS for time-based netlist power modeling. Standard traditional power simulation for post-layout design is time-consuming due to both layout steps and per-cycle power simulation. ATLAS achieves significant acceleration over the standard flow.

A. Limitation of Existing Power Modeling Works

In recent years, various novel data-driven power modeling techniques [2]–[8] have been proposed, providing unprecedented early-stage and fast power evaluations based on machine learning (ML) power models. Representative power modeling works have been summarized in Table I. However, no prior works can provide *time-based power* and generalize across *different designs* simultaneously. We categorize existing power modeling solutions into two main types:

Time-based but not Cross-design: An important type of power modeling works is time-based but not cross-design. As Table I shows, representative works include PRIMAL [2] and APOLLO [3]. PRIMAL [2] predicts per-cycle power by processing transitions of all registers with multiple ML methods, including both the convolutional neural network and principle component analysis, plus a linear model. APOLLO [3] predicts per-cycle power by selecting the most power-correlated signals. However, they [2], [3] require training a new design-

TABLE I: Summary of representative ML-based power models.

Power Models	Applied Stage	Support Workloads	Time-Based	Cross-Design	Target Layout
PRIMAL [DAC'20] [2] APOLLO [MICRO'21] [3]	RTL	Yes	Yes	No	No
[ICCAD'22] [4] SNS [ISCA'22] [5] SNS V2 [MICRO'23] [6]	RTL	No	No	Yes	No
MasterRTL [ICCAD'23] [7]	RTL	Yes	No	Yes	Yes
PowPredicCT [DAC'24] [8]	Layout	No	No	Yes	Yes
ATLAS	Netlist	**Yes**	**Yes**	**Yes**	**Yes**

*GRANNITE [9] and GRIPT [10] estimate toggle rate instead of power, thus not in the table; they are neither time-based nor targeting the layout.

specific power model from scratch for each new design. This development process, especially the label collection step, is highly time-consuming.

Cross-design but not Time-based: The second type of works [4]–[8] can provide a general power model that applies to new designs, which are unknown during model training. Perhaps due to the difficulty of generalization, none of these works further provides time-based power values. Moreover, most of these works [4]–[6], [8] do not model power based on different workloads. Instead, they only model the average power value based on the propagation of user-defined toggle rates (i.e., vectorless power values).

In summary, existing ML-based power modeling works cannot support both time-based and cross-design analysis. In addition, they also still suffer from the **lack of post-layout power modeling**. Most early power models [2]–[6] are not validated with the ultimate power after design layout. For faster data collection, they typically adopt power values simulated at the gate-level netlist stage as labels, skipping the layout process. As a result, they are not validated on ultimate post-layout power, which is heavily affected by many factors such as accurate metal wire capacitance, timing optimization on netlist (e.g., buffer insertion, netlist reconstruction, gate-sizing), the clock tree, etc.

We summarize four challenges we observed in state-of-the-art early-stage power prediction techniques. Notably, the first three challenges are all related to model accuracy, while the last one concerns the speed and efficiency of existing methods: **1) Generalization Across Designs:** Existing power models often require retraining for each new design and lack the ability to generalize across different designs. **2) Cross-Stage Gap:** There is a significant gap between gate-level netlist[1] and post-layout netlist, due to factors such as parasitic parameters and clock tree synthesis. **3) Limited Labeled Data:** Collecting accurate, labeled power data—especially time-based or post-layout labels—is highly time-consuming and costly, making it difficult to train robust models. **4) Speed:** Some existing ML-based power models are still not sufficiently fast, due to the collection of details features (e.g., netlist-stage per-cycle toggles). In Sec. I-C, we will analyze how our proposed ATLAS addresses each of these challenges.

B. Summary of Our Solution ATLAS

In this work, we propose ATLAS, targeting efficient evaluation of the time-based post-layout power of any given gate-level netlist. To the best of our knowledge, ATLAS is the first work that supports both *time-based* power simulation and general *cross-design* power modeling. Moreover, ATLAS targets the most accurate power label from *layout stage* and *realistic designs* (e.g., out-of-order central processing unit (CPU) designs instead of small blocks). Compared with standard simulation flow based on EDA tools, the general ATLAS solution bypasses both the layout process and time-based power simulation, achieving significant speedups.

[1]In this manuscript, if not explicitly described, by default "gate-level netlist" refers to the post-synthesis gate-level netlist before layout (e.g., the output of Synopsys Design Compiler). The post-layout netlist (with timing optimization and clock tree) will be explicitly mentioned.

ATLAS achieves unprecedented general time-based power modeling based on a customized *pre-training* and *fine-tuning* paradigm. It proposes the following novel strategies:

- **Sub-circuit Generation:** ATLAS first splits each design into non-overlapping sub-circuits. ATLAS will evaluate the per-cycle post-layout power of each small sub-circuit.
- **Pre-training:** A general netlist encoder model is pre-trained based on multiple self-supervised learning tasks without power labels. The encoder will encode each sub-module into a general embedding (i.e., vector) with rich design information. Such an informative embedding effectively supports challenging power modeling tasks.
- **Fine-tuning:** Based on the embedding from the pre-trained encoder, we fine-tune different multilayer perception (MLP) models for three power groups: combinational logic, register, and clock tree.

We evaluate ATLAS on different designs, whose scale range from 250K to 1240K cells under realistic workloads. ATLAS achieves high accuracy in per-cycle power modeling, with only 3.05% error percentage on average of total power. ATLAS is up to $400\times$ faster than the traditional commercial flow by bypassing both the layout process and standard time-based power simulation. Furthermore, ATLAS trained on designs with only 400K gates can successfully predict the power of a much larger design with 1.24M gates, with only 4.43% error percentage on average of total power. The results indicate the superior predictive capability of ATLAS, demonstrating its effectiveness in cross-stage, cross-design, and time-based power prediction. Furthermore, ATLAS is extended to bypass the toggle propagation process, achieving an average error percentage of only 5.00% and delivering up to $600\times$ speedup compared to the traditional commercial flow.

C. Analysis of ATLAS

ATLAS primarily answers the following key unsolved questions in the time-based cross-design layout power modeling problem, directly corresponding to the four challenges summarized in Sec. I-A:

- *Q1:* How to maintain high accuracy when the model applies to different unknown designs? (Address **Generalization Across Designs** challenge)
- *Q2:* How can we capture the obvious gaps (such as clock tree synthesis, buffer insertion, gate sizing, and actual parasitic parameters) between the gate-level and the post-layout netlists? (Address **Cross-Stage Gap** challenge)
- *Q3:* Given the difficulty in generating circuit power labels, how can we achieve high accuracy with limited labeled data? (Address **Limited Labeled Data** challenge)
- *Q4:* Can we make the netlist-stage power model even faster? (Address **Speed** challenge)

For *Q1*, ATLAS achieves high accuracy and generalization across different designs primarily through its self-supervised pre-training strategy:

Pre-training: ATLAS self-supervisely pre-trains its graph-transformer circuit encoder using three tasks: masked toggle propagation, masked node-type prediction, and sub-circuit size prediction. So the encoder learns rich structural and

functional representations of sub-circuits without labels. The masked-toggle and node-type tasks enforce understanding of switching activity propagation and cell types, while size prediction captures global circuit characteristics. By learning these intrinsic properties and behaviors, the encoder is better prepared to model and generalize across different designs.

For *Q2*, ATLAS adopts cross-stage alignment strategies to capture the gaps of netlists before and after layout:

Alignment: To explicitly bridge the gap between gate-level and post-layout netlists, ATLAS incorporates a cross-stage alignment contrastive learning task [11]. Following [11], the encoder is trained to produce similar embeddings for functionally equivalent sub-circuits in both the gate-level and post-layout representations, while pushing apart embeddings of functionally different circuits. This contrastive alignment enables the model to encode layout-specific information—such as buffer insertion, gate sizing, and routing effects—directly into the netlist representation. As a result, the encoder learns to "align" netlist features with their corresponding layout-induced changes, effectively narrowing the gap between the two design stages.

For *Q3*, to achieve high accuracy with limited labeled circuit power data, ATLAS leverages innovations in partitioning, power group modeling, and self-supervised learning:

❶ **Partitioning:** ATLAS partitions each netlist into a set of non-overlapping sub-circuits using hypergraph partitioning (e.g., hMetis [12]), while supporting the alignment between gate-level/post-layout netlist partitions. This approach directly enlarges the dataset and ensures that sub-circuit size remains consistent across designs of varying scale, significantly improving generalization to unseen or larger circuits. In comparison, previous methods mostly split a design into logic cones [5]–[7], resulting in a significant overlap between cones. Consequently, summing the power of these cones does not yield an accurate estimate of the total power.

❷ **Power Group Modeling:** Instead of predicting total power as a single target, ATLAS decomposes power into three groups—clock tree, combinational logic, and register power—aligning with industry-standard practices. ATLAS's fine-tuning is performed separately for these three power groups. This granularity allows the model to capture the distinct physical and functional characteristics of each group, address their unique prediction challenges, and isolate error sources. In addition, ATLAS also supports power prediction for memory group[2]

❸ **Self-supervised Learning:** ATLAS pre-trains its graph encoder on large unlabeled netlist datasets with self-supervised tasks, enabling it to learn useful structural and functional representations without power labels. These pre-trained features greatly improve downstream power modeling accuracy, even with limited labeled data.

For Q4, we further extend ATLAS to use only RTL toggle information as input, referred to as ATLAS_RTL. Unlike GRANNITE [9] and GRIPT [10], which merely predict toggle rates and still rely on commercial EDA tools to compute averaged total power without cross-stage prediction ability, ATLAS_RTL directly targets cross-stage per-cycle power prediction. In the standard simulation flow, the workload is first processed through RTL simulation to generate RTL toggles. Then the toggle propagation generates gate-level toggles on the netlist. We will discuss the time-consuming nature of toggle propagation and our method in detail in Sec. VII. Given that we have already divided total power into three groups, and toggle propagation only impacts the combinational logic group, we only need to approximate the toggle information for this group. To achieve this, we adopt a simple yet efficient method that leverages vectorless power analysis to approximate the per-cycle netlist toggle, which fully considers both the structure and functionality of the combinational logic as well as the per-cycle toggle information of its upstream registers. This approach allows ATLAS to bypass the time-consuming toggle propagation process while incurring a 1.95% increase in total power MAPE.

