Journal of Rare Cardiovascular Diseases
ISSN: 2299-3711 (Print) | e-ISSN: 2300-5505 (Online)
www.jrcd.eu
RESEARCH ARTICLE

Enhancing VLSI Time Driven Placement Parameters via Deep Reinforcement Learning

Bindushree V[1], Jayagowri R[2], Roohum Jegan[3], Sree Ranjani Rajendran[4], Vinolya S[5] and Y N Gowthami[6]

[1]Department of Electronics and communication Engg BMS college of Engineering Bengaluru, India
[2]Department of Electronics and communication Engg BMS college of Engineering Bengaluru, India
[3]Department of CSE(AIML) Saraswati College of Engineering, Kharghar, Navi Mumbai
[4]Electrical Engineering and Computer Science Department, Florida Atlantic University Boca Raton, FL 33431
[5]Department of Electronics and communication Engg BMS college of Engineering Bengaluru, India
[6]Department of Electronics and communication Engg BMS college of Engineering Bengaluru, India

*Corresponding Author
Dr. Lakshmi Arun

Article History
Received: 21.09.2025
Revised: 30.09.2025
Accepted: 22.10.2025
Published: 11.11.2025

Abstract: The rapid advancement of Very Large-Scale Integration (VLSI) technology has necessitated innovative approaches to optimize chip design processes, particularly in time-driven placement. This paper presents a novel methodology using Deep Reinforcement Learning (DRL) to enhance VLSI placement parameters, aiming to minimize wirelength and improve overall design efficiency. Traditional placement techniques often rely on manual tuning, which is time-consuming and suboptimal. By integrating reinforcement learning with deep neural networks, our approach automates parameter optimization, reducing human intervention and accelerating design convergence. The proposed framework employs an actor-critic architecture, combining policy-based and value-based reinforcement learning to dynamically adjust placement parameters. Key components include a state space encapsulating netlist features and placement parameters, a streamlined action set for parameter adjustments, and a normalized reward function based on Half-Perimeter Wire Length (HPWL). The model utilizes Long Short-Term Memory (LSTM) and attention mechanisms to handle complex dependencies and recurrent optimization processes. Experimental results on benchmark designs demonstrate significant improvements, with our DRL agent achieving up to an 8.7% reduction in HPWL compared to human baselines and outperforming existing methods like Multi-Armed Bandit (MAB) and traditional RL models. The approach also generalizes well to unseen netlists, showcasing its robustness.

Keywords: Placement Optimization, Reinforcement Learning, Deep Wirelength Minimization, FPGA.

INTRODUCTION

The remarkable growth of the electronics industry in both complexity and annual device production can be attributed to various advancements in the field of VLSI. The evolution of the chip design industry has spurred substantial progress in areas such as telecommunications, control systems, consumer electronics, high-performance computing, missile technology, and more. These applications experience processing speeds and application access that are unheard of, and VLSI makes this all feasible. As long as there are inventions and a very fast rate of growth in the VLSI sector to support those inventions, there will always be a market for these products. Because of the substantial number of cells and the precision needed for their placement in Very Large-Scale Integration (VLSI) chips, the idea of traditional design that is done manually is rendered obsolete. Any designer would find it extremely difficult to complete a project of this size without the aid of technology.

Electronic Design Automation (EDA) tools were introduced as a result, assisting designers in increasing design and verification efficiency. The development of several tools for each level of VLSI is the main goal of EDA. However, having EDA tools alone is insufficient for design because it necessitates a fundamental understanding of VLSI and its characteristics. This may aid in bridging the gap between specification and chip production. By defining a problem and an algorithm for the same problem, designers should possess the capability to develop a computer program that automates physical design. DRL is a sophisticated framework that unites two potent concepts: reinforcement learning and deep learning. This fusion equips AI agents with the capability to comprehend and navigate complex environments, rendering it highly effective for tackling intricate real-world problems.

In the realm of reinforcement learning, agents acquire decision making abilities by engaging with an environment. They take actions, observe the consequences, and refine their strategies based on the received rewards or penalties. This iterative process of learning through experimentation ultimately leads to the optimization of the agent's behavior. Conversely, deep learning harnesses neural networks to handle intricate patterns and representations within data. These networks are structured into multiple layers, each progressively refining features extracted from the input data. This hierarchical architecture empowers deep learning models to capture and understand intricate relationships inherent in the data. Within the domain of DRL, deep learning is employed to enhance

traditional reinforcement learning methodologies. By seamlessly integrating DNN into the framework, DRL enables agents to learn directly from raw sensory input, such as images or auditory signals. This advancement discards the need for manual feature engineering, as the neural network autonomously learns to extract pertinent information from the data. The convergence of reinforcement learning and deep learning within DRL yields impressive outcomes.

Agents are proficient in handling high-dimensional input spaces, allowing them to excel in tasks like image recognition and robotics. Nevertheless, it's important to acknowledge that DRL also presents challenges, including training instability and sample inefficiency, which demand thoughtful consideration and specialized techniques for resolution. In summation, deep reinforcement learning capitalizes on the strengths of RL and deep learning, empowering AI agents to acquire complex behaviors from raw data. This approach holds immense promise across a diverse array of applications, paving the way for substantial advancements in AI research and practical problem-solving.

This paper is organized as follows. Section-II presents the prior work on implementation of Machine learning algorithm in VLSI placement. Section-III presents the problem description, Section-IV presents the proposed methodology, Section-V presents results analysis and section-V concludes the proposed technique.

