Elastic EDA: Auto-scaling Cloud Resources for EDA Tasks via Learning-based Approaches

Linyu Zhu*, Xingyu Ma†, Shaogang Hao†, Yushan Pan‡ and Xinfei Guo*§¶
*University of Michigan – Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
†Tencent Quantum Lab, Tencent, Shenzhen, China
‡Xi'an Jiaotong-Liverpool University, Suzhou, China
§State Key Laboratory of Integrated Chips and Systems (SKLICS), Shanghai, China
{linyuzhu, xinfei.guo}@sjtu.edu.cn

***Abstract*—Utilizing cloud EDA for chip design allows access to on-demand high-performance computing (HPC) resources, significantly reducing development time and costs by eliminating the need for costly on-site infrastructure. Despite its benefits, cloud EDA faces significant challenges, primarily the lack of an effective cost model. A key issue is the absence of a mechanism for designers to accurately gauge the characteristics of their EDA jobs in cloud environment, as the design process involves a multitude of EDA tools and steps, often leading to the over or underestimation of needed computational resources. This problem is exacerbated by the varying computational demands of different designs and constraints. To bridge this knowledge gap, we introduce Elastic EDA, a methodology that harnesses machine learning (ML) to understand the characteristics of a design and its early stages, and to predict the computational needs for subsequent phases throughout the entire EDA flow. This approach effectively aligns design behaviors with computational resources, providing cost-efficient solutions for various cloud EDA scenarios. Compared to previous ML-based predictive frameworks for cloud EDA, the proposed method achieves over 60% higher prediction accuracy and supports various elastic computing environments, maximizing the efficiency of cloud resources. Compared to various baseline scheduling configurations in the cloud environment, the proposed framework achieves over 16% mean runtime improvement.**

***Index Terms*—Cloud, EDA, Elastic computing, Machine learning, Resource prediction**

I. Introduction

As the complexity and variety of modern chips grow, Electronic Design Automation (EDA) workloads exhibit varying demands on computing resources, with significant peaks and bursts. EDA tasks such as place and route, signoff, and analysis often require more processing power than in-house computing machines can offer. Migrating EDA workloads to the cloud allows for dynamic scaling of compute infrastructure, providing the necessary capacity when needed. This capability enables design teams to accelerate their product development cycles and reduce time to market by leveraging the cloud's vast compute, storage, and other resources [1], [2].

¶Corresponding author.

This work is supported in part by the National Science and Technology Major Project (Grant No. 2021ZD0114701), the State Key Laboratory of Integrated Chips and Systems (SKLICS) open fund, the CCF-Tencent Open Research Fund, and the SJTU Explore-X fund. Additional support is provided by XJTLU RDF-21-02-008 and the Jiangsu Provincial Double Initiative Plan JSSCBS20230474.

Despite these advantages, EDA-on-Cloud solutions face significant challenges related to cost and efficiency. Firstly, from the cloud vendor's perspective, it is critical to optimize the use of cloud computing resources when running various EDA tasks. Vendors typically use mechanisms to detect idle machines for newly scheduled tasks. However, these strategies can lead to suboptimal resource usage due to limited understanding of EDA task behaviors. Secondly, from the design house's perspective, it is crucial to identify cost-saving opportunities before purchasing cloud services. This requires accurate estimation of the computational scale, which informs decisions about the amount and duration of required resources. Such estimation models are currently lacking. Lastly, from designer's perspective, scheduling an EDA job on the cloud demands a deep understanding of task characteristics in cloud environments. Unfortunately, as noted by [3], there is little public information available on these characteristics. A model addressing this gap should aim to optimize the allocation of computing resources tailored to each specific EDA task, maximizing efficiency and effectiveness. Although there has been extensive work on predicting server loads for cloud computing independent of specific applications—examples include [4]–[7]—migrating EDA jobs to the cloud is not a straightforward process, especially for design teams with little or no experience managing cloud resources. Additionally, EDA tasks are unique and exhibit sequential behaviors that differ from many other cloud-based tasks. For instance, selecting the appropriate machine configuration based on a specific design at the beginning of the design phase is crucial. However, the compute requirements for specific stages running in a cloud environment remain underexplored. To develop such a usage model, it is crucial to understand both the cloud infrastructures and the detailed EDA behaviors, which can vary significantly among different designs and design environments.

EDA workloads often require a compute cluster, a process scheduler that orchestrates job distribution to compute nodes, and a high-performance shared file system. At a high level, a computer cluster in a cloud environment consists of two or more computers, or nodes, that work in parallel to achieve a common goal. This setup allows EDA workloads, which involve numerous individual, parallelizable tasks, to be distributed among the nodes in the cluster. As a result, these

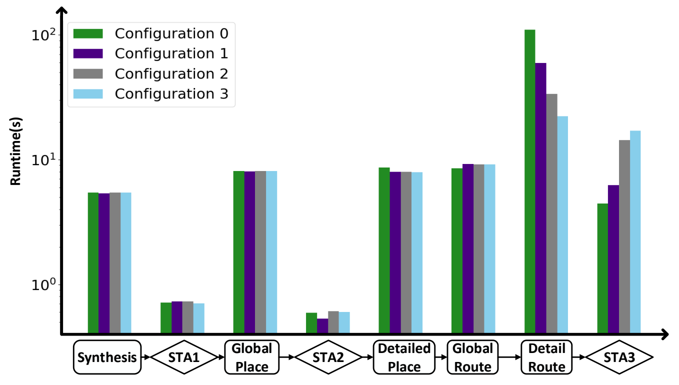


Fig. 1: A typical open-source EDA flow running a CPU design `picorv32` involves different CPU configurations at each stage. Configuration 0 corresponds to the least powerful single-core CPU, while Configuration 3 represents a more powerful eight-core CPU (Configuration $0 < 1 < 2 < 3$). The task scale at each stage varies significantly, and the runtime differs based on the CPU configuration used.

tasks can utilize the combined memory and processing power of each computer, enhancing overall performance. To simulate this behavior, we conducted an experiment by running a typical EDA flow based on the open-source OpenRoad project [8], [9] on various CPU configurations, ranging from one core (Configuration 0) to eight cores (Configuration 3). The design used is an open-source RISC-V CPU core named `picorv32` [1]. The entire flow begins with synthesis and proceeds through place and route (P&R). Between these major stages, static timing analysis (STA) processes are frequently invoked to check timing. Runtime is measured for each stage and each CPU configuration, with the results shown in Fig. 1. The results clearly indicate that the task scale at each stage varies significantly, with later stages typically taking longer due to the complexity of tasks in the P&R stages. Among the four CPU configurations, Configuration 3 shows a clear advantage in the detailed routing stage but exhibits contradictory behavior in the STA3 stage. Overall, this suggests that a more powerful CPU does not necessarily lead to the fastest runtime. Similar observations have been made for other designs and CPU configurations (In Section IV, more experimental results with various design scales will be discussed). It is widely known that current EDA algorithms are not specifically optimized to take advantage of parallelism, even when available. These observations further demonstrate that parallel machines can sometimes result in minimal runtime reduction or even increased runtime for certain stages. We hypothesize that this behavior may be due to the STA tool used in our experiment not optimized for parallelization effectively, leading to extra time spent on managing parallel tasks and synchronization