[2]We exclude memory power from total results because it is highly predictable; including it would artificially raise overall accuracy and obscure the evaluation of the ATLAS (see Sec. VIII-C). As memories are typically hard IP, we estimate their power at the gate-level netlist stage using a lookup-table approach that delivers very high accuracy with minimal effort.

II. Problem Formulation

We formulate our target problem below. Given a post-synthesized gate-level netlist N_g, the objective is to predict the per-cycle power consumption after placement and routing under real workloads. The model should capture the netlist transformation during layout. The model f will predict the power of the post-layout design:

$$f(N_g, T_{\text{Netlist}}, t) \;\rightarrow\; P_{total}(N_p, t) \tag{1}$$

where $P_{total}(t)$ is the per-cycle total post-layout power *label* at the cycle t. N_g is the gate-level netlist. T_{Netlist} is the toggle information at the netlist stage. N_p is the ultimate netlist after layout. In this work, we provide a faster version of ATLAS by supporting directly using RTL-stage toggles T_{RTL} as the model input, as formulated below:

$$f(N_g, T_{\text{RTL}}, t) \;\rightarrow\; P_{total}(N_p, t) \tag{2}$$

The power model f should satisfy: ❶ **Time-based prediction:** For every cycle t, f must predict per-cycle total power $P_{total}(t)$; ❷ **Cross-stage learning:** f should capture the transformation from gate-level netlist to post-layout netlist; ❸ **Cross-design generalization:** f should generalize across designs unseen during training.

III. Methodology Overview

This section provides an overview of the standard ATLAS, which uses netlist toggles T_{netlist} at the netlist stage as its input. As Fig. 2 shows, ATLAS includes three major steps: netlist preprocessing (Sec. IV and Fig. 3), pre-training (Sec. V and Fig. 4), and fine-tuning (Sec. VI and Fig. 5). Finally, we will further extend ATLAS to only use RTL toggles T_{RTL} to predict power in Sec. VII.

Preprocessing (Sec. IV). The flow begins with netlist preprocessing to prepare larger and more diverse data for ATLAS pre-training by splitting each gate-level netlist into many non-overlapping sub-circuits, which increases sample diversity for data-driven power modeling.

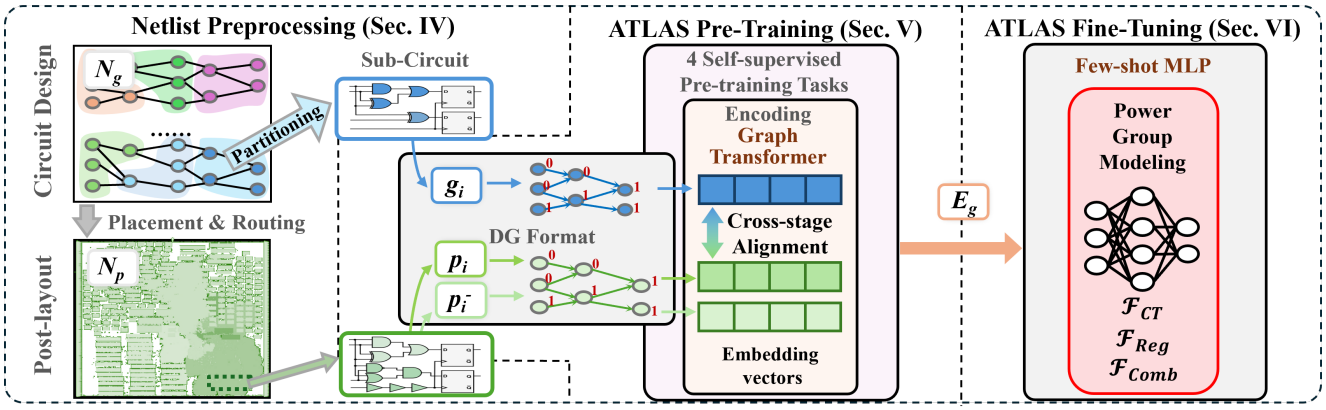

Fig. 2: ATLAS Overview. ATLAS includes three major steps: netlist preprocessing (Sec. IV), pre-training (Sec. V), and fine-tuning (Sec. VI).

ATLAS proposes to partition the gate-level netlist N_g into many non-overlapping sub-circuits g_i with roughly a pre-determined size. ATLAS also partitions the post-layout netlist into the same number of sub-circuits p_i, and ensures alignment between each pair of sub-circuits $\{g_i, p_i\}$. The detailed matching rules are described in Sec. IV. Our objective is for ATLAS to learn how each sub-circuit g_i evolves into its corresponding p_i across different design stages. Then we transform each sub-circuit into the graph format and annotate related features (e.g., cell type, cell power from the liberty file) to each node (i.e., cell instance). During training, the pre-processed data will be used to pre-train the encoder. During inference, ATLAS will encode each sub-circuit and predict its per-cycle power value.

Pre-training (Sec. V). The pre-training stage develops a self-supervised circuit encoder that maps sub-circuits into high-dimensional embeddings. By employing four customized tasks, the encoder learns inherent circuit semantics and the alignment between gate-level and post-layout netlists without power labels. Consequently, it captures layout-induced transformations—such as buffer insertion and clock tree synthesis—directly within the resulting vectors.

In comparison with previous self-supervised learning tasks on netlists [11], [13] that only target small combinational circuits, we target realistic sequential CPU designs. Moreover, we incorporate per-cycle toggle information into one of the four self-supervised learning tasks, allowing ATLAS to learn about toggle propagation.

Fine-tuning (Sec. VI).Finally, ATLAS fine-tunes lightweight MLPs for three categories: combinational, sequential, and clock networks. Using pre-trained embeddings as input, these models predict fine-grained per-cycle power for each sub-circuit group. The total power per cycle is then derived by aggregating results across all groups and sub-circuits.

ATLAS with RTL Toggles (Sec. VII). We extend ATLAS to infer power directly from RTL toggle data, avoiding slow commercial EDA toggle propagation. By approximating combinational toggles from per-cycle register toggles and netlist structural information, ATLAS enables fast end-to-end sub-circuit power estimation without toggle propagation and with only minor accuracy loss.

Clock Gating Implementation: It is worth noting that ATLAS supports clock gating in the target designs. In our flow, clock gating is automatically implemented at the gate-level netlist by the synthesis tool (e.g., Design Compiler), which extracts register enable conditions from RTL, clusters registers with the same enable, and inserts latches to control their clocks.

IV. Step One: Netlist Preprocessing

This section introduces the first step of ATLAS, netlist preprocessing, which generates the dataset in the desired format. This involves generating netlists from different stages and collecting key features that facilitate effective learning.

A. Netlist Collection for Self-supervised Learning

To support the learning of the general intrinsic netlist information across stages, we will collect different netlists during preprocessing. The collected netlists will be applied in self-supervised encoder pre-training in the next step. As shown in Fig. 3, we will generate two types of netlist for model pre-training: N_g, N_p. The N_g is the original gate-level netlist by synthesizing RTL code. After performing the complete layout process, we obtained the post-routing design layout, and we refer to the corresponding post-layout netlist as N_p. During encoder pre-training, as we will introduce in Sec. V, the objective is to learn the cross-stage alignment between N_g and N_p, making the embedding of the gate-level netlist E_g to capture and reflect the information of post-layout netlist E_p.

B. Sub-circuit Generation

Fig. 3 illustrates the netlist preprocessing process. Given a gate-level netlist N_g, we split it into a set of non-overlapping sub-circuits $N_g \in \{g_1, g_2, ..., g_i\}$. Then this original netlist N_g will be transformed to a corresponding post-layout netlist N_p, which will similarly yield a set of sub-circuits $N_p \in \{p_1, p_2, ..., p_i\}$. During this standard layout process, the netlist will be optimized for better timing through buffer insertion, netlist reconstruction, etc. Also, the clock tree will be synthesized and added to the design.

There are two mainstream partitioning approaches for quality-prediction tasks: cone-based and hierarchy-based. ❶ Cone-based methods (e.g., [5]–[7]) split the design into logic cones. Each subcircuit is a flip-flop endpoint plus all driving logic, leading to overlaps so that summing cone powers overestimate total design power. ❷ Hierarchy-based methods follow the netlist hierarchy but produce subcircuits with highly variable sizes; we observe that a model trained on 400K-cell designs performs poorly on a 1,200K-cell design.

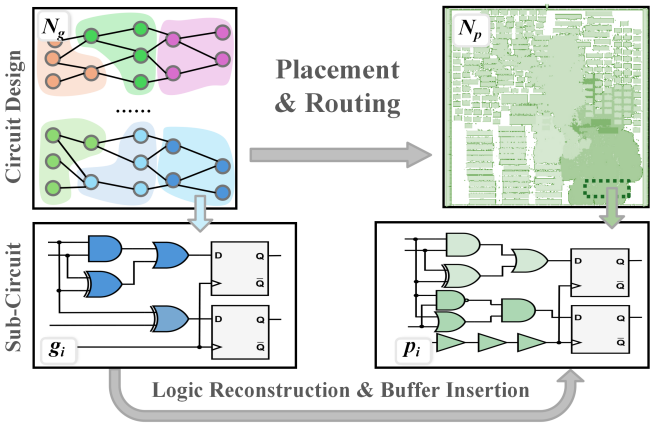

Fig. 3: Netlist Preprocessing of ATLAS. The preprocess includes circuit segmentation in both gate-level netlist N_g and post-layout netlist N_p. For each sub-circuit g_i in N_g, there exists a transformed expression p_i after layout.

In ATLAS, we apply the hypergraph partitioning (e.g., hMetis [12]) to the entire gate-level netlist N_g to generate many partitions, each being one sub-circuit. We keep the size of all sub-circuits (i.e., partitions) roughly the same. The size of each partition is a hyperparameter. The ATLAS model will ultimately predict the power of each sub-circuit (i.e., partition).

When building the training dataset, we match each gate-level sub-circuit to its corresponding layout-level sub-circuit. Since the vast majority of cell names remain unchanged from the gate level to the layout stage (over 80% in our design), we can identify most layout-level sub-circuit cells by name matching. For newly generated cells (e.g., from netlist restructuring, buffer insertion, or clock-tree synthesis) we iteratively assign each unassigned cell based on its neighbors: if all neighbors belong to the same sub-circuit, assign the cell there; if neighbors span multiple sub-circuits, assign it randomly. Repeat until all cells are assigned to produce the final partitioning. This approach offers obvious advantages:

Generalization capability: By partitioning the netlist into sub-circuits of equal size, the sub-circuit size remains consistent regardless of the overall design size. This enables models trained on small designs to accurately predict the power of a larger design. The experimental section further explores different partitioning methods to confirm this robust generalization.