LITERATURE REVIEW

An immense amount of research has been done regarding the implementation of ML algorithms in the VLSI placement. This review takes into account various ML models, Reinforcement learning techniques, deep RL techniques and algorithms implemented to boost the performance of placement tools, optimize, and effortlessly automate the placement process in the PD flow. According to Z. Wang et al. [1] reward and state transition functions of dynamic settings may change based on time, which is why this work addresses the incremental RL problem in continuous spaces for these environments. The aim was to switch from the initially learned policy in the original environment to a new one whenever the environment changes. With the incremental learning process, authors present a two-step strategy to increase adaptability: policy relaxation and importance weighting.

A proper exploration of the new environment is the first goal of the policy relaxation mechanism, which achieves this by lowering the behavior expectations for a few learning episodes to a consistent level. This results in a better long-term adaptation by reducing the conflict between the new knowledge and the previously held beliefs they're adapted to. The second step is the implementation of an important weighting technique based on finding that episodes with greater returns are more agreeing with the new environment and therefore contain more novel information. In order to encourage the prior optimal policy to be quickly replaced by a new one that works in the new environment, they provide larger weights during parameter update to episodes that contain more new information. Traditional navigation challenges and intricate locomotion tasks with various configurations were the subjects of experiments. The outcomes demonstrated that the suggested approach could manage a variety of dynamic situations and deliver a substantially faster learning process.

A. Agnesina et al. [2] the physical design flow depends on the placement's quality. A human engineer often devotes a significant amount of time to fine-tuning the various settings of commercial places to meet PPA goals. To enhance placement settings of commercial EDA tools, this study suggests a deep reinforcement learning (RL) architecture. Researchers create an autonomous agent that is taught exclusively by RL via self-search and learns to tune parameters optimally without the assistance of humans or domain expertise. Researchers combine manually created characteristics from graph topology theory with graph embeddings produced by unsupervised Graph Neural Networks to generalize to unseen netlists. The sparsity of the data and the latency of placement runs are overcome by their RL algorithms. When compared to a human engineer and a state-of-the-art tool auto-tuner, their trained RL agent improves wirelength on unseen netlists by up to 11% and 2.5%, respectively, in just one placement iteration (20X and 50Xless iteration).

A. Mansoor et al. [3] have implemented a unique placement method (RS3DPlace) based on simulated Annealing (SA) and Reinforcement Learning (RL), which is the earlier machine learning strategy for Monolithic 3D ICs (M3D). RS3D Place rapidly calculates a draft solution using RL's capacity for learning, which SA then uses to produce a better final solution. Although the gate-level M3D design style is the focus of the present implementation, it may be applied to other M3D design styles as well as other 2D and 3D physical design optimization issues. We evaluated RS3DPlace for 8- 128-bit MUX-based right arithmetic shifter circuits and a circuit with non-regular connections in comparison to Mux-based shifters, which are optimized in 2- layered M3D technology, to demonstrate the efficiency of the technique. Additionally, according to experimental findings, the total cost function is on average 16% better than it is with Random Initialized SA (Rand SA).

Mrinal Mathur [4] demonstrates that solving time-consuming placement-based activities requires focusing on complicated, industry-wide problems with a big impact. They provide a fresh RL-based method for placing the macros quickly and more effectively to maximize PPA values. Designers demonstrate that they produce placements with improved outcomes and outperformed state-of-the-art baselines. These findings demonstrate that their agent reduced wirelength without incurring any additional training costs and generalized well when compared to EDA technologies.

S. F. Almeida et al. [5] the placement engine may generate an impractical routing solution as a result of the search for wirelength optimization, necessitating the repetition of earlier processes and raising the total project cost. Due to its cheap computing cost, placement algorithms have historically used pin density to determine routability. This has turned out to be inefficient at advanced technology nodes, nevertheless, because of tighter production regulations and complicated standard cell layouts. Although routability is a topic that many placement strategies aim to solve, the issue is that these models rely on certain heuristics or designer expertise. As a result, researchers provide a methodology based on machine learning for addressing routability during the placement stage. The different machine learning models used in the placement process is reviewed and presented the analysis in [24]. On the basis of literature review the objective and problem statement is defined in this work.

Problem Statement and Objectives

This work focuses on the idea of timing driven placement (TDP). Since placement plays a very important role in the physical design flow it is significant for the Physical implementation which speeds up design turnaround time in the post CTS and Routing. The majority of existing works simply concern themselves with creating the standard cells from scratch in a blank floor plan. However, placement extends beyond it where incremental placement being quite important. There is room for improvement at this stage because there are less earlier works. To reduce the need for manual intervention and determine the processor's runtime based on the number of standard cells, this research focuses on reinforcement learning. Reinforcement learning is a branch of machine learning that addresses how intelligent agents should make decisions within an environment to maximize cumulative rewards through a combination of exploration and exploitation strategy

METHODOLOGY AND IMPLEMENTATION DETAILS

Proposed Methodology

The subsequent placement phase, which legalizes the altered placement, must adhere to two essential criteria. Firstly, the placement process should commence from the perturbed placement, including the tentative positions for the newly introduced gates. Initiating the placement process from scratch when dealing with the new netlist would likely result in a lack of convergence between the netlist transformation and placement procedure. Secondly, the modifications made to the perturbed placement should not be overly minimal. Transforming the perturbed placement into a legal one with minimal alterations would render the subsequent netlist transformation phase ineffective.

To address the first requirement, ECO placement techniques are typically employed. However, it's worth noting that ECO placement techniques do not satisfy the second requirement. In the following section, it is explored that a placement improvement procedure fulfills both of these criteria. Both theoretical and experimental evidence indicates that the linear assignment method, when combined with appropriate net models, can be effectively used to determine high quality placements concerning both area and wire length. An important problem in systems and time driven is placement optimization, which refers to the problem of mapping the nodes of a graph onto a limited set of resources to optimize for an objective.