[1]It should be noted that although the selected tiny CPU core is not likely a representative scale for cloud-based EDA, we still chose it because the key focus is on the runtime behavior among various machines, which remains representative.

overhead. Thus, this necessitates a better approach to allocate cloud resources for sequential EDA tasks. A more sophisticated resource allocation model is needed to optimize the use of cloud infrastructure, ensuring that each EDA task receives the appropriate amount of computing power based on its specific requirements and behavior. This approach would improve efficiency and effectiveness, particularly in stages where parallelism does not significantly reduce runtime.

Inspired by prior observations, this paper proposes *Elastic EDA*, a methodology that leverages learning-based approaches to understand the characteristics of a design in its early stages and predict the computational needs for subsequent phases throughout the entire EDA flow. This represents the first attempt to tailor cloud resources based on actual design behavior and EDA task characteristics. The key contributions of this framework are summarized as follows:

- The *Elastic EDA* framework features a prediction model that gauges resource usage based on the assumption that design details are not visible. This enhances the elastic computing model from the cloud vendor's perspective by considering the actual EDA task behaviors occurring at the beginning of the flow.
- The framework also includes a prediction model that integrates temporal and spatial information to forecast computational resources needed for subsequent design stages, based on actual design details and early stage characteristics. This helps designers schedule the optimal computational process for individual EDA tasks.
- Validated across various use scenarios, the proposed framework achieves a significantly higher prediction accuracy compared to recently proposed models in a similar category. This leads to greater efficiency and better resource utilization.

II. Related Work

From the cloud provider's perspective, Scale-Out Computing has been introduced to quickly deploy an EDA environment on Amazon AWS [10]. Google Cloud uses configurations such as OpenPiton, Icarus, SLURM, and its standard compute setups to accelerate EDA verification tasks [11]. Additionally, major EDA companies have begun developing cloud-optimized EDA deployment models [12], [13]. Several attempts have been made in academia to deploy EDA on the cloud. Examples include prototypes like the Cloud-EDA Platform for design and testing [14] and chip verification [15], [16]. The Cloud Columba project provides users with easy access to state-of-the-art design automation approaches, regardless of their end device's computing power. Users simply formulate their design requests, and the cloud server synthesizes a customized manufacturing-ready biochip design [17].

While it is exciting to see both industry and academia embracing EDA-on-Cloud solutions, there are potential challenges [1], [2]. Two major challenges hinder the advancement of EDA-on-Cloud: security issues and workload category mapping. Security concerns about moving unique and business-critical proprietary IP to a public cloud have slowed this transition. Recent work focuses on IP protection methods to resolve these obstacles, employing novel data and storage migration management technologies [18], [19]. Workload category mapping is another concern. Since EDA workloads are user-driven, realistic workload models must include design behavior patterns linked to tasks. This aligns with the motivation behind our work: developing a framework that links design characteristics with computing resources. A closely related work is by [3], which proposes a model based on graph convolutional networks (GCN) to predict the total runtime of a given stage based on different configurations. This approach uses RTL-level design to build an and-inverter graph (AIG) and then predicts the runtime of each EDA stage. However, this method highly depends on an IP AIG graph dataset, which overlooks the fact that design IPs are sometimes invisible to the cloud vendor due to security concerns, rendering the solution infeasible in such scenarios. Moreover, it ignores that the same design with different constraints can result in completely different optimization efforts and varying runtime. Our proposed Elastic EDA platform addresses these restrictions by providing a more generalized solution adaptive to various cloud use scenarios. It considers the behaviors of the design and EDA tools and the uniqueness of each run by incorporating runtime data from earlier design phases.

III. Proposed Elastic EDA Framework

A. Customized Cloud EDA Platform

Cloud computing services are primarily offered in three ways: Infrastructure as a Service (IaaS), where users get virtual access to physical hardware resources; Platform as a Service (PaaS), where users run and manage their applications while abstracting away the underlying server administration; and Software as a Service (SaaS), where users get on-demand access to software without having to manage their own installations. While PaaS is convenient for distributing EDA tasks, most EDA software tools have strict IP protection, making them difficult to reinstall. Therefore, a hybrid approach combining IaaS, PaaS, and SaaS is commonly used for EDA on cloud platforms.

An example of a customized cloud EDA platform is illustrated in Fig. 2. After creating a team, users access the platform using a secret ID/Key. Each team needs to create a master cluster node for team management, remote desktop access, task submission, data transfer, and EDA tools access. For storage, multi-tier cloud file storage is used, independent of cluster nodes. For job submissions, there are typically two modes:

- **Elastic Compute Mode**: Computing nodes are created at the beginning of a task and destroyed at the end, following the task lifecycle. Billing stops once the resources are destroyed (provider supplies what is needed). Users do not need to pre-configure compute nodes. After the master cluster node submits a task, elastic resources are automatically managed: created when the task is submitted and destroyed after the task is completed.
- **Cluster Compute Mode**: Users manually create and destroy compute nodes to execute tasks (user buys what they need). After the node is used, it must be manually destroyed to stop billing. Users must pre-configure compute nodes, such as CPU, memory, and GPU.

Though the two submission modes differ in their resource allocation strategies, they share a common goal: to maximize the available computation resources and deliver the best performance for customized EDA tasks.

B. Problem Formulation

Based on the two submission modes employed in a typical EDA-on-Cloud platform, we aim to formulate the problem of runtime prediction accordingly. For elastic compute mode, which operates on a pay-as-you-go basis, it is crucial for the cloud platform to accurately predict task scale to avoid over or under-utilization. Current elastic computing schedulers often fail to understand the actual EDA task behavior and its resource requirements. In this mode, design details are typically hidden from the cloud platform due to security concerns. However, logs and runtime statistics from completed runs are available. Given this limited information, the prediction problem can be addressed using a sequential model based on temporal data, with the cloud platform as the consumer of the prediction model. For cluster compute mode, users need to select their own hardware configuration, with the cost being proportional to the rental time of the chosen setup. In this mode, both design details and runtime statistics are visible to the user, providing both spatial and temporal information for the prediction task. A learning-based model is developed to help users estimate their task configuration, runtime, and cost, and to enable auto-scaling of the cluster.