Non-overlapping: Works [5]–[7] based on logic cones unavoidably involve significant overlapping between different cones, making them actually inappropriate for power modeling. Significant logic overlap leads to power overestimation when summing individual logic cones. In contrast, our approach employs non-overlapping sub-circuits. This ensures the total design power can be accurately estimated by aggregating the power of all constituent sub-circuits.

C. Feature Collection

We split all netlists from Sec. IV-A into sub-circuits. Then we convert each sub-circuit to a directed graph (DG). Each node corresponds to each cell instance and directed edges are the wires connecting these nodes. By representing sub-circuits as DGs, we will leverage the advantages of graph transformer models to capture the circuit structures and semantics.

Then we proceed to collect features for every graph node. To benefit the subsequent learning process, we have carefully selected four types of features: Node Type, Per-cycle Toggle, Cell Internal, and Leakage Power.

1) Node Type: We categorize all cells according to their functionality into 18 types, using one-hot encoding to represent the cell type. We identified the 17[3] most common node types in our dataset: `DF`, `AO`, `ND`, `OA`, `MU`, `CK`, `NR`, `IN`, `XN`, `BU`, `IO`, `FA`, `MO`, `XO`, `AN`, `MA`, and `OR`. All other less common node types are grouped into an 18$^{\text{th}}$ type labeled as `Other`. ATLAS will learn to recognize the node type and how the node type effectively conveys each cell's functional characteristics.

2) Per-cycle Toggle: Per-cycle toggle represents switching activity at a node's output port during a specific timestamp under real workloads. Including a per-cycle toggle aims for ATLAS to learn the signal propagation relation between nodes and to predict signal propagation. During the pre-training, the encoder will learn by predicting the masked toggle behavior.

3) Cell Internal and Leakage Power: For each cell instance, both internal and leakage power values will be parsed from the lookup tables in the `.lib` files from the technology library according to its cell type. These standard cell power values provide important cell information and are directly power-related.

V. Step Two: ATLAS Pre-training

ATLAS trains a general circuit encoder to convert each sub-circuit into a representation. By handling each sub-circuit independently, ATLAS can generate a diverse and comprehensive set of training samples, which is crucial for model generalization. Fig. 4 illustrates the entire self-supervised pre-training process of the encoder model based on efficient graph transformer [15]. The encoder will generate an overall graph embedding for the whole input graph. By performing contrastive learning on each sub-circuit independently, the encoder can fully learn the transformations that occur in sub-circuits across different stages.

The encoder is pre-trained on the unlabeled circuit dataset, targeting the following two general learning goals: 1) learning the toggle propagation of netlists; 2) learning the alignment between the netlist and layout. Our proposed four self-supervised learning tasks can be roughly categorized into three types, as summarized below.

1) Learning toggle propagation based on masked node recovery (Tasks ❶, ❷). We will mask (i.e., hide) important node properties (e.g., type, toggle) of randomly selected nodes. A temporary MLP-based classification head will predict the masked node properties based on the encoder-generated node embedding. During pre-training, the encoder and classification head (to be discarded after pre-training) will be trained end-to-end to maximize accuracy. The encoder will thus learn to encode the structural information of connected circuit operators within the

[3] `DF` is flip-flops; `XN` is Exclusive NOR; `XO` is Exclusive OR; `AN` is AND-type; `FA` is 1-bit full/half adder; `AO` is AND-OR(-Inverter); `ND` is NAND; `OA` is OR-AND(-Inverter); `MU` is multiplexer; `CK` is clock-related cells; `NR` is NOR; `IN` is NAND/NOR with 1 inverted input; `BU` is buffer; `IO` is inverter-OR-AND; `MO` is MOAI22Dx; `MA` is MAOI22Dx; `OR` is OR gate. Details of the 17 types can be found in the official TSMC 40nm standard cell library [14].

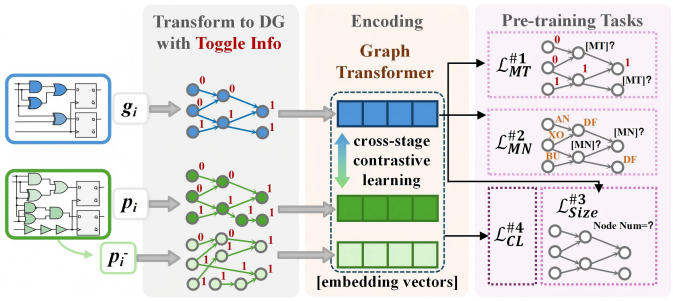

Fig. 4: ATLAS Pre-training. ATLAS employs a self-supervised and cross-stage flow that incorporates one type of contrastive learning. Additionally, ATLAS includes predictions for [MASK_TOGGLE] ([MT]), sub-circuit size, and [MASK_NODE_TYPE] ([MN]).

circuit graph. Since whether a node toggles depends on its upstream nodes and the logic structure, masked recovery tasks can help the encoder simulate toggle propagation and learn this behavior.

2) Size recognition (Task ❸). A temporary MLP-based regression head will predict the total number of nodes in the given graph (i.e., sub-circuit netlist) based on encoder-generated graph embedding. During pre-training, the encoder and regression head (to be discarded after pre-training) will be trained end-to-end to maximize accuracy. The encoder will learn to encode more global graph information in the overall graph embedding.
3) Netlist contrastive learning (Tasks ❹). The encoder is trained to minimize the embedding distance in the latent space between matched sub-circuits, such as gate-level sub-circuits and their corresponding post-layout sub-circuits (positive samples). At the same time, the encoder maximizes the embedding distance in the latent space between different sub-circuits (negative samples).

We formally introduce and formulate each learning task below.

Task #1 Masked toggle propagation learning: To train our encoder model to handle per-cycle workloads, we mask the per-cycle toggle information of randomly selected nodes. This pre-training task trains the encoder to capture the toggle information and the propagation of toggles. Specifically, we randomly mask the toggle feature (0 or 1) of selected nodes and replace it with a special token [MASK_TOGGLE]. Subsequently, we predict whether these nodes will toggle based on the embedding of the masked node, which also captures neighboring nodes' information. We represent the input circuit in masked graph format as $\hat{G}$. The ground-truth toggle feature of the masked nodes is represented by $\boldsymbol{t}_{\hat{G}}^{msk}$. The predicted toggle feature for the masked nodes is represented by $\boldsymbol{p}^{msk}(\hat{G})$, which is based on node embeddings[4]. The objective is to minimize the cross-entropy (CE) loss between $\boldsymbol{t}_{\hat{G}}^{msk}$ and $\boldsymbol{p}^{msk}(\hat{G})$, as expressed in the following formula:

$$\mathcal{L}_{\text{MT}}^{\#1} = \mathbb{E}_{\hat{G}\sim\mathcal{D}} CE\left(\boldsymbol{t}_{\hat{G}}^{msk},\ \boldsymbol{p}^{msk}(\hat{G})\right) \tag{3}$$

where $\mathbb{E}_{\hat{G}\sim\mathcal{D}}$ represents the expectation $\mathbb{E}$ over the circuit graph dataset $\mathcal{D}$.

[4]To simplify the formulation, the encoder model does not directly appear in the loss term in Eq. (3). The prediction $\boldsymbol{p}^{msk}(\hat{G})$ is based on encoder model.

Task #2 Masked node type learning: Similar to the toggle task, we randomly mask the node type information and replace it with a special token [MASK_NODE_TYPE]. The input circuit's masked representation is denoted as $\hat{G}$, with the ground-truth type for the masked nodes represented by $\boldsymbol{c}_{\hat{G}}^{\text{msk}}$ and the predicted type denoted by $\boldsymbol{q}^{\text{msk}}(\hat{G})$. The objective is to minimize the cross-entropy (CE) loss between the true one-hot encoding $\boldsymbol{c}_{\hat{G}}^{\text{msk}}$ and the predicted one-hot encoding $\boldsymbol{q}^{\text{msk}}(\hat{G})$, expressed as:

$$\mathcal{L}_{\text{MN}}^{\#2} = \mathbb{E}_{(E_g)\sim\mathcal{D}} CE\left(\boldsymbol{c}_{\hat{G}}^{\text{msk}},\ \boldsymbol{q}^{\text{msk}}(\hat{G})\right) \tag{4}$$

Task #3 Sub-circuit size learning: The size of a circuit design is often directly correlated with its overall power consumption. We propose a pre-training task to recognize the size of each sub-circuit. The objective is to minimize the MSE between predicted node number Num_{pre} and the actual number of nodes Num_{true}:

$$\mathcal{L}_{\text{Size}}^{\#3} = \mathbb{E}_{\hat{G}\sim\mathcal{D}}\left[\left(\text{Num}_{pre} - \text{Num}_{true}\right)^2\right] \tag{5}$$

Task #4 Cross-stage alignment contrastive learning: For cross-stage contrastive learning, we aim for the gate-level netlist N_g to be closer in the embedding space to the corresponding post-layout netlist N_p. This alignment task targets capturing layout information from N_p in netlist embedding. For such contrastive learning, the original sample is g_i, while the positive sample is p_i, and the negative samples p_i^- comprise all other functionally distinct sub-circuits in the batch. We denote the embeddings of p_i and p_i^- as E_p and E_p^-, respectively. We also adopt the InfoNCE loss function for this purpose:

$$\mathcal{L}_{\text{CL}}^{\#4} = \mathbb{E}_{(E_g,E_p)\sim\mathcal{D}} CL\left(E_g, E_p, E_p^-\right) \tag{6}$$

Finally, we formulate the complete self-supervised pre-training objective of our model by jointly employing the four learning tasks:

$$\mathcal{L} = \mathcal{L}_{\text{MT}}^{\#1} + \mathcal{L}_{\text{MN}}^{\#2} + \mathcal{L}_{\text{Size}}^{\#3} + \mathcal{L}_{\text{CL}}^{\#4} \tag{7}$$

For ATLAS's encoder, traditional GNNs [16] and Graph Transformers[17][18] are often resource-intensive on large graphs. We instead adopt SGFormer [15], which enables global node propagation with linear complexity, without requiring positional encodings, preprocessing, or extra loss functions.