4.1.2 Block Diagram

The block diagram consists of two main components: "Deep Reinforcement Learning" and "Error Signal Generator for Deep Reinforcement Learning." As shown in Figure 1, deep RL block embodies a system or process integral to deep reinforcement learning. Deep reinforcement learning amalgamates reinforcement learning, characterized by learning through experimentation, with deep learning, which employs neural networks to manage intricate patterns. Error signal generator block seems to generate an error signal, possibly to facilitate learning or adjustments in the deep reinforcement learning process.

RL Environment

Overview

We have developed a reinforcement learning (RL) agent aimed at autonomously optimizing the parameter

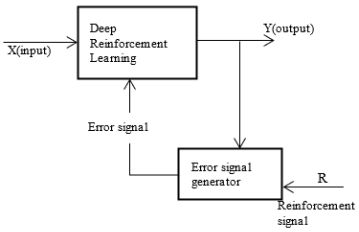


Fig .1. Block Diagram of proposed method.

Configurations of a placement tool. The primary goal of this agent is to minimize wirelength. The RL problem we address comprises four essential components:

1. States: Our state space encompasses all netlists existing within the environment and the complete spectrum of feasible parameter setting combinations (denoted as P) available through the placement tool, such as tools like Cadence Innovus or Synopsys ICC2. A singular state, represented as 𝑠, encapsulates a unique netlist and its corresponding current parameter configuration.
2. Actions: The agent has access to a set of actions that it can employ to manipulate the current parameter settings. Each action, denoted as 𝑎, brings about changes in a subset of parameters.
3. State Transition: Given a particular state (𝑠𝑡), and in response to an action, the subsequent state (𝑠𝑡+1) emerges. This progression involves the same netlist while incorporating updated parameter values in line with the action undertaken.
4. Reward: The reward mechanism we utilize pertains to the negative of the "Half-Perimeter Wire Length" (HPWL) output derived from a commercial Electronic Design Automation (EDA) placement tool. The reward value experiences an increment if the undertaken action leads to an enhancement in parameter settings, specifically geared toward minimizing wirelength. Our approach entails the construction of an RL agent proficient in adjusting parameter settings within a placement tool autonomously. This endeavor is driven by the objective of reducing wirelength in netlists. The problem's core components include defining states involving netlists and parameters, actions influencing parameter alterations, state transitions based on actions, and a reward structure grounded in the reduction of wirelength through improved parameter adjustments. Depicted in Figure 2, the realm of reinforcement learning (RL) involves the agent's acquisition of knowledge by engaging with its surroundings, unfolding across discrete time steps. At each distinct time step denoted as 𝑡, the agent.

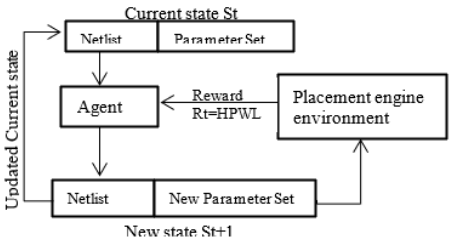


Fig.2. Interaction between the reinforcement learning agent and environment in the suggested approach.

within set A. The selection process aligns with the agent's policy 𝜋, functioning as a mapping mechanism steering states towards actions. In return for its action, the agent is provided with a reward signal expressed as 𝑟𝑡, concurrently transitioning to the subsequent state, S𝑡+1. This cyclic progression persists until the agent ultimately arrives at a terminal state, marking a conclusion. Subsequently, the cycle reinitiates as the agent embarks on a new learning journey. In essence, this portrayal delineates the fundamental mechanics of RL comprehending how agents engage with environments, make choices based on policies, accrue rewards, and advance through states, ultimately shaping their learning process within a cyclic framework.

RL Settings

Objective: Given a netlist, determine arg min𝑝 ∈P 𝑟𝑢𝑛𝑡𝑖𝑚𝑒 (𝑝), where P represents the complete set of parameter combinations, and 𝑟𝑢𝑛𝑡𝑖𝑚𝑒 is obtained from the tool's output. Approach:

(1) Define the environment as a black-box placement tool.

(2) Define a state 𝑠 that approximates the current parameter set 𝑝𝑎𝑟𝑎𝑚𝑠 ∈ P and includes the target netlist.

(3) Specify actions for modifying 𝑝𝑎𝑟𝑎𝑚𝑠.

(4) Establish a reward □ proportional to the negative of □□□□, thereby encouraging the agent to minimize wirelength.
(5) Choose a discount factor □, ensuring that the agent aims to reduce wirelength within the fewest possible steps.

The States

We define our state by encompassing a set of twelve placement parameters originating from Cadence Innovus, essential for executing the current placement task (Table 1). Alongside these parameters, we integrate information metrics pertaining to the netlist undergoing placement. This netlist-related information incorporates a blend of metadata-based insights (such as cell count, floorplan area, etc.) and graph-based topological attributes (Table 2). Additionally, we incorporate unsupervised features extracted via a graph we incorporate unsupervised features extracted via a graph neural network.

The inclusion of netlist characteristics holds significance in facilitating knowledge transfer across highly diverse netlists, effectively enabling our agent to extrapolate its tuning strategy to previously unseen netlists. This adaptability is essential as the optimal policy likely hinges on the intricacies unique to each netlist. In a formal representation, our state is constructed through the concatenation of several components. These components encompass one-hot encoded categorical parameters (utilizing Booleans or enumerations), integer parameters, as well as both integer and floating-point netlist features. This comprehensive state formulation ensures that essential attributes from both the placement environment and the netlist information are captured, thereby empowering our agent to make informed decisions during the tuning process.