C. Overview of the Learning-based Auto-scaling Cloud EDA Framework

Cloud platforms naturally offer cost-effective computing resources and access to extensive user groups, facilitating the collection of training datasets. Although users are concerned about IP security, secure training methods like Federated Learning (FL) can be easily implemented on the cloud. Leveraging these conditions, we propose a general learning-based auto-scaling framework to enhance the efficiency of EDA tasks on the cloud for both users and the platform, as shown in Fig. 3. Unlike other EDA prediction frameworks, our framework integrates real EDA cloud scenarios with a learning-based model, providing reasonable interpretability. The key elements of the proposed framework are discussed in the following sections.

1) Task Hardware Monitor: The task hardware monitor tracks hardware usage in the cloud environment, providing detailed process information. In Elastic Computing mode, the cloud platform has monitoring permissions, while in Cluster Computing mode, the user holds these permissions. As shown in the figure, sequential information (such as runtime and constraints of a series of stages) is extracted from the monitor logs.

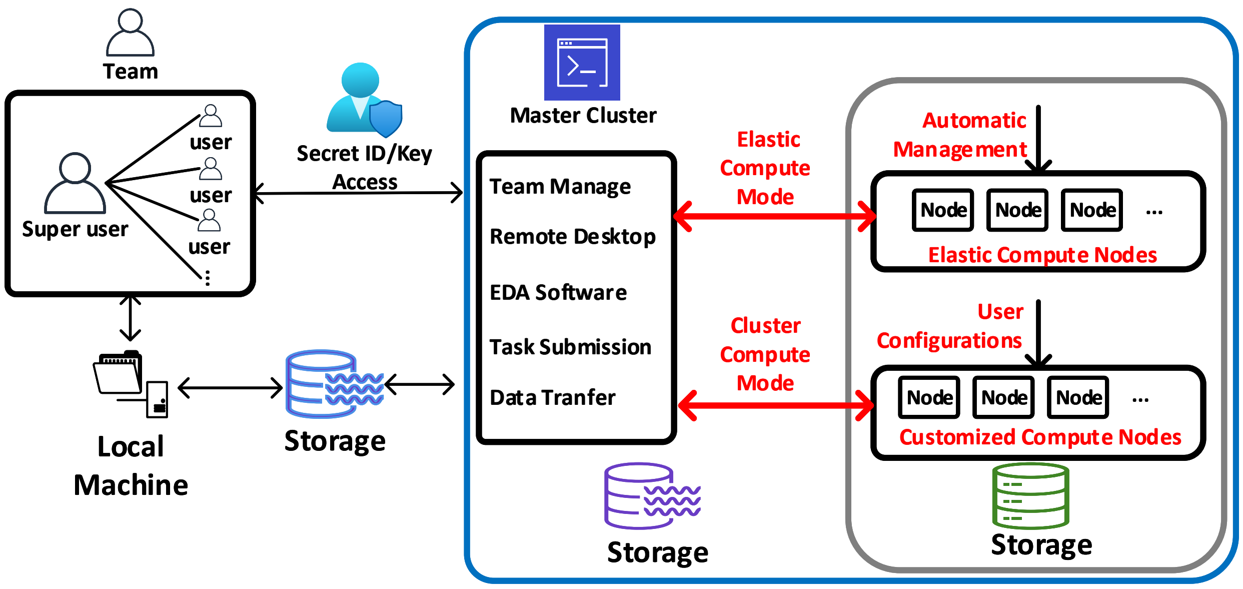
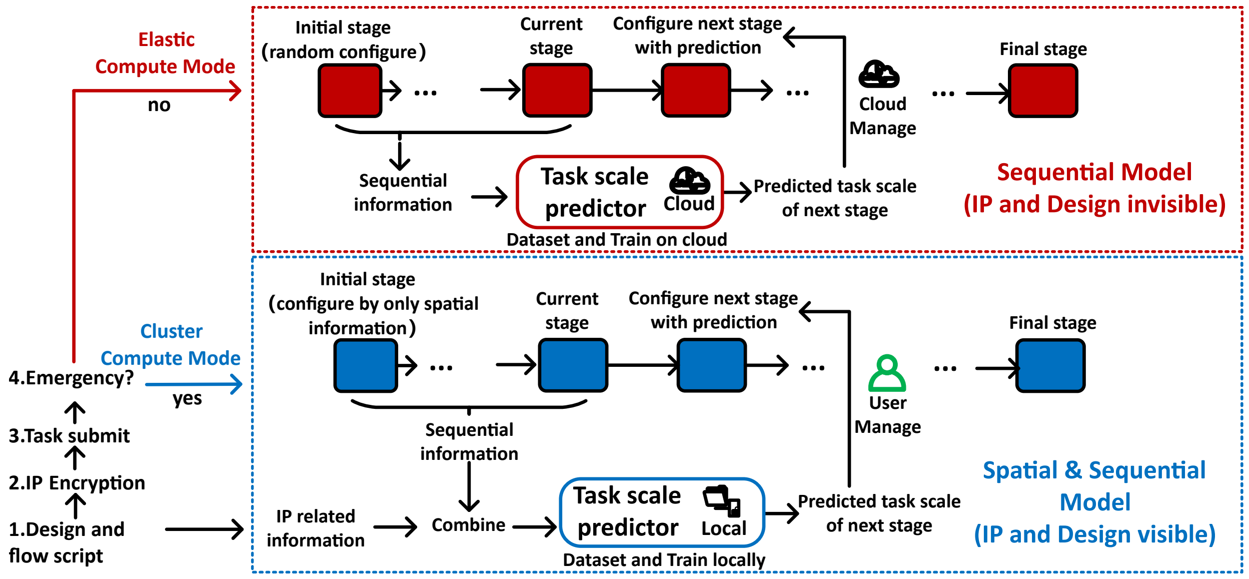


Fig. 2: An example of a typical EDA-customized cloud platform architecture includes multi-level storage, team user management, secure cloud access, and two modes for distributing EDA tasks.



Elastic Compute Mode
no
Initial stage (random configure)
Current stage
Configure next stage with prediction
Cloud Manage
Final stage
Sequential information
Task scale predictor
Cloud
Predicted task scale of next stage
Dataset and Train on cloud
Sequential Model (IP and Design invisible)
Cluster Compute Mode
4.Emergency?
yes
3.Task submit
2.IP Encryption
1.Design and flow script
Initial stage (configure by only spatial information)
Current stage
Configure next stage with prediction
User Manage
Final stage
Sequential information
IP related information
Combine
Task scale predictor
Local
Predicted task scale of next stage
Dataset and Train locally
Spatial & Sequential Model (IP and Design visible)

Fig. 3: The proposed learning-based auto-scaling Elastic EDA framework.