VI. Step Three: ATLAS Fine-tuning

To apply pre-trained ATLAS on downstream power modeling tasks, as shown in Fig. 5, we further decompose the overall power of each sub-circuit into three power groups: the clock tree power, the combinational logic power, and the register power[5]. Our three fine-tuning models operate on each sub-circuit independently, predicting the power of all three groups for every individual sub-circuit. For the overall circuit design, the power of each group is obtained by summing the corresponding power group values across all sub-circuits. The total power of the circuit design is then calculated as the sum

[5]Our register power group includes (and is dominated by) the power of the clock pin in each register. Accordingly, our clock tree power group does not include such register clock pin power.

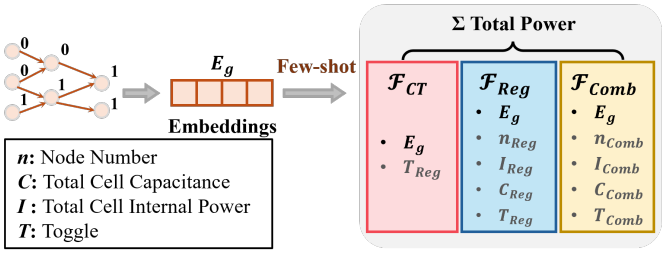

Fig. 5: ATLAS Fine-Tuning. The total post-layout power is divided into different groups as three separate fine-tuning tasks, which are predicted by three fine-tuning models. Other than embedding vectors E_g, ATLAS uses cell internal power, cell capacitance, toggle, and node number as features, avoiding the need for cumbersome feature engineering and complex machine learning models.

of the three power groups. We divide total power into three categories for two reasons: ❶ **Distinct Power Characteristics:** Circuit components differ in power behavior, distribution, and influencing factors. ❷ **Industry Alignment:** Commercial EDA tools typically report clock tree, combinational logic, and register power separately.

We use an MLP to fine-tune each power group with dedicated models $\mathcal{F}_{CT}$, $\mathcal{F}_{Comb}$, and $\mathcal{F}_{Reg}$. MLPs keep the framework fully neural and end-to-end trainable, simplifying integration while improving flexibility, scalability, and maintainability. These fine-tuned models rely on ❶ the encoder embedding E_g instead of extensive feature engineering, and ❷ simple gate-level sub-circuit features (cell count, internal power, capacitance, toggle counts).

Regarding the clock tree group, it is completely absent in the post-synthesis netlist N_g and appears only in the post-layout netlist N_p, which is unknown during inference. The only information that can reflect the toggle behavior of the clock tree is the registers connected to it. Therefore, we rely solely on the embedding E_g and the total toggle information of the register group T_{Reg} to predict the power of the clock tree group. T_{Reg} denotes the total number of output-pin toggles of register groups during every cycle.

For both combinational and register groups, we choose to combine the embeddings E_g with additional relevant features available from the netlist N_g. For the combinational group, we utilize the number of combinational nodes n_{Comb} of the gate-level netlist, along with the overall effective cell internal power I_{Comb}, overall effective cell capacitance C_{Comb} and total toggle information T_{Comb} of the combinational group as features. The cell internal power and cell capacitance of each node are collected from the lookup tables in the `.lib` file of the technology library. The capacitance for each cell is calculated as follows: it is the sum of the output pin capacitance of the cell and the input pin capacitances of all the cells it drives. Both types of capacitance values are obtained from the .lib file. To calculate the total cell internal power I_{Comb} and total cell capacitance C_{Comb}, we multiply each cell internal power and capacitance in the combinational group by the per-cycle toggle of their output pins and sum up the results. The reason for multiplying by the per-cycle toggle rate is to obtain the effective per-cycle capacitance or internal power, which are strongly correlated with the power consumption of the corresponding power group. T_{Comb} denotes the total number of output-pin toggles of the combinational group during every cycle.

As for the register group, similar to the combinational group, we adopt the number of register nodes n_{Reg}, the overall effective cell internal power I_{Reg}, the overall effective cell capacitance C_{Reg} and total toggle information T_{Reg} of the register group from N_g as additional features for fine-tuning. The methods for obtaining n_{Reg}, T_{Reg}, and C_{Reg} is identical to those used for the combinational group. As for I_{Reg}, we multiply the toggle activity of each pin by the cell internal power, and sum them up.

Our MLP fine-tuning models are lightweight. The total predicted power at cycle t is:

$$\mathcal{F}_{CT}(E_g, T_{Reg}, t) + \mathcal{F}_{Reg}(E_g, n_{Reg}, I_{Reg}, C_{Reg}, T_{Reg}, t) + \mathcal{F}_{Comb}(E_g, n_{Comb}, I_{Comb}, C_{Comb}, T_{Comb}, t)$$

VII. Extending the ATLAS Approach: Power Prediction Using RTL Stage Toggles

To further accelerate ATLAS, we extend ATLAS to enable per-cycle power prediction directly using toggle information available at the RTL stage. It removes the computational overhead of toggle propagation by skipping this process, which converts the RTL toggles to the netlist toggles.

A. Reasons for Prediction with RTL Toggles

The propagation of signal toggles from the RTL stage to the netlist stage is inherently a time-consuming process, particularly for time-based power analysis. In the standard simulation flow, the workload is first processed through RTL simulation to generate RTL toggles. Next, toggle propagation is performed to produce gate-level toggles on the gate-level netlist. EDA tools use the toggle of registers from the RTL simulation as the starting point, and propagate the toggle activity through the combinational network. This propagation process will simulate the switching activity of individual combinational gates.

At the RTL stages, most *registers* can already be inferred based on sequential RTL signals. To bypass the toggle propagation process, we need to approximate the toggle information for the *combinational logic group*. As described in Sec. VI, we have divided the total power into three groups: clock tree, register, and combinational logic. The main features used for predicting the power of the clock tree and register groups are both related to register toggle activity, which is already available at the RTL stage.

B. Approximate Method

When only RTL toggles are available, power modeling is challenged by the structural gap to the netlist stage: RTL lacks explicit combinational logic. However, we observe that there is another type of power analysis named vectorless power analysis, which is not the target of this work, but can help solve the challenge. Vectorless analysis can directly provide a power evaluation when testbenches are not available yet, usually at the early phase of a design project. Leveraging such vectorless power analysis, we can efficiently extract features about combinational logic, without the need for workload-driven netlist toggles. Such features will be a good approximation of the

real per-cycle toggle rate, since they are also heavily power-related. Notably, we only utilize vectorless analysis to help get features for the combinational logic. **Our final power predictions remain per-cycle and workload-aware.**

Unlike vector-based power analysis, which simulates real workloads cycle by cycle, vectorless analysis assigns probabilistic toggle rates (e.g., 40%) to inputs and registers based on tool or user assumptions. These probabilities are then propagated through the combinational logic according to circuit structure and Boolean functionality, independent of workload data. Specifically, using a commercial timing and power analysis tool (e.g., PTPX®), we first annotate each register with the same vectorless toggle rate $T_{Reg}^{Vectorless}$, the tool will generate a toggle rate $T_{Comb,i}^{Vectorless}$ of every combinational logic i, which reflects the expected switching activity under these probabilistic assumptions. This generated vectorless toggle rate is a static estimation, derived from the assumed switching probabilities and logic propagation. To approximate our target workload-dependent toggling behavior, we combine the vectorless estimation with the actual per-cycle toggle of its upstream registers, which are available in RTL toggles. Denote T_{Reg_sum} as the sum of per-cycle toggles of all upstream registers. We compute the per-cycle toggle count for combinational logic gate i as:

$$T_{Comb,i} = \frac{T_{Comb,i}^{Vectorless} * T_{Reg\_sum}}{T_{Reg}^{Vectorless}} \tag{8}$$

where $T_{Comb,i}$ reflects the estimated per-cycle switching activity for gate i in the combinational logic group, bridging vectorless and vector-based power modeling.

C. ATLAS Flow with RTL Toggles

With this new additional toggle approximation, the remaining ATLAS flow remains largely the same for such extension to RTL toggles. We adopt the embedding E_{g_RTL}, the number of combinational nodes n_{Comb}, the overall effective cell internal power I_{Comb}, the overall effective cell capacitance C_{Comb_RTL} and C_{Reg}, and the total toggle information T_{Comb_RTL} as fine-tuning features. The methods for obtaining n_{Comb} and C_{Reg} are identical to those described in Sec. VI. T_{Comb_RTL} is the sum of approximated $T_{Comb,i}$ in one sub-circuit; I_{Comb_RTL} and C_{Comb_RTL} are the sum of each cell internal power and output pin capacitance in one sub-circuit multiplied by the approximated $T_{Comb,i}$, respectively.

The revised fine-tuning model for power prediction in the combinational logic group, using only RTL-stage information, is as follows:

$$\begin{aligned}\mathcal{F}_{Comb\_RTL}(&E_{g\_RTL}, n_{Comb}, I_{Comb\_RTL},\\ &C_{Comb\_RTL}, T_{Comb\_RTL}, C_{Reg}, t)\end{aligned} \tag{9}$$

Here we denote the embedding based on RTL toggles with our approximation as E_{g_RTL}, to distinguish from E_g with netlist toggles. Since the embedding changes from E_g to E_{g_RTL}, the E_g used in the fine-tuning models for register and clock tree group also becomes E_{g_RTL}. As shown in Experimental Result (Sec. IX-B), this change has minimal impact on the prediction accuracy for the register and clock tree groups.

VIII. Experimental Implementation

In this section, we introduce the setup of our experiments and our implementations in detail. Our experiment results will be given in the next section (Sec. IX.)

A. ATLAS Implementation and Experiment Data Generation

We implement our ATLAS using PyTorch and PyTorch Geometric (PyG). Our pre-training is conducted on a Linux machine equipped with an NVIDIA A6000 GPU, while the fine-tuning tasks are performed on an Intel Xeon processor with 128GB of memory.

RTL simulation on workloads is based on Synopsys VCS®[19]. The logic synthesis is executed at a clock frequency of 1GHz using Synopsys Design Compiler®[20] (DC). In our experiments, we utilized the TSMC 40nm standard cell library [14], along with the corresponding Memory Compiler. We utilized the Innovus® to perform mixed-size placement, clock tree synthesis, and routing, with each step including timing optimization, ultimately resulting in the chip design layout. After detailed routing with Innovus®, we dump out the post-layout ultimate netlist and corresponding RC values in Standard Parasitic Extraction Format (SPEF) files. Per-cycle toggle propagation and power simulation are performed based on PrimeTime PX® [1] (PTPX®). We ran PTPX® at both the gate-level and post-layout stages, using the former as the baseline and the latter as the golden label. Accurate post-layout power simulation was conducted based on the post-layout netlist, SPEF files, and diverse workloads; while the gate-level baseline relied on the gate-level netlist and workloads.