Name	Type	Groups	Objective	Value
Clock gate aware Uniform density	Bool Bool	Global Global	Indicates that placement process taken into consideration the presence of clock gate cells within the design Facilitates achieving a balanced distribution of cells	2 2
Eco max density Legalization gap Max density	Integer Integer Integer	Detail Detail Global	The upper limit for permissible distance during placement legalization The smallest allowable gap between instances on sites Regulates the upper limit of density within local bins.	[0,100] [0,100] [0,100]
Eco priority Activity power driven Wire length opt	Enum Enum Enum	Detail Detail + effort Detail + Effort	Priority assigned to instances for the refinement placement process Degree of exertion for the activity driven power placer Enhances wirelength optimization	3 3 3 3

Blockage channel	Enum	Global	through cell swapping Generates	2
Timing effort	Enum	Detail + Effort	obstructions within narrow channels between macros during placement.	3
Clock power driven	Enum	Detail + Effort	Degree of commitment for the timing-driven placement approach.	3
Congestion effort	Enum	Detail + Effort	Extent of engagement for the clock power-driven placement strategy.	
			The degree of dedication to alleviate congestion.	

TABLE.1. TARGETED PLACEMENT PARAMETER.

The actions

To circumvent an overly complex learning scenario involving 24 distinct actions and one for each placement parameter – we opted for a more streamlined approach. Our strategy involves categorizing tuning variables based on their nature (Boolean, Enumerate, Numeric) and their relevance to placement ("Global," "Detailed," "Effort") towards the upper limit. Similarly, for enumerates, actions like "down" represent transitioning from a higher level to a medium one. In addition, we introduced an action that preserves the current parameter settings without modification. This action acts as a trigger, enabling environment reset in situations where it's selected consecutively. This strategy yields a concise set of eleven diverse actions, as detailed in Table 3. Our aim in constructing the action space was to strike a balance between simplicity to facilitate neural network training and sufficient expressiveness to encompass the full range of parameter adjustments achievable through these transformations.

TOPOLOGICAL(10)		METADATA(10)	
Name	Type	Name	Type
Average degree	float	#cells	integer
Average fanout	float	#net	integer
Largest SCC	integer	#cell pins	integer
Max. clique	integer	#IO	integer
Chromatic nb	integer	#nets w.fanout €[5,10]	integer
Max logie level	integer	#nets w.fanout>=10	integer
RCC	float	#FFs	integer
CC	float	Total cell area (um^2)	integer
Fiedler value	float	#hardmacro	integer
Spectral radius	float	Macro area (um^2)	integer

TABLE.2. HANDCRAFTED NETLIST FEATURE .

1. Boolean Flipping
2. Increment Integers
3. Decrement Integers
4. Increase Effort Levels
5. Decrease Effort Levels
6. Raise Detailed Focus
7. Lower Detailed Focus
8. Heighten Global Focus (excluding booleans)
9. Reduce Global Focus (excluding booleans)
10. Mingle Inversions: Timing, Congestion, and WL Efforts
11. No Action Taken

Table.3. ACTIONS.

The Reward Structure

To ensure effective learning across a range of netlists exhibiting diverse wirelengths, adopting a reward directly linearly linked to Half-Perimeter Wire Length (HPWL) proves challenging. To enhance convergence in our approach, we opt for a normalized reward function. This function serves to equalize the magnitudes of value approximations across different netlists. The normalized reward function takes the form:

$$Rt = \frac{HPWL\ Human\ Baseline - HPWLt}{HPWL\ Human\ Baseline}$$

It's important to note that while formulating rewards in this manner relies on knowledge of □□□□ Human Baseline, representing the anticipated baseline wirelength for a design, obtaining this baseline merely necessitates a single placement conducted by an engineer.

f) Reinforcement Learning Placement Agent Leveraging the environment description elucidated in the preceding section, our approach revolves around training an agent to independently fine-tune the parameters of the placement tool. The following outlines our methodology:

• Within a specific state, the agent discerns the optimal action by relying on the probability outputs of its policy network.

• To ensure effective training of the policy network, we embrace an actor-critic framework, unifying the benefits of both value-based and policy-based optimization algorithms.

• To tackle the well-documented challenges of latency and sparsity encountered in the application of Reinforcement Learning (RL) to Electronic Design Automation (EDA), we introduce multiple simultaneous environments, allowing for the accumulation of diverse experiences. To facilitate the learning of a recursive optimization process characterized by intricate interdependencies, our agent's architectural design incorporates a deep neural network equipped with a recurrent layer and an attention mechanism.

This tailored architecture effectively addresses the intricacies of EDA optimization. The architecture proposed in [2] is modified by using ReLu activation function instead of tanh function and also by introducing the more dense layer to achieve the higher accuracy.

Learning of Actor – critic framework

In our chosen architectural framework, our objective is to develop a policy that optimizes value while simultaneously learning and incorporating knowledge from the environment into other predictions. This framework, known as the actor-critic approach, is illustrated in Figure.3. Within this context, the term "actor" refers to the policy's role in action selection, while the estimated value function is designated as the "critic" since it evaluates and provides assessments of the actions executed by the actor. Actor-critic algorithms integrate elements from both value-based and policy-based methods, providing a comprehensive approach to reinforcement learning. This approach is depicted in Figure 3.

A network architecture featuring two distinct components.

The actor-critic framework seamlessly integrates both policy and value models. The holistic agent network can be conceptualized as a deep neural network. In this arrangement, the policy component directs the adjustment of placement parameter settings, while the value component evaluates the effectiveness of the current configuration. Crucially, the network architecture is designed to facilitate the reciprocal flow of information between value and policy predictions, characterized by the integration of a Long Short Term Memory (LSTM) and an Attention

mechanism, facilitates the acquisition of knowledge pertaining to intricate recurrent optimization processes. Specifics regarding the sub-networks employed in this architecture are outlined in Table. 4 for reference. Our algorithm identifies the location of placement using LSTM model for forecasting algorithm for TDP. The wire length varies it varies the negative slacks.