2) *Task Scale Predictor:* For the two job submission modes, represented in blue and red, the task hardware monitor provides limited sequential information from the previous completed EDA stages. In Elastic Computing mode (red), a sequential model is used as a predictor to help the cloud platform estimate the scale of the next submission. In Cluster Computing mode (blue), IP-related information is also visible and can be utilized by the predictor. Details of these models will be discussed in Section III-D.

3) *Dataset and Training:* For their own flow and tools, users can create their own dataset and train locally, ensuring IP information remains secure. In Elastic Computing mode, secure datasets and training methods are used on the cloud to protect user information.

4) *Feature Selection:* The proposed framework offers expandable feature selection, making any sequential information suitable for the corresponding prediction task. For the same design, design constraints (e.g. clock period, optimization efforts, etc.) are also key parameters for task scale and can be included in the sequential information. While IP encryption is common for EDA-on-Cloud tasks, our findings indicate that typical encryption does not significantly alter the workload requirements for a design. To validate this, we conducted an experiment where five representative designs were selected, and a pseudo-encryption algorithm was applied. This pseudo-encryption, simulating a widely used IP encryption algorithm,

TABLE I: Comparison of workload between original and encrypted designs: Five representative designs were selected and tested under the same machine configuration.

Design		#Instructions	#CPU Cycles	Runtime (s)
AES	**Original**	22,342,940,637	35,957,329,062	8.626628428
	Encrypted	19,638,121,154	35,183,232,170	8.522584263
APU	**Original**	28,521,657,474	47,981,666,910	11.556868033
	Encrypted	28,989,785,279	48,298,568,990	11.646499922
BM64	**Original**	40,059,617,955	67,675,644,064	16.274650208
	Encrypted	40,458,037,561	69,832,686,347	16.777453340
xtea	**Original**	12,416,901,332	19,627,392,068	4.727457353
	Encrypted	11,808,841,891	20,383,888,794	4.897448668
yhuff	**Original**	86,390,603,256	144,141,387,253	34.635229694
	Encrypted	85,537,851,182	142,438,265,468	34.271870977

involved manually replacing some gates at random. The original and encrypted designs were then tested under the same configuration, with the number of instructions, CPU cycles, and runtime during the synthesis stage recorded in Table I. The results suggest that minor design alterations lead to similar workloads. Consequently, our predictor is designed to ensure that slight changes in design, such as those from encryption, do not significantly impact IP-related information.

5) Configuration Algorithms: When a task is urgent for the user, Cluster Computing is the preferred choice as it allows customization of hardware resources. For both modes, once the task scale of the next EDA stage is predicted, multiple configuration algorithms are created to select the right machine. For instance, users can balance cost and computing runtime or opt for a policy that minimizes runtime. In our experiment (Section IV), we use greedy least runtime in each stage for Cluster Computing and greedy least total batch tasks runtime for Cluster Computing, neglecting cost.

D. Learning-based Prediction Models

The task scale predictor plays a crucial role in the proposed framework, bridging the gap between EDA on cloud characteristics and the cloud environment. Previous research [3] only addresses scenarios where IP information is visible, which aligns with our cluster compute mode. In contrast, our paper introduces prediction models tailored for both job submission modes.

1) Sequential Model: Given that the EDA flow can be modeled as a sequential process, where the output of one stage serves as the input for the subsequent stage, the utilization of hardware resources is closely correlated. Moreover, the monitoring of hardware resource consumption is publicly accessible for cloud platforms already. Hence, we employ sequential models to encode input parameters from previous design stages sequentially. This approach enables us to model dependencies by considering the encoded vectors from various stages as time-domain dependent information, facilitating the prediction of task scale for the next stage. The detailed architecture of our sequential model is shown in the left panel of Fig. 4. In our experiment, input sequence X_t of each stage is embedded from current stage configuration (CPU with different cores) and current stage task scale (corresponding runtime), which is measured by hardware resource monitoring. The predicted task scale y_t of next stage will be used to help configure computing resource of next stage. In our experimentation, we explore three mainstream sequential models as Sequential Layers.

The t^{th} stage of the EDA flow represents step t. The Recurrent Neural Network (RNN) is a fundamental sequential model, allowing the output from the previous step is fed as input to the current step:

$$h_t = RNN_{enc}(X_t, h_{t-1})$$
$$s_t = RNN_{dec}(y_t, s_{t-1})$$

Here, h_t denotes the encoder hidden state at time step t, with input sequence X_t, while s_t represents the decoder hidden state at time step t, with output y_t. Likewise, the Long Short-Term Memory (LSTM) network is a modified version of the RNN network, incorporating a backward connection. The basic LSTM architecture comprises three gates: the input gate I, forget gate F, and output gate O. These gate connections help retain long-term memories within the network, preserving information from previous time steps. When given an input sequence X_t at time step t, the gate connections are governed as follows:

$$I_t = S(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$
$$F_t = S(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$
$$O_t = S(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$
$$\widetilde{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$
$$C_t = F_t \odot C_{t-1} + I_t \odot \widetilde{C}_t$$
$$H_t = O_t \odot tanh(C_t)$$