Our dataset consists of 12 realistic designs (named C1 to C12) that support workload simulation. Our dataset is derived from the `Boom` series CPUs in Chipyard [21], which were generated by extensively modifying various CPU parameters. Table II presents the cell count statistics for twelve different designs at both the gate-level and post-layout stages. The cell counts range from approximately 250,000 to 1240,000, indicating their difference. For the same design, the increase in cell count from the gate-level stage to the post-layout stage reflects the impact of timing optimization and clock tree synthesis during the layout process. During netlist preprocessing, we use hMetis [12] to partition the gate-level netlist into sub-circuits, each containing approximately 200 gates.

We use two experimental setups. **Setting 1:** In Sec. IX, C2, C4, C7, C9, and C11 are for training; the rest are for testing. This gives a balanced train/test split for standard model evaluation. **Setting 2:** In Sec. X, C2, C3, and C4 are used for training, with all other designs for testing. Here, the training set is much smaller, and some test cases (like C12) are much larger than any training case. This tests ATLAS's

TABLE II: The statistics of the gate counts for the twelve designs at the gate-level and post-layout stages.

	C1	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*	*C9*	*C10*	*C11*	*C12*
Gate-level	242012	267975	321116	399207	513305	570678	668339	838823	797875	1012021	1079086	1120456
Post-layout	257969	285917	345872	418708	547915	605131	713740	929767	851145	1144901	1206049	1244124

generalization and shows the advantage of our partitioning approach. For encoder pre-training, all four self-supervised tasks run jointly (41 epochs/29h in Setting 1, 33 epochs/9h in Setting 2). The encoder is trained once and reused across designs. We use ReLU activation, batch size 256, and learning rate 1e-4 with Adam. For fine-tuning, three MLPs (one per power group) are trained: 4 hidden layers for registers, 3 for combinational and clock tree groups. Each group trains for less than 30 minutes with a learning rate 1e-4.

B. Power Modeling Experiment Setup

Commercial Tool as Baseline. We will evaluate the modeling accuracy and efficiency of per-cycle post-layout power based on post-synthesis netlists (without any layout information). The tested design is strictly not *seen* by the power model during training. As summarized in the introduction, prior works either are only design-specific models that do not apply to new designs [2], [3], or can only evaluate the average power [4]–[8]. No prior ML-based power models apply or can be easily extended to this challenging cross-design per-cycle power modeling task. Therefore, we compare ATLAS with the standard commercial tool at the gate-level netlist stage.

We evaluate the accuracy of per-cycle power modeling with Mean Absolute Percentage Error (MAPE) between the i^{th} per-cycle power label Y_i and prediction $\hat{Y}_i$, assuming altogether m cycles in the tested workload.

$$\text{MAPE} = \frac{1}{m}\sum_{i=1}^{m}\left|\frac{Y_i - \hat{Y}_i}{Y_i}\right| \times 100\% \tag{10}$$

C. Exclusion of Memory Group from Results.

In our power analysis, memory (SRAM) makes up almost half of the total power. Still, we can predict memory power very accurately and easily at the netlist stage. This is because SRAM macros are standardized, well-packaged IP blocks—unlike standard cells or clock trees, their layout and circuits hardly change during physical design. So, SRAM power depends mostly on its operational state and access patterns, not on layout changes. Additionally, we found that clock gating does not affect the SRAM clock ports in our designs; they remain directly connected to the chip's clock input. This means the toggle behavior of SRAM remains consistent from RTL to netlist.

SRAM power mainly comes from port switching (such as read/write address lines, data lines) during each cycle. Each toggle incurs dynamic power consumption, and these values are listed in the lookup table in *.lib* technology file. So, we just need to count port toggles during workload simulation and use .lib energy parameters to look up and calculate total energy for each operation (read, write, etc.). To improve flexibility, we use a simple Ridge Regression model, taking toggle counts as input and outputting total power consumption. This is efficient and doesn't need heavy power simulation tools with only 0.75% error. Since memory power prediction is already highly accurate, including it in ATLAS would reduce overall error but make the result dominated by memory. To better show ATLAS's strength, we exclude this easier memory group and focus on the more challenging combinational logic, clock tree, and register power components.

IX. Experimental Results

This section presents the experimental results of the ATLAS framework. We compare ATLAS's power prediction accuracy to commercial tools across different designs, using either netlist or RTL toggle as inputs. We also demonstrate the power envelope checking capability of ATLAS.

A. Power Prediction Results with Netlist Toggle

We use C2, C4, C7, C9, and C11 as training datasets, and others as testing datasets. In Table III, we provide a comprehensive comparison of results for ATLAS and Gate-Level PTPX®, which displays the MAPE for different power groups under different workloads. Gate-Level PTPX® exhibits the $> 56\%$ prediction error in total power in all cases, while ATLAS shows $\leq 4.4\%$ error. **It is worth noting that** our comparison between ATLAS and Gate-Level PTPX® is to highlight that commercial tools' power estimates at the gate-level netlist stage do not accurately reflect the power consumption at the post-layout stage. This underscores the necessity for novel power models such as ATLAS to bridge the gap in power estimation across different design stages.

For the register group in Table III, Gate-Level PTPX® achieves a low MAPE of 5.06% since the netlist structure remains unchanged from pre- to post-layout. Although the structure remains unchanged, factors such as gate sizing and actual routing information, both of which aren't captured by Gate-Level PTPX®. ATLAS, on the other hand, can learn these details through its carefully designed pretraining phase, resulting in a lower average MAPE of 3.79%.

For the clock tree group in Table III, the gate-level netlist lacks the clock tree, so Gate-Level PTPX® is inapplicable. ATLAS achieves an average MAPE of 5.41%. When analyzing the entire clock network (clock tree + register), the average

TABLE III: MAPE (%) of seven designs. The second group, "Clock Tree" column, is N/A for all designs in Gate-Level PTPX®.

Testing Design	MAPE Error (%) of ATLAS					MAPE Error (%) of Gate-Level PTPX®				
	Register	Clock Tree	Clock Tree+Register	Combinational	Total	Register	Clock Tree	Clock Tree+Register	Combinational	Total
C1	4.69	7.83	6.33	9.72	4.17	8.03	**N/A**	57.92	71.33	61.36
C3	5.83	11.39	3.50	7.37	3.67	7.50		56.02	72.81	60.61
C5	5.28	6.01	4.80	7.61	4.40	7.80		57.05	73.80	62.21
C6	3.54	3.37	3.03	9.11	3.40	3.57		55.53	73.75	60.37
C8	2.24	1.96	1.63	7.84	2.24	5.42		54.41	73.96	59.59
C10	3.17	6.48	1.82	3.07	1.62	1.24		50.64	73.90	56.54
C12	1.80	0.84	0.95	7.87	1.83	1.84		50.18	75.81	57.06
Avg	**3.79**	**5.41**	**3.15**	**7.51**	**3.05**	**5.06**	**N/A**	**54.54**	**73.62**	**59.68**

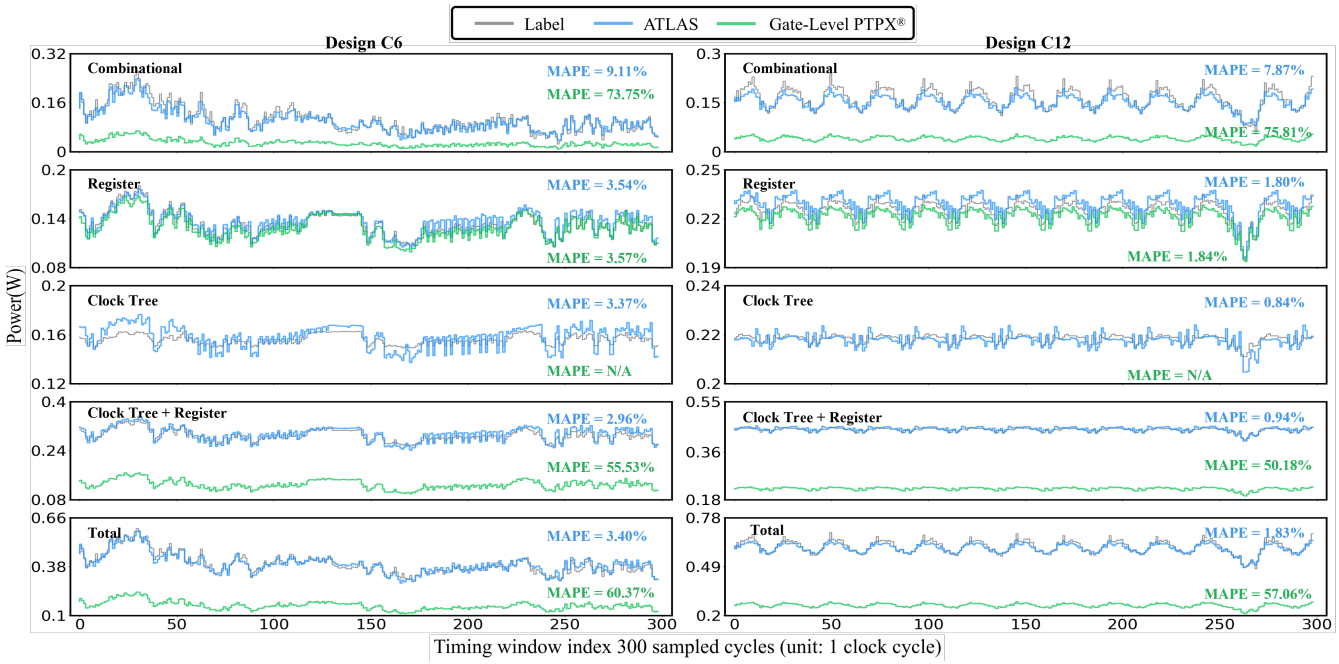

Fig. 6: This figure visualizes the power prediction results of *C6* and *C12* across 300 cycles presented in Table III. The labels were obtained by simulating the netlist and SPEF files using PTPX® after detailed routing with Innovus®. The Gate-Level PTPX® values, which are obtained from the gate-level netlist, illustrate the significant differences in netlist power between gate level and layout.

TABLE IV: Comparison of power prediction using netlist toggle vs. RTL toggle information. Column names represent: Comb. (Combinational logic group); Net. (using netlist toggle information); RTL. (using RTL toggle information).