The comprehensive design of our deep neural network is visually depicted in Figure.4. To compute both value and policy, we initially pass the combination of placement parameters and graph-extracted features through two feedforward fully-connected (FC) layers featuring ReLu activations because with our study we identified ReLU is often better suited than tanh for time-driven placement problems due to its faster learning, avoidance of vanishing gradients, efficiency, and ability to handle deep networks, which may need to handle both temporal and spatial data for the placement of components based on time constraints, These properties make ReLU a more effective activation function where large amounts of data must be processed quickly and accurately. So the network for agent proposed in fig.4 uses ReLU network. This is followed by another FC linear layer.

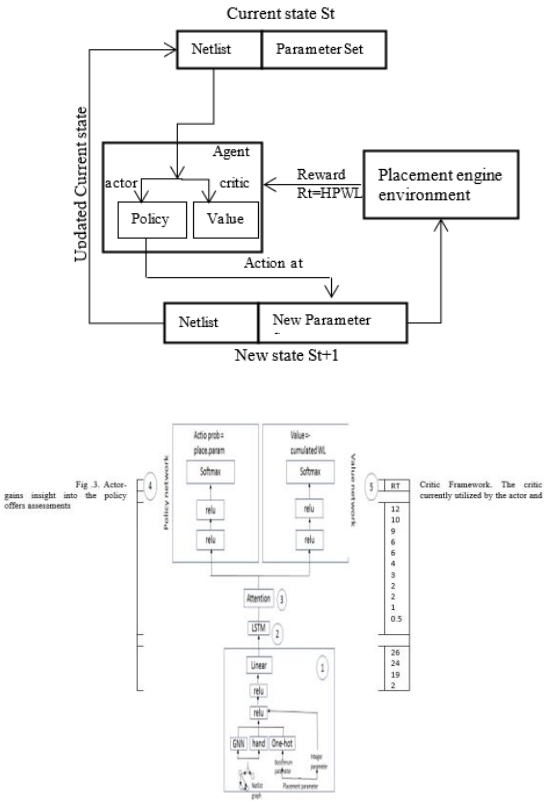


Fig.4. The comprehensive network structure of agent.

Subsequently, we introduce a Long ShortTerm Memory (LSTM) module equipped with layer normalization and have included sequential layer, convolution, activation max pooling layer dropout and flatten layer, housing 16 hidden standard units with forget gate functionality. Notably, the feed-forward FC layers do not possess any memory components. The incorporation of an LSTM, a recurrent layer, enables the model to make decisions based on prior states, aligning with traditional optimization methodologies rooted in recurrent approaches. Subsequently, this hidden state serves as input for the two branches of the network, each consisting of two fully-connected (FC) layers. One branch concludes with an output softmax layer for policy, while the other ends with an output linear layer for value estimation. Details regarding the network's parameters are provided in Table 4 for reference

RESULT

For the training and testing of our agent, we have chosen a set of 15 benchmark designs sourced from OpenCores, the ISPD 2012 contest, and two RISC-V single cores, as outlined in Table 4. The initial eleven designs are utilized for training purposes, while the remaining four serve as the test dataset.

Table.5. Benchmark statistics.

Part	Input	Hidden	Output

1.Shared body 2.LSTM(6 unroll) 3.Attention 4.policy 5.value	79 16 16x6 16 16	(64,32) (relu) 16 Wa, Wc (32,32) (relu) (32,16)(relu)	16(linear) 16x6 16 11(softma x) 1(linear)

Table. 4 is the Benchmark Characteristics Derived from a Commercial 28nm Technology. Here, LL signifies the maximum logic level, RCC represents the Rich Club Coefficient (in units of −4), and Sp. R. denotes the Spectral Radius. RT corresponds to the mean placement runtime employing Innovus, measured in minutes.

Training Result There are a total of 6 images used out of which 3 images used for training and another 3 for testing. To facilitate placement parameter optimization, we opt for the use of 3 filters, as per our preference. Each of these filters comprises 81 neurons, forming a 9x9 grid of neurons. Each neuron in this grid is connected to 9 other neurons, corresponding to the 3x3 receptive field, and considering there are 3 filters, the total number of neurons in this configuration amounts to 243. In our deep neural network (DNN), we incorporate a total of 13 layers. For the processing of each image, it is divided into a 3x3 grid. Convolution operations are performed on each grid, and we utilize max-pooling layers to down sample the data, enabling us to comprehend the depth of each image effec tively. (i) Number of iteration = 5

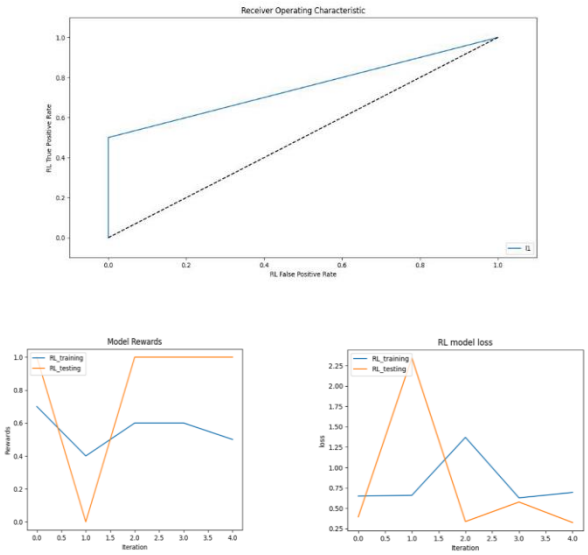


Fig.5 For 5 iterations (a) model reward and (b) RL model loss (c) Receiver operating .