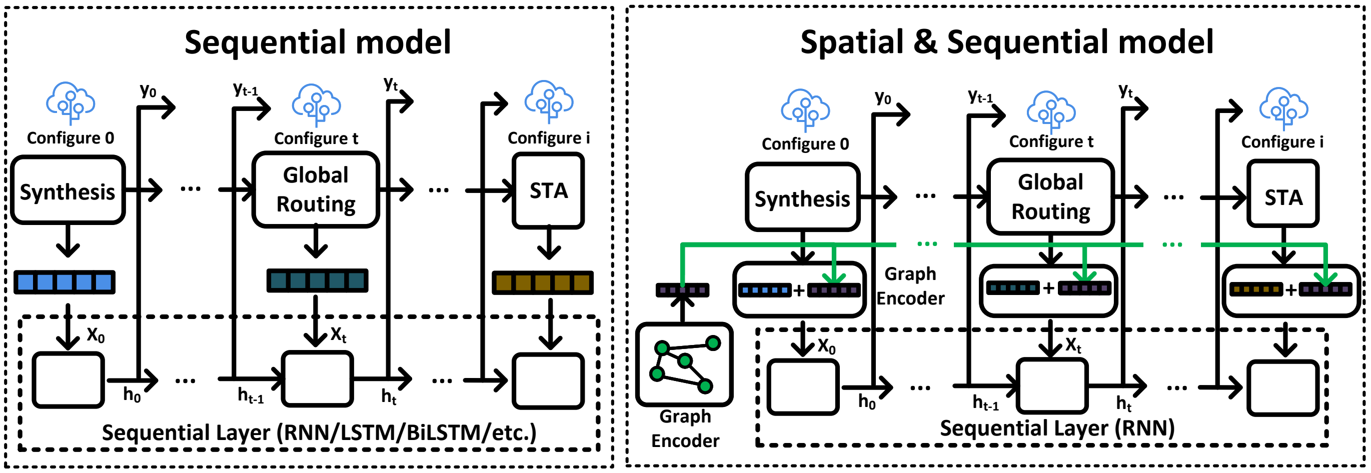


Fig. 4: On the left is the Sequential model predictor. On the right is the Spatial & Sequential model, where a Graph Encoder (Fig. 5) is incorporated to extract spatial information.

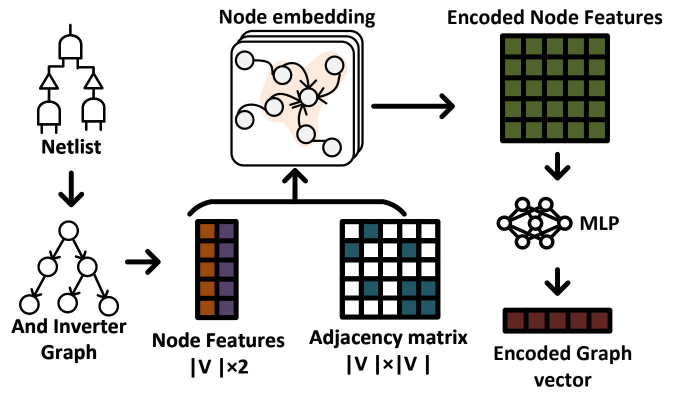


Fig. 5: Graph Encoder used in Fig.4, encoding design spatial feature information into vectors.

Here, W and b denote the weights and biases, $S(x)$ represents the sigmoid activation function, and H_{t-1} refers to the output from the previous time step. The output H is often termed as the hidden state, obtained from the cell state C. The symbol $\odot$ denotes element-wise multiplication. A Bidirectional LSTM (BiLSTM) is a sequence processing model comprising two LSTMs: one processes the input in a forward direction, and the other in a backward direction. BiLSTMs enhance the network's information by capturing both preceding and succeeding context, thereby improving algorithmic understanding.

2) Spatial Encoder: While the sequential model is effective for predicting subsequent behavior, especially when the design information is invisible in the elastic compute model, the EDA workload also depends heavily on design characteristics, such as the scale of the design and the difficulty of meeting the design target. In the cluster computing mode, where IP and design information are visible to users, it is essential to incorporate the design information as the spatial dimension. A Graph Encoder is used to encode design spatial information into vectors as shown in Fig. 5. First, the Register Transfer Level (RTL) netlist is transformed into And-Inverter Graph (AIG). This AIG graph is stored in Node Features and an Adjacency matrix, where Node Features include the in-degrees and out-degrees of each node. A Graph Neural Network (GNN) layer processes the initial Node Features into Node Embedding, outputting high-dimensional Encoded Node Features. Finally, Multilayer Perceptron (MLP) layers generate a low-dimensional Encoded Graph vector.

The extendibility of the trained model to new input designs relies on the creation of an AIG for each new netlist. This process is necessary to generate the spatial encoding required for accurate predictions. Although the AIG creation is a prerequisite, the trained model can generalize to new designs, utilizing the encoded features to make predictions without retraining. This approach ensures the model's applicability across different designs and EDA tools.

3) Spatial & Sequential Model: With the spatial information available from the design, we develop the Spatial & Sequential model, illustrated on the right of Fig. 4. Compared to the sequential model, Spatial & Sequential model concatenates the input sequence X_t at each time step t with spatial information provided by the Encoded Graph vector output from Graph Encoder. The RNN is chosen for the sequential layers to predict the next stage task scale. By combining the GNN-based Graph Encoder with the RNN-based sequential model, spatial and time-series information in the EDA flow are comprehensively encoded in our proposed Spatial & Sequential Model.

IV. Evaluation Results

In this section, we will explain the setup, datasets, baselines, and results of our experiments. For fair comparison, we evaluate the predictor models using the open-source dataset from the state of the art (SOTA) work [3]. This dataset is referred to as **Dataset 1**. In addition, for simulating real EDA tasks in cloud scenarios, we assess the efficiency of our framework across 12 open-source designs in the 45nm technology node [20] with the open-source design flow based on OpenRoad [9]. This dataset is referred to as **Dataset 2**. Learning-based prediction

TABLE II: Prediction accuracy results of baseline models, sequential models (RNN, LSTM, BiLSTM), and the spatial & sequential model by testing with **Dataset 1**. This corresponds to the cluster computing mode.

σ	**Model**	**MAE Ratio**	**Spearman**	**Kendall**	**Pearson**
0.1	XGBoost	0.10	0.97	**0.88**	0.98
	MLP	0.33	0.97	0.87	0.99
	Sequential (RNN)	0.08	0.97	0.87	0.99
	Sequential (LSTM)	0.09	0.97	0.87	0.99
	Sequential (BiLSTM)	0.09	0.97	0.87	0.99
	Spatial & Sequential	**0.07**	0.97	0.86	**0.99**
0.15	XGBoost	0.15	0.91	0.77	0.96
	MLP	0.39	0.93	0.81	0.96
	Sequential (RNN)	0.15	0.93	0.81	0.96
	Sequential (LSTM)	0.16	0.93	0.81	0.95
	Sequential (BiLSTM)	0.17	0.93	0.81	0.95
	Spatial & Sequential	**0.04**	**0.99**	**0.93**	**1.00**
0.2	XGBoost	0.18	0.87	0.70	0.93
	MLP	0.43	0.91	0.75	0.93
	Sequential (RNN)	0.18	0.91	0.76	0.94
	Sequential (LSTM)	0.2	0.91	0.76	0.93
	Sequential (BiLSTM)	0.2	0.91	0.76	0.93
	Spatial & Sequential	**0.03**	**0.98**	**0.93**	**1.00**

models are trained and tested on a Linux server machine equipped with a 3.7GHz AMD 5900X CPU and 128GB RAM, utilizing PyTorch and the Deep Graph Library (DGL) environment. Prediction accuracy is evaluated using four metrics: Mean Absolute Error Ratio (MAE Ratio), Spearman, Kendall, and Pearson correlation coefficients. MAE Ratio quantifies the mean of absolute errors between predicted and actual values, normalized by the actual values and averaged across n designs (in Equation 1). Sequential models, including RNN, LSTM, and BiLSTM, are tested in our proposed sequential model predictor. Additionally, Spatial & Sequential model predictors utilize RNN and GNN graph encoders. Baseline models, such as Multilayer Perceptron (MLP) and XGBoost, are also included for comparison. Furthermore, to validate the efficiency improvement of our framework in real EDA-on-cloud scenarios, we consider three baseline scenarios, which we discuss in the following subsections.