Testing Design	MAPE error (%) of ATLAS							
	Comb.		Register		Clock Tree		Total Power	
	Net.	RTL.	Net.	RTL.	Net.	RTL.	Net.	RTL.
C1	9.72	14.61	4.69	4.70	7.83	8.00	4.17	5.98
C3	7.37	11.72	5.83	5.83	11.39	11.49	3.67	4.25
C5	7.61	18.92	5.28	5.29	6.01	6.39	4.40	7.51
C6	9.11	17.69	3.54	3.54	3.37	3.59	3.40	5.55
C8	7.84	15.33	2.24	2.24	1.96	2.00	2.24	4.02
C10	3.07	17.56	3.17	3.16	6.48	6.49	1.62	2.96
C12	7.87	18.97	1.80	1.80	0.84	0.87	1.83	4.74
Avg	**7.51**	**16.40**	**3.79**	**3.79**	**5.41**	**5.55**	**3.05**	**5.00**

MAPE of Gate-Level PTPX® rises sharply to 54.54%, whereas ATLAS achieves a much lower MAPE of 3.15%. These results further demonstrate ATLAS's capability to predict cross-stage changes in the entire clock network, including clock tree synthesis, gate sizing, and actual routing effects. In terms of the most challenging combinational group prediction in Table III, ATLAS shows an average MAPE of 7.51%, while Gate-Level PTPX® has a considerably higher MAPE of 73.62%.

Fig. 6 visualizes power prediction results of ATLAS (blue), Gate-Level PTPX® (green), and post-layout golden labels (grey) over 300 cycles for designs *C6* and *C12* in Table III. For both designs, ATLAS closely matches the labels with total power MAPE of 3.40% and 1.83%, while Gate-Level PTPX® exceeds 57% error and deviates significantly.

B. Power Prediction Results Directly with RTL Toggle

As described in Sec. VII, we use RTL toggle as a faster solution to estimate the power. The results, summarized in Table IV, compare the power prediction accuracy when using netlist toggle information versus RTL toggle information for three power groups and the total power.

TABLE V: Kendall Correlation and ROC-AUC for different designs.

Design	Kendall Correlation (τ)				ROC-AUC			
	ATLAS-Netlist		ATLAS-RTL		ATLAS-Netlist		ATLAS-RTL	
	Total	Comb	Total	Comb	Total	Comb	Total	Comb
C1	0.74	0.80	0.72	0.66	0.98	0.99	0.96	0.97
C3	0.82	0.85	0.68	0.67	0.99	0.99	0.98	0.98
C5	0.81	0.86	0.75	0.75	1.00	0.99	0.98	0.98
C6	0.79	0.84	0.73	0.70	1.00	1.00	0.99	0.98
C8	0.68	0.77	0.71	0.61	0.99	1.00	0.99	0.99
C10	0.82	0.85	0.67	0.70	0.99	1.00	0.91	0.94
C12	0.86	0.83	0.74	0.72	1.00	0.99	0.97	0.95
Avg	**0.79**	**0.83**	**0.71**	**0.69**	**0.99**	**1.00**	**0.97**	**0.97**

Table IV shows that using RTL toggle data raises the average MAPE for total power by only 1.95 percentage points (from 3.05% to 5.00%), so RTL-based estimates are still quite accurate overall. For the combinational group, MAPE increases moderately (16.40% vs. 7.51%) compared to netlist toggles, as expected due to the less detailed toggle activity available at the RTL stage. Register and clock tree predictions stay very accurate because register toggle data is already available at RTL. Overall, our method is practical and effective, enabling accurate total power estimation without the slow netlist-level toggle propagation.

In summary, ATLAS-RTL achieves a 16.40% MAPE for combinational logic group power and a 5.00% MAPE for total power, which are acceptable results. In contrast, the Gate-level PTPX® exhibits a much higher error of 73.62% (in Table III) for combinational logic power estimation. Moreover, to the best of our knowledge, there is no prior academic work predicts cross-stage and cross-design per-cycle total power, let alone separate prediction of combinational logic group power.

C. Evaluating Power Envelope Checking Capability

Although ATLAS-RTL exhibits lower accuracy than ATLAS-Netlist in both combinational logic and total power prediction, it remains effective for power envelope checking.

TABLE VI: Comparison of runtime (seconds) for different power estimation flows on various designs. ATLAS-Netlist. and ATLAS-RTL. represent ATLAS using netlist toggle and RTL toggle, respectively. Other names represent: RTL Sim. (RTL simulation time); Tog. Pro. (Toggle propagation time); Pre.+Inf. (netlist preprocessing time plus inference time); Sim. (Power simulation time). Total1 includes both RTL simulation and toggle propagation time, while Total2 only includes RTL simulation time.

Testing Design	RTL Sim.	Tog. Pro.	Traditional Flow			ATLAS-Netlist.		ATLAS-RTL.	
			P&R	Sim.	Total1	Pre. +Inf.	Total1	Pre. +Inf.	Total2
C1	1	60	44386	93	44540	84	145	92	93
C3	1	71	65503	109	65684	101	173	112	113
C5	2	85	106192	133	106412	138	225	151	153
C6	2	102	117802	149	118055	152	256	162	164
C8	2	120	144821	188	145131	212	334	233	235
C10	3	159	167754	234	168150	251	413	292	295
C12	3	163	170549	241	170956	264	430	305	308
Avg	2	109	116715	164	116990	172	282	192	194

To validate this, we employ Kendall correlation (τ) and ROC-AUC (area under the Receiver Operating Characteristic curve) to demonstrate that ATLAS-RTL and ATLAS-Netlist can correctly identify the maximum-power cycles. Specifically, τ measures the global ranking consistency across all cycles; while ROC-AUC formulates the identification of maximum-power cycles as a binary classification task, with the top 5% designated as high-power cycles and the remaining cycles as low-power cycles. By varying the classification threshold on the predicted power values, we generate the ROC curve and compute the AUC. The results are summarized in Table V:

- **τ:** ATLAS-Netlist reaches an average τ of 0.79 (Total) and 0.83 (Comb), while ATLAS-RTL achieves 0.71 and 0.69. Considering that a τ greater than 0.50 is generally regarded as indicating a strong correlation [22], these values demonstrate that both models preserve the relative ordering of cycles with respect to power.
- **ROC-AUC:** Both models achieve near-perfect ROC-AUC scores (0.99 for ATLAS-Netlist and 0.97 for ATLAS-RTL). This demonstrates that they can reliably distinguish high-power cycles from low-power cycles, which is critical for power envelope checking.

D. Runtime Comparison

Table VI presents the time taken for back-end processing using Innovus, the time for power simulations using PTPX®, and the time taken by ATLAS to directly predict power. Here, we included the runtime used for the entire preprocessing step in ATLAS. We calculated the average time (in seconds) for seven designs. Compared with the complete traditional flow that performs both full layout and time-based power simulations, ATLAS achieves over $> 400\times$ speedup in estimating a 300-cycle workload when using netlist toggles, and over $> 600\times$ speedup when using RTL toggles. Additionally, we found that for C1, if the workload length exceeds 60,000 cycles, the time required for toggle propagation will surpass the time for P&R. Therefore, for workloads that require long simulation times (such as operating systems or application segments), the time savings from ATLAS-Netlist to ATLAS-RTL will far exceed those from Traditional Flow to ATLAS-Netlist.

X. Discussion

TABLE VII: Ablation studies on the combinational and clock tree groups demonstrate the effectiveness of our pre-trained encoder. Column names represent: Without Embed. (without using the embedding of our pre-trained encoder).

Testing Design	MAPE error (%) of ATLAS			
	Combinational		Clock Tree	
	Without Embed.	ATLAS	Without Embed.	ATLAS
C1	30.92	9.72	26.89	7.83
C3	33.48	7.37	23.24	11.39
C5	23.26	7.61	22.23	6.01
C6	37.85	9.11	12.38	3.37
C8	35.59	7.84	3.26	1.96
C10	52.25	3.07	10.84	6.48
C12	45.36	7.87	12.80	0.84
Avg	**36.96**	**7.51**	**15.95**	**5.41**

In this section, we provide an in-depth discussion of ATLAS's capabilities and advantages, demonstrate the effectiveness of the self-supervised encoder through ablation studies. We highlight ATLAS's generalization in three aspects: larger and unseen designs, cross-workload capability, and timing diversity. Then we compare different partitioning strategies, and show the model's robustness for clock-gated circuits.

A. Ablation Study about Self-Supervised Encoder

To demonstrate the superiority of the pre-trained encoder in ATLAS, we conduct a decomposed analysis via ablation studies by removing the embedding generated by the pre-trained encoder. During the fine-tuning phase, we train the MLP models using only the other features from the power groups described in Sec. VI, excluding the pre-trained encoder embedding. The training set is still Setting 1.

The results of the ablation study, as shown in Table VII, highlight the critical role of the pre-trained encoder embedding in ATLAS. Specifically, incorporating the embedding (ATLAS) significantly reduces the MAPE across all test designs. For instance, in the combinational group, the average MAPE decreases from 36.96% (Without Embed.) to 7.51% (with ATLAS), and in the clock tree group, the MAPE drops from 15.95% to 5.41%. These findings consistently demonstrate the effectiveness and superiority of our pre-trained encoder in capturing cross-stage netlist variations, including timing optimization, netlist restructuring, and real routing information. The substantial reduction in error further validates the importance of the embedding representations learned through our self-supervised contrastive learning during pre-training.

B. Generalization Capability on Larger and Unseen Designs

Generalization is a crucial metric for models, referring to their ability to adapt to new data. We demonstrate the generalization ability of ATLAS by introducing a challenging experimental setting (Setting 2) as shown in Table VIII. This setting poses a substantial challenge, as the model must generalize from much smaller training examples to much larger and complex designs. As mentioned in Sec. VIII-A, Setting 2 consists of designs C2, C3, and C4, which range from 260K to 400K cells in size. The testing set consists of all other designs, all of which are larger than those in the training set except for C1. C1 is included to demonstrate that the model doesn't fit only to predict large designs. To demonstrate the cross-design capability of ATLAS, we conducted experiments on Xiangshan [23] CPU, an industry-oriented design emphasizing

TABLE VIII: Generalization capability of ATLAS: comparison across experimental settings. Δ is calculated exclusively on the testing designs common to both experimental settings.