Characteristic. The confusion matrix reveals specific values: TP (True Positives) equal to 6, TN (True Negatives) equal to 0, FP (False Positives) equal to 3, and FN (False Negatives) equal to 3. This arises from a dataset comprising 6 images. However, the noteworthy observation is that the accuracy of the model stands at 75%. From Figure 5 (a), it is evident that as the total number of iterations increases, there is a corresponding increase in the reward for both training and testing. However, in Figure 5.1(b), a different trend is observed. As the number of iterations increases, the error decreases. This suggests that with more iterations, the model's performance improves, leading to higher rewards during training and reduced errors during testing. The Receiver Operating Characteristic (ROC) curve is a valuable tool for assessing the performance of classification models, especially in scenarios where class distribution is imbalanced or when the cost of false positives and false negatives varies. It helps in choosing an appropriate threshold for making decisions based on the model's predictions, balancing the trade-off between sensitivity and specificity. The ROC curve is generated and studied in our method to understand how well the images are learnt based on no of iterations. In the fig 5 (c) TPR at 0.5 and "bending," you are likely referring to the true positive rate at a specific threshold of 0.5 and observing that the ROC curve may be bending away from the diagonal, indicating improved classification performance at that threshold. This is a desirable characteristic, as it implies that the model is making better trade-offs between true positives and false positives. (ii) Number of iteration=100 From Figure 6(a), it is evident that as the total number of iterations increases, there is a corresponding increase in the reward

for both training and testing. However, in Figure 6(b), a different trend is observed. As the number of iterations increases, the error decreases. This suggests that with more iterations, the model's performance improves, leading to higher rewards during training and reduced errors during testing. In this specific scenario, during 100 iterations of the model's performance evaluation, the confusion matrix yields the following results: - True Positives (TP) = 6 ,- True Negatives (TN) = 0 ,- False Positives (FP) = 0 ,- False Negatives (FN) = 6 These values are derived from a dataset containing a total of 6 images. Remarkably, the accuracy achieved in this case is 100%. This remarkable accuracy indicates that, with the benefit of 100 iterations, the model has successfully learned and accurately classified all of the images in the dataset. In Fig.6(c) is observing a TPR of 1 (meaning perfect sensitivity) at a threshold of 1, this suggests that all positive instances are correctly classified as positive, and there are no false negatives. However, this is unusual in practice because it implies that the model is

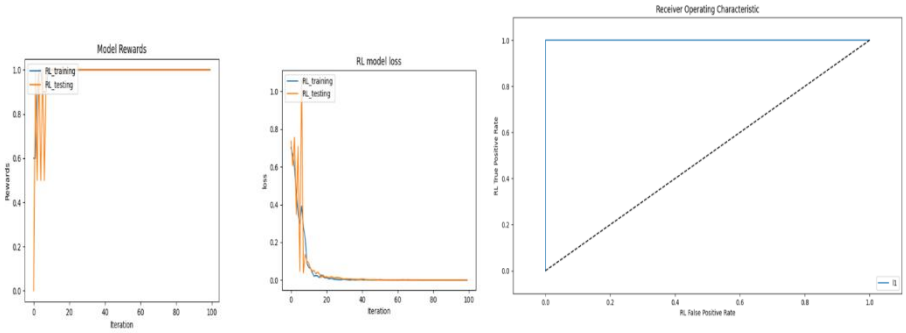


Fig.6 For 100 iterations (a) model reward and (b) RL model loss (c) ROC curve.

Extremely confident in its predictions. The concept of "bending" in the ROC curve is more commonly associated with changes in sensitivity and specificity as the threshold varies in a typical range between 0 and 1.

This table.6 presents a comparison of HPWL results after placement on training netlists, involving human-designed placement, the Multi-Armed Bandit (MAB) method [23], the existing RL model and our RL-based approach. HPWL values are expressed in meters (□). The Δ column signifies the percentage of negative improvement compared to the human-designed placement. Table.6 presents the most optimal wavelengths achieved by the Multi-Armed Bandit (MAB) approach, the existing RL model and our RL agent during the training process. The human baseline reference is established through the efforts of a skilled engineer who dedicated a day to parameter tuning. It's noteworthy that our RL agent exhibits superior performance compared to MAB and the existing RL model on the majority of netlists, achieving a remarkable 8.7% reduction in HPWL on the AVC-Nova Core benchmark. In summary, all the methods demonstrate substantial improvements over the human baseline.

Table.6. Comparative Analysis of Half-Perimeter Bounding Box (HPWL) Following Placement on Training Netlist

Netlist	Human	MAB [23] (Δ %)	RL (Δ %)	Our RL Algorithm
PCI	0.01	0.0092 (-8.0%)	0.0092 (-8.0%)	0.00903 (-7.1%)
DMA	0.149	0.139 (-6.7%)	0.135 (-9.4%)	0.120000000000000001 (-9.3%)
819	0.3	0.28 (-6.7%)	0.28 (-6.7%)	0.25 (6.8%)
DES	0.42	0.37 (-11.9%)	0.36 (-14.3%)	0.35 (14.6%)
VGA	1.52	1.40 (-7.9%)	1.41 (-7.2%)	1.7 (-7.7%)
ECG	0.72	0.65 (-9.7%)	0.68 (-5.5%)	0.6599999999999999 (-5.5%)

Rocket	1.33	1.27 (-4.5%)	1.20 (-9.8%)	1.9 (-9.8%)
AES	1.49	1.44 (-2.7%)	1.40 (-6.0%)	1.8 (-6.9%)
AVC-nova	1.59	1.49 (-6.3%)	1.46 (-8.2%)	1.5 (-8.7%)
Tate	1.53	1.42 (-7.2%)	1.45 (-5.2%)	1.5 (-5.8%)

Testing Result

To assess the generalization capability of our agent, we evaluate its performance on four previously unseen test netlists. This evaluation is conducted without any further training, meaning that the network parameters remain fixed. The RL agent iteratively enhances an initially random parameter set by selecting the action (denoted as) with the highest predicted probability. Since our actions are deterministic, we can determine the resulting parameter set, which is then provided as feedback to the network. This process is repeated until the estimated value decreases for three consecutive updates. At this point, we backtrack to the parameter settings that yielded the highest value.