$$MAE\ Ratio = \frac{\sum_{i=1}^{n} |predicted_i - actual_i| / actual_i}{n} \quad (1)$$

A. Prediction Accuracy

To make fair comparison, we evaluate the accuracy of the task scale predictor in our framework using the open-source dataset from previous work [3] [21] , which utilizes the GF 14nm technology node and major commercial EDA flows from the EPFL benchmark suite and OpenCores. This dataset (**Dataset 1**) includes EDA task runtimes with various CPU configurations (from single-core to 8-core). Based on this dataset available in [21], we train our proposed models on 184 known designs and test them on 46 unseen designs. To address the small scale of this dataset and prevent overfitting, we add Gaussian distribution noise $X \sim \mathcal{N}(\mu, \sigma^2)$. The noise coefficient σ is set as 0.1, 0.15 and 0.2 in our experiment. Table II presents the prediction accuracy results of various models for predicting a routing stage runtime based on prior stages. Each data point corresponds to an averaged value across all 46 unseen designs. The scatter plots in Fig. 6 show the correlation between predicted and actual data, with the X-axis representing actual data and the Y-axis representing predicted data, both in seconds (s). The blue straight line indicates perfect prediction accuracy. The results demonstrate that the Spatial & Sequential model achieves the best prediction accuracy across most metrics and noise coefficients (σ). The Sequential models perform second-best, underscoring the importance of considering the sequential behaviors of the design flow.

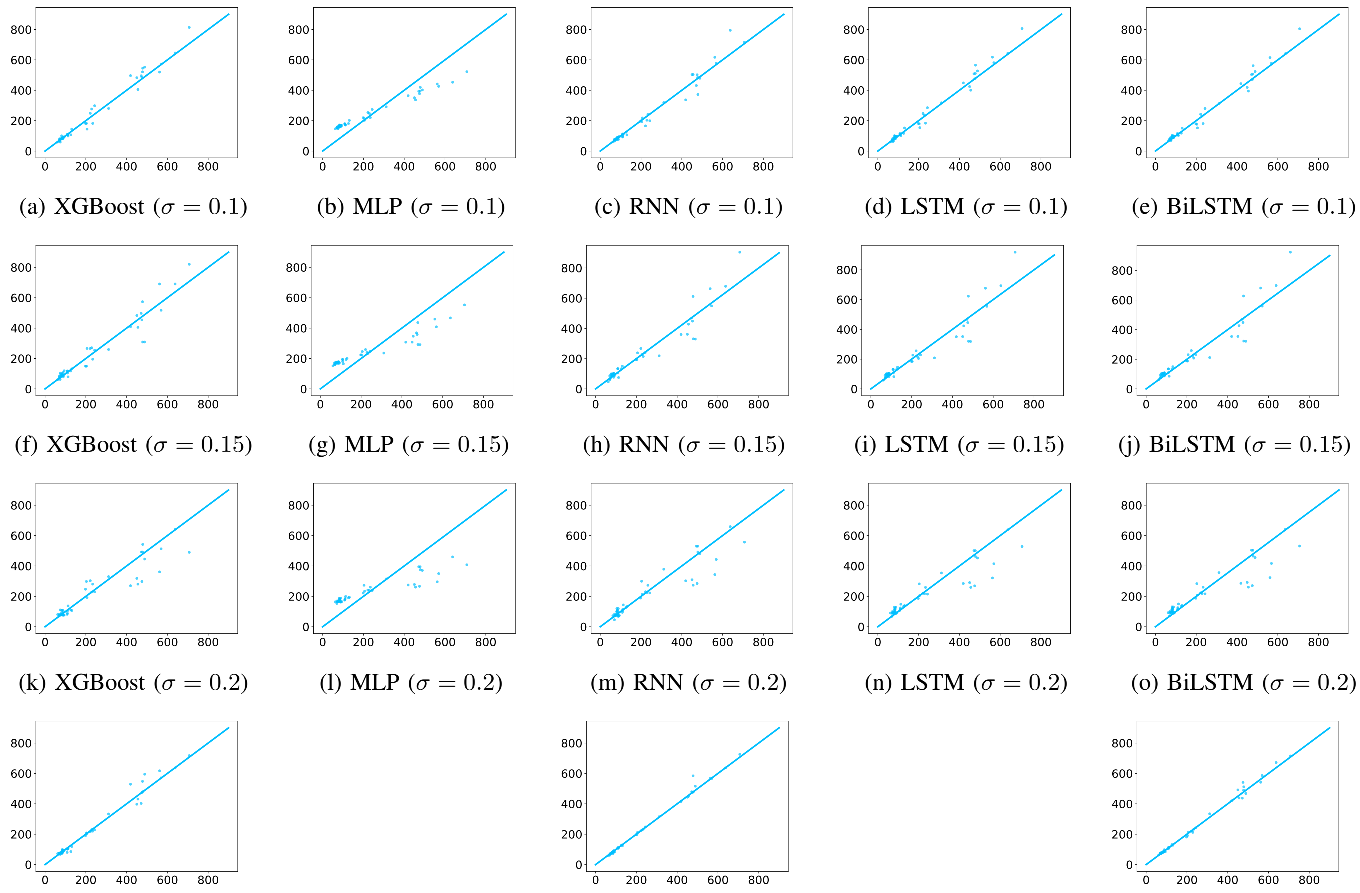
By testing with the same **Dataset 1**, we compare the prediction accuracy of the routing stage in Table III against the reported data from the closest SOTA work [3], which employs GCN for resource prediction. Only the MAE Ratio has been reported in the previous work. Our proposed model significantly outperforms the SOTA, showing over a 62% reduction in the MAE Ratio.

TABLE III: Prediction accuracy compared to SOTA work based on GCN. Our Spatial & Sequential model shows a 62% reduction in MAE ratio on the same dataset **(Dataset 1)**. Spearman, Kendall, and Pearson results for the SOTA work were not reported in their paper. This corresponds to the cluster computing mode.