Testing Design	MAPE error (%) of ATLAS											
	Register			Clock Tree			Combinational			Total		
	Training Designs:		Δ	Training Designs:		Δ	Training Designs:		Δ	Training Designs:		Δ
	C2,4,7,9,11	*C2,3,4*		*C2,4,7,9,11*	*C2,3,4*		*C2,4,7,9,11*	*C2,3,4*		*C2,4,7,9,11*	*C2,3,4*	
C1	4.69	4.52	-0.17	7.83	10.82	2.99	9.72	8.05	-1.67	4.17	6.27	2.10
C3	5.83			11.39			7.37			3.67		
C5	5.28	5.18	-0.10	6.01	5.10	-0.91	7.61	13.38	5.77	4.40	6.00	1.60
C6	3.54	3.62	0.08	3.37	3.32	-0.05	9.11	12.11	3.00	3.40	4.12	0.72
C8	2.24	2.18	-0.06	1.96	2.64	0.68	7.84	8.28	0.44	2.24	2.15	-0.09
C10	3.17	3.67	0.50	6.48	3.77	-2.71	3.07	5.37	2.30	1.62	1.22	-0.40
C12	1.80	2.29	0.49	0.84	1.43	0.59	7.87	7.75	-0.12	1.83	2.00	0.17
C7		7.41			3.27			17.09			8.86	
C9		3.52			2.10			11.97			3.94	
C11		3.76			10.49			8.65			5.34	
Xiangshan	5.33	5.74	0.41	5.19	4.55	-0.64	15.18	16.50	1.32	3.89	4.04	0.15
Avg	**3.99**	**4.19**	**0.16**	**5.38**	**4.75**	**0.00**	**8.47**	**10.92**	**1.58**	**3.15**	**4.39**	**0.61**

TABLE IX: Comparison of different workloads. The training set includes C2, C3, and C4 under Workload 1 (W1). Each cell shows the MAPE error of four different workloads (W1, W2, W3, and W4).

Testing Design	MAPE Error (%) of ATLAS (Training Set: C2, 3, 4; under W1)															
	Register				Clock Tree				Combinational				Total			
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
C1	4.5	4.3	3.3	3.8	10.8	17.1	15.0	13.9	8.1	12.8	14.2	13.1	6.3	7.6	5.5	5.8
C5	5.2	4.6	4.4	4.2	5.1	6.3	3.9	4.5	13.4	8.8	10.3	9.9	6.0	4.8	4.9	4.7
C6	3.6	2.3	3.1	2.9	3.3	8.6	2.2	3.0	12.1	10.5	9.4	10.1	4.1	4.7	3.2	3.4
C7	7.4	9.2	7.6	7.9	3.3	11.0	1.7	8.6	17.1	13.4	15.1	14.8	8.9	10.8	7.7	9.2
C8	2.2	4.1	4.3	3.9	2.6	1.3	1.2	1.8	8.3	13.0	15.3	14.1	2.2	4.8	5.2	4.8
C9	3.5	3.8	3.2	3.2	2.1	3.4	4.4	3.8	12.0	11.2	8.7	9.2	3.9	4.0	3.0	3.2
C10	3.7	3.5	3.5	3.8	3.8	5.1	5.5	5.1	5.4	9.2	9.9	9.4	1.2	2.9	2.9	3.3
C11	3.8	2.5	2.0	2.4	10.5	10.2	9.7	9.9	8.7	11.7	12.6	9.8	5.3	5.1	4.5	4.3
C12	2.3	2.0	2.2	2.6	1.4	2.7	2.3	2.2	7.8	12.0	12.4	10.4	2.0	4.2	4.6	5.1
Avg	4.0	4.0	3.7	3.9	4.8	7.3	5.1	5.9	10.3	11.4	12.0	11.2	4.4	5.5	4.6	4.9

high frequency, large window size, and high concurrency, in contrast to Boom's simpler structure. The gate-level netlist of Xiangshan contains 1,516,486 gates, and the post-layout netlist contains 1,777,139 gates, which is much larger than C12.

Experimental results in Table VIII demonstrate that training on only three small designs enables accurate power prediction for all larger designs, C5–C12 and Xiangshan. Notably, C10–C12 and Xiangshan are each an order of magnitude larger than the training set. For the largest Xiangshan, compared to Setting 1, the MAPE with Setting 2 increases by only 0.41% and 1.32% in the register and combinational group, respectively, while it even decreases by 0.64% in the clock tree group. For the total power, the MAPE increases by just 0.15%. When considering all testing designs common to both experimental settings, the MAPE increases by only 0.16%, 0.00%, 1.58%, and 0.61% for the register, clock tree, combinational logic, and total power, respectively. These results further demonstrate that ATLAS possesses good generalization and cross-design capability, enabling accurate prediction for significantly larger designs, whose sizes are unprecedented for the model.

C. Cross-Workload Generalization Capability

We used four different workloads [24] as inputs to evaluate ATLAS's cross-workload performance. W1 is the workload used in all previous experiments and is the Median Filter benchmark. W2 is the Towers of Hanoi benchmark; W3 is the Vector-Vector Add benchmark; W4 is the Dhrystone synthetic integer benchmark. It is important to note that in this experiment, we did not retrain the model; the model remains the same as in Sec. X-B, where C2, C3, and C4 under W1 were used as the training set. The experimental results are shown in Table IX, where ATLAS achieves a total power prediction MAPE of 5.5%, 4.6%, and 4.9% for W2, W3, and W4, respectively, compared to 4.4% for W1. Considering that the finetuned model was trained on W1, we believe this slight increase is minimal. This demonstrates that ATLAS achieves good cross-workload performance.

TABLE X: MAPE across different testing designs with training designs [C2, C3, C4] at various clock frequencies.

Test	MAPE (%) of ATLAS															
	0.5 GHz				1 GHz				2 GHz				4 GHz			
	clk	reg	comb	total	clk	reg	comb	total	clk	reg	comb	total	clk	reg	comb	total
C1	5.8	4.4	9.0	3.9	6.4	4.3	9.1	4.0	7.5	4.2	11.1	6.6	8.3	4.3	12.0	6.8
C2	Training Set								4.9	2.1	7.3	3.9	5.2	2.1	7.9	4.0
C3									3.5	2.0	6.7	3.5	3.9	2.2	6.6	3.8
C4									4.4	1.7	6.8	3.3	4.7	1.6	7.2	3.5
C5	6.1	3.6	7.7	4.9	5.3	3.6	7.0	4.6	7.3	3.5	10.4	6.8	8.0	3.5	10.9	7.1
C6	5.5	3.3	9.0	3.3	4.2	3.4	9.2	3.2	6.8	3.4	12.3	5.7	7.4	3.5	13.0	6.9
Avg	**5.8**	**3.8**	**8.6**	**4.0**	**5.3**	**3.7**	**8.4**	**3.9**	**5.7**	**2.8**	**9.1**	**5.0**	**6.2**	**2.9**	**9.6**	**5.4**

D. Generalization Capability on Timing Diversity

To evaluate the generalization capability of timing diversity, we extended our experiments beyond the original 1GHz setting. Specifically, synthesis and physical design flows were re-run for six designs (C1–C6) at three target frequencies: 0.5GHz, 2GHz, and 4GHz, resulting in a total of 18 new flows. Frequency was incorporated as an additional feature in both the pre-training and fine-tuning stages: 1) In the pre-training stage, frequency was added to each graph node. 2) In the fine-tuning stage, frequency was included as an input feature to the MLPs predicting the three power groups.

Designs C2, C3, and C4 at 0.5GHz and 1GHz were used for training, while the remaining designs served as test cases. We use the Median workload (W1) here. As shown in Table X, ATLAS achieves a total power MAPE of around 4%–5% across this challenging triple-cross scenario (cross stages, cross designs, and cross timing targets). These results demonstrate that ATLAS maintains an acceptable generalization capability even under varying timing constraints.

E. Ablation Study about Different Partitioning Methods

In the Sec. IV-B, we discussed the impact of different partitioning methods on ATLAS. To further demonstrate the superiority of the hMetis-based partitioning method, we conducted additional comparative experiments based on Sec. X-B, with the results presented in Table XI. In these experiments, only the sub-circuit format described in Sec IV-B was varied; all other procedures remained the same, and the training designs were still C2, C3, and C4. The workload is W1

TABLE XI: Comparison of different partitioning methods. Training set includes C2, C3, and C4. Each cell shows the MAPE error of the hierarchy-based partitioning method and ATLAS for each group. Column names represent: hier. (hierarchy-based partitioning method), ATLAS (hMetis-based [12] partitioning method)

Testing Design	MAPE Error (%) of Different Partitioning Methods							
	Register		Clock Tree		Combinational		Total	
	hier.	ATLAS	hier.	ATLAS	hier.	ATLAS	hier.	ATLAS
C1	**2.02**	4.52	**5.91**	10.82	9.67	**8.05**	**3.85**	6.27
C5	7.65	**5.18**	**4.51**	5.10	15.81	**13.38**	9.05	**6.00**
C6	4.13	**3.62**	7.47	**3.32**	14.22	**12.11**	6.27	**4.12**
C7	15.60	**7.41**	12.58	**3.27**	20.90	**17.09**	16.73	**8.86**
C8	19.57	**2.18**	16.50	**2.64**	15.98	**8.28**	17.47	**2.15**
C9	15.14	**3.52**	14.02	**2.10**	14.09	**11.97**	14.37	**3.94**
C10	21.27	**3.67**	20.40	**3.77**	14.19	**5.37**	19.28	**1.22**
C11	21.94	**3.76**	19.49	**10.49**	10.99	**8.65**	17.76	**5.34**
C12	26.93	**2.29**	22.07	**1.43**	22.19	**7.75**	24.07	**2.00**
Avg	14.92	**4.02**	13.66	**4.77**	15.34	**10.29**	14.32	**4.43**

TABLE XII: MAPE error for designs with and without clock gating.

Testing Design		MAPE Error (%)	
		Register	Clock Tree
C2	Without clock gating	1.78	4.70
	With clock gating	2.12	6.98
C3	Without clock gating	1.08	1.96
	With clock gating	7.82	9.83

The comparative results indicate that the hierarchy-based partitioning method only outperforms ATLAS in predicting total power for smaller designs. Across all larger testing designs, ATLAS consistently outperforms the hierarchy-based method, as evidenced by the lower MAPE values (with the best results in each row highlighted in bold). For example, the average total MAPE across all designs drops from 14.32% (hierarchy) to 4.43% (ATLAS), representing a substantial relative improvement.

These results support that ATLAS generalizes significantly better than the hierarchy-based partitioning method. The reasons for the poorer generalization of the hierarchy-based partitioning method can be summarized as follows:

1) Sub-circuit size variation across designs: Hierarchy-based partitioning divides chips into functional sub-circuits, but large size variability across designs makes training sets less representative. ATLAS solves this by partitioning netlists into sub-circuits of nearly equal size, which improves generalization.