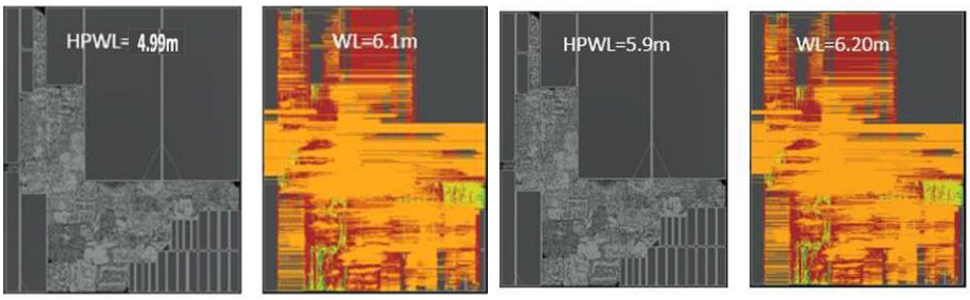
Circuit	Metric	human	MAB	RL	Our RL
LDPC	WL(m)	1.65	1.57	1.53	1.0
	WNS(ns)	-0.005	-0.001	-0.001	-0.002
	Power(mw)	162.10	156.49	154.77	150.63
OpenPt	WL(m)	6.31	6.24	6.20	6.1
	WNS(ns)	-0.003	-0.001	0.0	0.0
	Power(mw)	192.08	190.95	189.72	161.1
Netcard	WL(m)	8.01	7.44	7.15	6.3
	WNS(ns)	-0.006	-0.007	-0.004	0.0
	Power(mw)	174.05	170.51	167.70	153.5
Leon3	WL(m)	5.66	5.53	5.41	5.0
	WNS(ns)	-0.005	-0.001	-0.003	-0.001
	Power(mw)	156.83	156.00	155.51	153.2

This methodology allows us to identify a "good" candidate parameter set without actually performing any placement. Subsequently, we execute a single placement using this parameter set and record the resulting wirelength. In contrast, the Multi-Armed Bandit (MAB) approach relies on the reward signal to suggest new parameter sets, necessitating the execution of actual placements by the tool. We monitor the best wirelength obtained and allocate 50 sequential iterations for the MAB's optimization process. Table.7 displays the optimal wirelength results obtained by our RL agent , RL and the MAB across all four test netlists. It's evident that our RL agent consistently outperforms the MAB in terms of wirelength optimization, and notably, it achieves this superior performance with just a single placement. To validate that the improvement in Half-Perimeter Wire Length (HPWL) achieved during placement translates into a corresponding reduction in the final routed wirelength, we conducted routing for the placed designs. The layouts of the OpenPiton Core designs are visually depicted in Figure 8. It ensured that routing was completed without encountering congestion issues or Design Rule Check (DRC) violations. The Power, Performance, and Area (PPA) metrics for the routed designs are summarized in Table 8. Our observations indicate that the reduction in HPWL achieved during placement is consistently preserved after routing

for all test designs. This translates to significant wirelength savings, with LDPC and Net Card designs achieving reductions of 7.3% and 11% compared to the human baseline.

Netlist	Human	Number of iteration	MAB[23]in m (%negative improvement)	Number of iteration	RL[2] in m (% negative improvement)	Number of iteration	Proposed RL in m (% negative improvement)	Number of iteration
LDPC	1.14	20	1.04 (-8.8%)	50	1.03 (-9.77%)	1	1.02 (-10.73%)	1
Openpt	5.26	20	5.11 (-2.9%)	50	5.9 (-3.87%)	1	5.4 (-4.87%)	1
Netcard	4.88	20	4.45 (-8.8%)	50	4.34 (-11.1%)	1	4.1 (-13.4%)	1
Leon3	3.52	20	3.37 (-4.3%)	50	3.29 (-6.5%)	1	3.2 (-8.7%)	1

TABLE 8. Comparison of PPA (Power, Performance, Area) after Routing on test set



(a) Human design (runtime 7hrs) (b) Multi-armed bandit (runtime 16hrs)
(c) RL (runtime 20min) d) Proposed RL (runtime 16.6min)

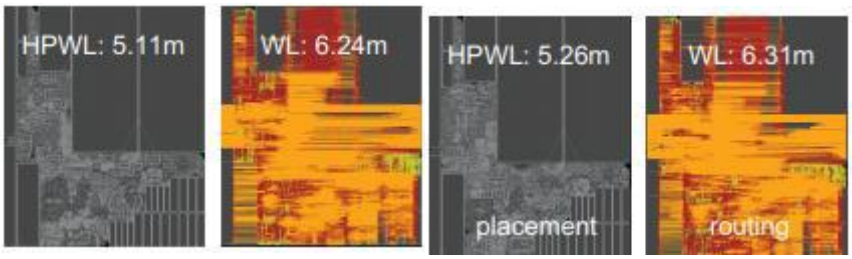


Fig. 7. GDSII layout of Openpiton.