		MAE Ratio	**Spearman**	**Kendall**	**Pearson**
GCN [3] (SOTA)		0.13	-	-	-
Spatial & Sequential	σ=0.1	**0.07**	0.97	0.86	0.99
	σ=0.15	**0.04**	0.99	0.93	1.00
	σ=0.2	**0.03**	0.98	0.93	1.00
	Mean	**0.05**	0.98	0.91	1.00

TABLE IV: Statistics of our generated EDA task design test case for creating **Dataset 2** in efficiency evaluation experiment.

Design Name	**Design Information**		
	#cell	**#wire**	**#ff**
`AES`	16758	17172	530
`ibex`	15700	18226	1931
`dynamic node`	13445	13095	2303
`APU`	3147	8027	390
`BM64`	9641	22995	1287
`picorv32`	9596	19856	1702
`PPU`	11035	22668	2895
`salsa20`	22012	47213	3677
`usb2p0 core`	780	2139	65
`wbqspiflash`	2008	5790	280
`xtea`	2018	5047	179
`yhuff`	11901	27091	2351



(a) XGBoost ($\sigma = 0.1$) (b) MLP ($\sigma = 0.1$) (c) RNN ($\sigma = 0.1$) (d) LSTM ($\sigma = 0.1$) (e) BiLSTM ($\sigma = 0.1$)

(f) XGBoost ($\sigma = 0.15$) (g) MLP ($\sigma = 0.15$) (h) RNN ($\sigma = 0.15$) (i) LSTM ($\sigma = 0.15$) (j) BiLSTM ($\sigma = 0.15$)

(k) XGBoost ($\sigma = 0.2$) (l) MLP ($\sigma = 0.2$) (m) RNN ($\sigma = 0.2$) (n) LSTM ($\sigma = 0.2$) (o) BiLSTM ($\sigma = 0.2$)

(p) S&S ($\sigma = 0.1$) (q) S&S ($\sigma = 0.15$) (r) S&S ($\sigma = 0.2$)

Fig. 6: Prediction accuracy scatter plot by testing with **Dataset 1**: S&S represents Spatial & Sequential model. In each subfigure, the X-axis shows actual data and the Y-axis shows predicted data (unit: seconds).

B. *Efficiency Improvement*

Since the dataset in [3] only contains small-scale designs and does not consider the variations in runtime for individual EDA steps, it limits the evaluation of efficiency improvement under realistic EDA-on-Cloud scenarios. In this paper, we have created a new dataset (**Dataset 2**) based on 12 representative benchmarks of varying sizes and structures from OpenCores, as detailed in Table IV. The EDA flow is finely segmented into 8 stages, utilizing open-source EAD tools and the 45nm technology node. To minimize unnecessary runtime, all optimization efforts in the flow have been set to low, while timing-driven optimizations have been enabled where applicable. Other tuning parameters are maintained at their default values. To evaluate the efficiency of AIG graph construction, we collected runtime data for creating AIG graphs for four representative designs from Dataset 2. Each generation was repeated three times to account for variability, with the results presented in Table V. The data shows that the AIG graph construction process is relatively fast and straightforward, utilizing the open-source logic synthesis tool Yosys, which automates the generation of AIG graphs.

TABLE V: The runtime for AIG graph construction was measured across four representative designs, with each generation repeated three times (corresponding to Test #).

	Runtime (s)			
Design	**Test 1**	**Test 2**	**Test 3**	**Mean**
`AES`	5.17	5.18	5.06	5.14
`APU`	7.46	7.29	7.58	7.44
`BM64`	8.55	8.20	8.24	8.33
`xtea`	2.76	3.11	2.76	2.88

Consequently, the runtime for AIG graph construction can be considered negligible compared to the actual EDA tasks.

In cluster computing mode, efficiency is defined as the total runtime required to complete the entire EDA flow for a specific design under a user's management. In comparison, we simulate scenarios where configurations are randomly chosen at each stage, referred to as Baseline1 in Fig. 7. Additionally, we select the "best" configuration (with the highest number of CPU cores) at each stage, denoted as Baseline2. For instance,

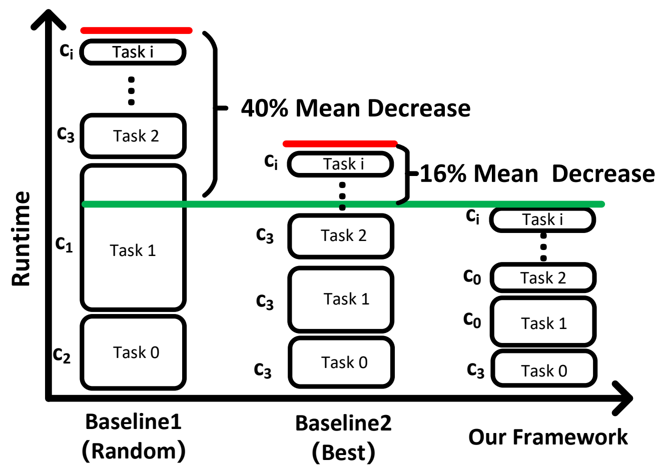


Fig. 7: Illustration of cluster computing mode efficiency improvement. c_i represents different CPU configurations.

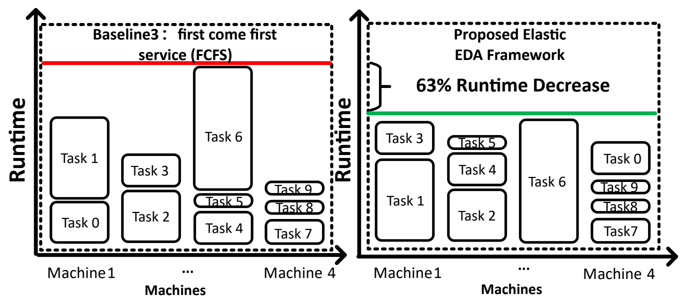


Fig. 8: Illustration of efficiency improvement in elastic computing mode.

tasks from 0 to i are assigned configurations c_0 to c_i for each policy. Our framework optimizes task distribution across stages by greedily selecting configurations with the shortest predicted runtime, thus achieving a reduced overall runtime for completing the entire EDA flow.

In Elastic Computing mode, efficiency is defined as the total runtime required to complete a batch of submitted tasks across machines with varying configurations. Leveraging task scale prediction, our framework allocates tasks evenly across machines based on the mean workload. For comparison, a first-come-first-served (FCFS) approach is implemented, denoted as Baseline3. For example, as illustrated in Fig. 8, when 10 tasks (task0 to task9) are simultaneously submitted by users and four machines with different configurations are available, FCFS is employed on the left, where tasks are managed in the order of submission. However, with task runtime prediction across different configurations, our framework evenly distributes tasks, resulting in a reduced total runtime for completing all submitted tasks.