2) Hierarchy-less cells in post-layout stage: At the post-layout stage, many frequently toggling cells—such as buffers and inverters on the clock tree—do not belong to any hierarchy. In contrast, the gate-level stage has almost no hierarchy-less cells, so it's hard to predict the power of this part.

F. Comparison of Clock Gating

In Fig. 7, we present the label waveforms for the register group and clock tree group of C3 with and without clock gating (note that the combinational group is not compared since clock gating does not affect it). It can be observed that, under the same y-axis scale, the register group waveform without clock gating exhibits relatively smooth fluctuations, while the clock tree group is nearly a flat line. In contrast, for the design with clock gating, both groups display larger and more irregular fluctuations.

We also perform a small-scale experiment under W1, using C1 and C4 as the training set and C2 and C3 as the testing set, and find that designs with clock gating present greater challenges for power prediction, as shown in Table XII. In particular, MAPE for the register and clock-tree groups is higher with clock gating, confirming the increased difficulty. In summary, we extended ATLAS to accurately predict power in clock-gated designs, addressing a more challenging and realistic class of power-estimation problems.

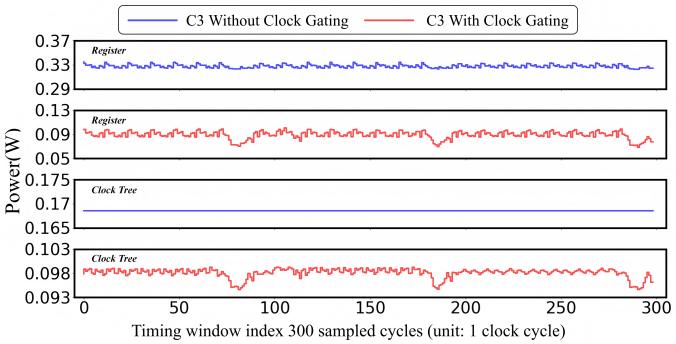

Fig. 7: A comparative analysis of the label waveforms for the register part and the clock tree part in design C3.

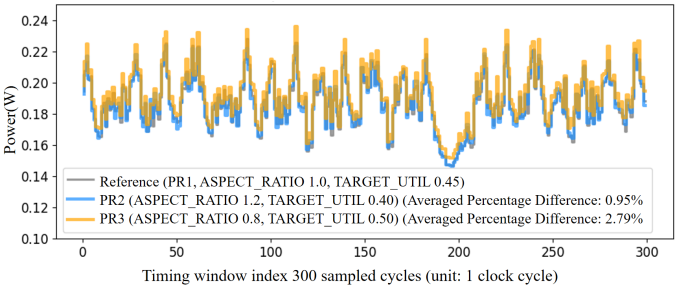

Fig. 8: Different layout power consumption of C1 under W1.

G. Limitation and Future Work

ATLAS is currently the only work in academia capable of cross-design and cross-stage prediction of combinational logic, clock tree, register, and total power. The results of ATLAS shown in Table III are highly impressive. Given the limited datasets and resources available to us, we believe ATLAS can be further improved in the following directions:

1) Support for multiple power delivery networks: Industrial designs often have multiple power islands, while the designs in our dataset contain only one. Migrating ATLAS to handle designs with multiple power islands is feasible; this can be achieved by adding voltage as a feature to the model or by training separate models for different power islands.

2) Support for multiple layouts: Currently, our placement and routing flow is fixed. If the floorplan's aspect ratio or utilization changes, the post-layout netlist may also change slightly. As shown in Figure 8, we generated different layouts of Design C1 by adjusting flow parameters. PR1 is the original layout (aspect ratio 1.0, utilization 45%), while PR2 (1.2, 40%) and PR3 (0.8, 50%) yield only 0.95% and 2.79% average differences in total power under W1, respectively. These small variations confirm that ATLAS using a single layout as the training label is reasonable. Moreover, ATLAS already achieves accurate post-layout power estimation under a fixed flow. If the flow changes, the modified aspects (e.g., aspect ratio, utilization) can be incorporated as additional features in the model, enabling prediction across different layouts as well. Since generating multiple layouts is time- and resource-intensive, future work could explore more efficient methods to obtain layouts of comparable quality.

XI. Conclusion

In this paper, we propose ATLAS, a pioneering framework for fine-grained per-cycle power prediction. ATLAS is the first work that supports both time-based power simulation and general cross-design power modeling. It achieves unprecedented general time-based power modeling based on a customized pre-training and fine-tuning paradigm. When evaluated on the prediction of per-cycle post-layout power based on gate-level netlist, ATLAS achieves a remarkable MAPE of 3.05% for total power estimation with inference speeds that are $> 400\times$ faster than traditional flow. Furthermore, ATLAS is extended to bypass the toggle propagation process, achieving an average error percentage of only 5.00% and delivering up to $600\times$ speedup compared to the traditional commercial flow.

References

[1] Synopsys, "PrimePower: RTL to signoff power analysis," 2023.
[2] Y. Zhou *et al.*, "Primal: Power inference using machine learning," in *Proc. Design Automation Conf. (DAC)*, 2019, pp. 1–6.
[3] Z. Xie *et al.*, "Apollo: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *MICRO-54*, 2021.
[4] P. Sengupta *et al.*, "How good is your Verilog RTL code? A quick answer from machine learning," in *ICCAD*, 2022.
[5] C. Xu *et al.*, "Sns's not a synthesizer: a deep-learning-based synthesis predictor," in *ISCA*, 2022, p. 847–859.
[6] C. Xu, P. Sharma, T. Wang, and L. W. Wills, "Fast, robust and transferable prediction for hardware logic synthesis," in *MICRO*, 2023.
[7] W. Fang *et al.*, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *ICCAD,*, 2023, pp. 1–9.
[8] Y. Du *et al.*, "Powpredict: Cross-stage power prediction with circuit-transformation-aware learning," in *DAC*, 2024, pp. 1–6.
[9] Y. Zhang *et al.*, "Grannite: Graph neural network inference for transferable power estimation," in *DAC*, 2020, pp. 1–6.
[10] M. B. Rakesh *et al.*, "Gript: Graph attention-assisted inductive methodology for fast and accurate average power estimation from rtl simulation skipping gate-level simulation," *IEEE TCAD*, 2025.
[11] Z. Wang *et al.*, "Functionality matters in netlist representation learning," in *DAC*, 2022, p. 61–66.
[12] G. Karypis *et al.*, "Multilevel hypergraph partitioning: applications in vlsi domain," in *IEEE Transactions on VLSI Systems*, 1999, pp. 69–79.
[13] Z. Shi *et al.*, "Deepgate2: Functionality-aware circuit representation learning," in *ICCAD*, 2023, pp. 1–9.
[14] *TSMC 40nm LP process technology*, https://www.tsmc.com/ english/dedicatedFoundry/technology/logic/l_40nm, 2008.
[15] Q. Wu *et al.*, "Sgformer: Simplifying and empowering transformers for large-graph representations," in *NeurIPS*, 2023, pp. 1–21.
[16] K. Xu *et al.*, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019, pp. 1–17.
[17] Q. Wu *et al.*, "DIFFormer: Scalable (graph) transformers induced by energy constrained diffusion," in *ICLR*, 2023, pp. 1–26.
[18] Q. Wu, W. Zhao *et al.*, "Nodeformer: a scalable graph structure learning transformer for node classification," in *NeurIPS*, 2024.
[19] Synopsys, "VCS® functional verification solution," 2021.
[20] "Design Compiler® RTL Synthesis," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html.
[21] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, 2020.
[22] A. R. Gilpin, "Table for conversion of kendall's tau to spearman's rho within the context of measures of magnitude of effect for meta-analysis," *Educational and Psychological Measurement*, pp. 87 – 92, 1993.
[23] Y. Xu *et al.*, "Towards Developing High Performance RISC-V Processors Using Agile Methodology," in *MICRO*, 2022, pp. 1178–1199.
[24] riscv tests, "https://github.com/riscv-software-src/riscv-tests/."

Wenkai Li is currently a Ph.D. student with the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. He received his master's degree from the Hong Kong University of Science and Technology in 2024 and his B.Eng. degree from Harbin Institute of Technology (Shenzhen) in 2023. His research interests include Electronic Design Automation (EDA).

Yao Lu received the B.E. degree from Southeast University, in 2020, and the master's degree from Fudan University, in 2023. She is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong. Her current research interests focus on machine learning applications in Electronic Design Automation.

Wenji Fang is currently a Ph.D. student with the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. He received his M.Phil. degree from the Hong Kong University of Science and Technology (Guangzhou) in 2024, and his B.Eng. degree from Nanjing University of Aeronautics and Astronautics in 2021. His research interests include Electronic Design Automation and VLSI design verification.

Yugao Zhu received the Bachelor degree in electronic engineering from Tsinghua University, Beijing, China, in 2022. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology. His research interests include graph algorithms, data mining.

Ziyan Guo is currently a Ph.D. student with the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. He received his M.Phil. degree in Singapore University of Technology and Design in 2025, and his B.Eng. degree from Harbin Institute of Technology in 2023. His research interests include Electronic Design Automation (EDA).

Jing Wang received the B.S. degree from Peking University, in 2022, and the master's degree from the University of Hong Kong, in 2023. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology. His research interests include agile VLSI design methodologies.

Mengming Li is currently a Ph.D. student in the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology. His research interests include Computer Architecture and EDA. He has authored multiple papers published in top-tier computer architecture conferences, including ISCA, MICRO, and HPCA.

Qijun Zhang received the B.Eng. degree from Tongji University, Shanghai, China, in 2022. He is currently a Ph.D. student in the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). His research interests include Computer Architecture and Electronics Design Automation.

Zhiyao Xie is an Assistant Professor of the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). Zhiyao received his Ph.D. degree from Duke University in 2022 and B.Eng. from City University of Hong Kong in 2017. His research interests include machine learning algorithms for EDA and VLSI design. He has received multiple prestigious awards, including the IEEE/ACM MICRO 2021 Best Paper Award, ACM SIGDA SRF Best Research Poster Award 2022, ASP-DAC 2023 Best Paper Award, ACM Outstanding Dissertation Award in EDA 2023, EDAA Outstanding Dissertation Award 2023, and the 2023 Early Career Award from Hong Kong Research Grants Council (RGC).