CONCLUSION

Deep RL is a promising approach for solving combinatorial problems, and enables domain adaptation and direct optimization of non-differentiable objective functions. Training RL policies is a very challenging task, in part due to the brittleness of gradient updates and the costliness of evaluating rewards. In this work, we provide an overview of problem, and discuss strategies for training successful RL agents. our RL agent exhibits superior performance compared to MAB and the existing RL model on the majority of netlists, achieving a remarkable 8.7% reduction in HPWL on the AVC-Nova Core benchmark and the reduction in HPWL achieved during placement is consistently preserved after routing for all test designs. This translates to significant wirelength savings, with LDPC and Net Card designs achieving reductions of 7.3% and 11% compared to the human baseline.We predict a trend towards more effective RL-based domain adaptation techniques, in which graph neural networks will play a key role in enabling both higher sample efficiency and more optimal placements. We also foresee a future in which easy to use RL-based placement tools will enable non-ML experts to harness and improve upon this powerful technique.

Acknowledgement

The authors would like to express their sincere gratitude to the Department of Electronics and Communication Engineering at BMS College of Engineering, Bengaluru, India, and the Electrical Engineering and Computer Science Department at Florida Atlantic University, Boca Raton, FL, USA, for their continuous support throughout the research. Special thanks to the researchers who contributed valuable insights and guidance during the study.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Author Contribution

The authors confirm contribution to the paper as follows:

Study Conception and Design: Bindushree V, Jayagowri R

Data Collection: Bindushree V, Roohum Jegan

Analysis and Interpretation of Results: Bindushree V, Jayagowri R, Roohum Jegan, Sree Ranjani Rajendran

Draft Manuscript Preparation: Bindushree V, Jayagowri R, Vinolya S , Y N Gowthami

Final Manuscript Review and Approval: Bindushree V, Jayagowri R, Roohum Jegan

REFERENCES

1. Wang, Z., H.-X. Li, and C. Chen. "Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting." IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 6, June 2020, pp. 1870–83.
2. Agnesina, A., K. Chang, and S. K. Lim. "VLSI Placement Parameter Optimization Using Deep Reinforcement Learning." Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2020, pp. 1–9.
3. Mansoor, A., and M. Chrzanowska-Jeske. "RS3DPlace: Monolithic 3D IC Placement Using Reinforcement Learning and Simulated Annealing." Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, 2022, pp. 394–98.
4. Mathur, Mrinal. "Routing and Placement of Macros Using Deep Reinforcement Learning." Unpublished manuscript.
5. Almeida, S. F., J. Luís Güntzel, L. Behjat, and C. Meinhardt. "Routability-Driven Detailed Placement Using Reinforcement Learning." Proceedings of the IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC), Patras, Greece, 2022, pp. 1–2.
6. Manimegalai, R., et al. "Placement and Routing for 3D FPGAs Using Reinforcement Learning and Support Vector Machines." Proceedings of the 18th International Conference on VLSI Design and 4th International Conference on Embedded Systems Design, Kolkata, India, 2005, pp. 451–56.
7. Cheng, Ruoyu, and Junchi Yan. "On Joint Learning for Solving Placement and Routing in Chip Design." arXiv preprint arXiv:2105.01150, 2021.
8. Bai, L., and L. Chen. "Machine-Learning-Based Early-Stage Timing Prediction in SoC Physical Design." Proceedings of the 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Qingdao, China, 2018, pp. 1–3.
9. Barboza, E. C., N. Shukla, Y. Chen, and J. Hu. "Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism." Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, 2019, pp. 1–6.
10. Mozhzhukhina, A. "VLSI Element Placement Method Based on Deep Reinforcement Learning." International Journal of Professional Science, no. 2, 2022.
11. Huang, Y.-Y., C.-T. Lin, W.-L. Liang, and H.-M. Chen. "Learning-Based Placement Refinement to Reduce DRC Short Violations." Proceedings of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2021, pp. 1–4.
12. Chang, Fu-Chieh. "Late Breaking Results: Flexible Chip Placement via Reinforcement Learning." Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC), 2022.
13. Esmaeili, P., T. Martin, S. Areibi, and G. Grewal. "Guiding FPGA Detailed Placement via Reinforcement Learning." Proceedings of the IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC), Patras, Greece, 2022, pp. 1–6.
14. Liu, S., et al. "Global Placement with Deep Learning Enabled Explicit Routability Optimization." Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2021, pp. 1821–24.
15. Lai, Yao, Yao Mu, and Ping Luo. "MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.
16. Lin, Y., et al. "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement." Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, 2019, pp. 1–6.
17. Wu, Lei, Deng, Guoqi Li, and Yuan Xie. "Core Placement Optimization for Multichip Many-Core Neural Network Systems with Reinforcement Learning." ACM Transactions on Design

Automation of Electronic Systems (TODAES), vol. 26, no. 2, Mar. 2021, Article 11.
18. Mirhoseini, A., et al. "Chip Placement with Deep Reinforcement Learning." arXiv preprint arXiv:2004.10746, 2020.
19. Gandhi, U. "A Reinforcement Learning-Based Framework to Generate Routing Solutions and Correct Violations in VLSI Physical Design." M.S. Thesis, University of Calgary, 2020.
20. Gandhi, U., I. Bustany, W. Swartz, and L. Behjat. "A Reinforcement Learning-Based Framework for Solving Physical Design Routing Problem in the Absence of Large Test Sets." Proceedings of the ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD), 2019.
21. Goldie, A., and A. Mirhoseini. "Placement Optimization with Deep Reinforcement Learning." Proceedings of the International Symposium on Physical Design (ISPD), ACM, 2020.
22. Kahng, Andrew B. "Machine Learning Applications in Physical Design: Recent Results and Directions." Proceedings of the International Symposium on Physical Design (ISPD '18), Association for Computing Machinery, New York, 2018, pp. 68–73.
23. Ansel, J., et al. "OpenTuner: An Extensible Framework for Program Autotuning." Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT '14), 2014.
24. Bindushree, V., and R. Jaya Gowri. "Implementation of Machine Learning in the Field of VLSI Placement: A Review." Journal of Emerging Technologies and Innovative Research, vol. 10, no. 2, Feb. 2023, pp. d112–d116.