The detailed efficiency improvement results are summarized in Table VI. We compare our framework in cluster computing mode and elastic computing mode with three baselines, each defined differently as mentioned above. This has been evaluated in all 12 designs. In each design, improvement is

TABLE VI: Demonstrating efficiency improvement with **Dataset 2** through comparison with three baselines defined in Fig. 7 and 8.

	Cluster Compute Mode		Elastic Compute Mode
Designs	**Proposed vs. Baseline1**	**Proposed vs. Baseline2**	**Proposed vs. Baseline3**
`AES`	51%	4%	63%
`picorv32`	43%	19%	61%
`APU`	13%	4%	51%
`BM64`	44%	18%	64%
`dynamic node`	39%	1%	61%
`ibex`	55%	39%	62%
`PPU`	80%	87%	91%
`salsa20`	40%	4%	63%
`usb2p0 core`	27%	3%	62%
`wbqspiflash`	34%	1%	60%
`xtea`	26%	3%	60%
`yhuff`	31%	3%	63%
Mean	**40%**	**16%**	**63%**

calculated by simulating 100 times and obtaining the average result. Baseline1 involves randomly choosing configuration when users submit tasks, and Baseline2 selects the best configuration. Baseline3 follows a first-come-first-served (FCFS) approach. The results demonstrate that our proposed framework achieves a 40% and 16% mean runtime decrease compared to Baseline1 and Baseline2 in cluster computing mode respectively, and a 63% mean runtime decrease compared to Baseline3 in elastic computing mode. While the current comparison uses simpler baseline scheduling strategies, future work will involve benchmarking against more sophisticated scheduling algorithms to better highlight the efficiency of our proposed method.

V. Conclusions and Future Work

In this paper, we propose Elastic EDA, a framework that leverages learning-based approaches to accurately gauge the characteristics of EDA jobs in a cloud environment. The proposed framework develops prediction models tailored for two existing usage modes: elastic compute mode and cluster compute mode, utilizing the available sequential and/or temporal information in a design environment. Through evaluations with multiple datasets and designs, the proposed framework outperforms the state-of-the-art prediction model in terms of MAE ratio reduction, achieving over a 62% improvement. Additionally, the proposed framework demonstrates significant efficiency improvements compared to several baseline scenarios that mirror typical EDA-on-Cloud behavior. This work serves as one of the first attempts to overcome the cost and efficiency barriers that hinder EDA-on-cloud deployment. We are in the process of open-sourcing the framework as an ongoing effort. For future work, we aim to test the framework under various cloud environments and design flows.

References

[1] L. Stok, "The next 25 years in eda: A cloudy future?" *IEEE Design & Test of Computers*, vol. 31, no. 02, pp. 40–46, 2014.

[2] N. Sehgal, J. M. Acken, and S. Sohoni, "Is the eda industry ready for cloud computing?" *IETE Technical Review*, vol. 33, no. 4, pp. 345–356, 2016.

[3] A. Hosny and S. Reda, "Characterizing and Optimizing EDA Flows for the Cloud," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 3040–3051, 2022.

[4] Gao, Jiechao and Wang, Haoyu and Shen, Haiying, "Machine learning based workload prediction in cloud computing," in *2020 29th international conference on computer communications and networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[5] Di, Sheng and Kondo, Derrick and Cirne, Walfredo, "Host load prediction in a Google compute cloud with a Bayesian model," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[6] Zhong, Wei and Zhuang, Yi and Sun, Jian and Gu, Jingjing, "A load prediction model for cloud computing using PSO-based weighted wavelet support vector machine," *Applied Intelligence*, vol. 48, pp. 4072–4083, 2018.

[7] Masdari, Mohammad and Khoshnevis, Afsane, "A survey and classification of the workload forecasting methods in cloud computing," *Cluster Computing*, vol. 23, no. 4, pp. 2399–2424, 2020.

[8] A. B. Kahng and T. Spyrou, "The openroad project: Unleashing hardware innovation," in *Proc. GOMAC*, 2021.

[9] "Openroad-flow-script," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts, 2023.

[10] Ahmed Elzeftawi, "Scaling eda workloads using scale-out computing on aws," https://aws.amazon.com/blogs/industries/scaling-eda-workloads-using-scale-out-computing-on-aws/, 2020.

[11] Sashi,Obilisetty and Mark,Mims, "Scale your eda flows: How google cloud enables faster verification," https://cloud.google.com/blog/products/compute/scale-up-your-eda-flows-on-google-cloud, 2020.

[12] Synopsys, "https://www.synopsys.com/cloud.html," https://www.synopsys.com/cloud.html.

[13] Cadence, "Introduction to cadence cloud," https://www.cadence.com/en_US/home/solutions/cadence-cloud.html.

[14] C. Man, Z. Shi, Z. Xu, Y. Zong, K. Pang, and Y. Li, "Cloud-eda: a paas platform architecture and application development for ic design & test," in *Proceedings of 2014 International Conference on Cloud Computing and Internet of Things*. IEEE, 2014, pp. 1–4.

[15] J. Liu, Z. Lu, X. Xie, J. Wang, and J. Lai, "An exponential dynamic weighted fair queuing algorithm for task scheduling in chip verification platform," in *2019 IEEE 13th International Conference on ASIC (ASICON)*. IEEE, 2019, pp. 1–4.

[16] H. Ranjan, "Cloud computing and eda: Is cloud technology ready for verification…(and is verification ready for cloud)?" in *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*. IEEE, 2011, pp. 1–2.

[17] T.-M. Tseng, M. Li, Y. Zhang, T.-Y. Ho, and U. Schlichtmann, "Cloud columba: Accessible design automation platform for production and inspiration," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–6.

[18] H. Badier, J. C. L. Lann, P. Coussy, and G. Gogniat, "Transient Key-based Obfuscation for HLS in an Untrusted Cloud Environment," *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*, pp. 1118–1123, 2019.

[19] N. K. Sehgal, P. C. P. Bhatt, J. M. Acken, N. K. Sehgal, P. C. P. Bhatt, and J. M. Acken, "Migrating a complex industry to cloud," *Cloud Computing with Security: Concepts and Practices*, pp. 155–172, 2020.

[20] "Nangate 45nm library," http://www.nangate.com/, 2006.

[21] "Characterizing and Optimizing EDA Flows for the Cloud," https://github.com/scale-lab/EDAonCloud, 2021.