ACCELERATING CHIPLET PLACEMENT AND ROUTING OPTIMIZATION WITH MACHINE LEARNING

Haeyeon Rachel Kim[1,*] **Federico Berto** [1,*] **Junghyun Lee** [1] **Hyunjun An** [1] **Taein Shin** [1]
Chuanbo Hua [1] **Jinkyoo Park** [1] **Youngwoo Kim** [2] **Joungho Kim** [1]
[1]KAIST, [2]Sejong University

Track : 14. Machine Learning for Microelectronics, Signaling & System Design

Audience Level : All

KEY TAKEAWAYS (50 WORDS)

Chiplet technology improves system performance and customization while maintaining cost efficiency. However, chiplet placement and routing pose significant challenges due to hard constraints and combinatorial complexity. We formulate the problem as a constrained optimization task, propose a novel representation and benchmark, and implement reinforcement learning approaches to optimize chiplet-based system design.

ABSTRACT/SESSION DESCRIPTION (200 WORDS)

Chiplet technology has emerged as a promising solution to address the growing demand for improved system performance and customization while maintaining production cost efficiency. Chiplet-based systems offer several key economic advantages over monolithic chips, including improved wafer yield, mixed process technology node integration, reduced time to market, and the ability to overcome reticle size limitations. However, the heterogeneous integration of an increasing number of chiplets and their interconnections poses a significant challenge in chiplet placement and routing. Chiplet placement is subject to various hard constraints, such as physical limitations, signal integrity, power integrity, and thermal coupling considerations, making it a complex problem distinct from traditional chip placement.

In this paper, we formulate the chiplet placement and routing problem as a constrained combinatorial optimization task and propose a novel representation and benchmark for the problem. We implement various reinforcement learning approaches using open-source libraries to train autoregressive policies, aiming to address the complex challenges associated with chiplet placement and routing. Our work contributes to the advancement of chiplet technology and its application in various domains, enabling the design of high-performance, cost-efficient, and customizable chiplet-based systems.

EXTENDED ABSTRACT/SESSION FOCUS (500 WORDS)

Chiplet technology has emerged as a promising solution to address the growing demand for improved system performance and customization while maintaining production cost efficiency. As Moore predicted in 1965, building large systems out of smaller, separately packaged, and interconnected functions may prove more economical. Chiplet-based systems offer several key economic advantages over monolithic chips, including improved wafer yield, mixed process technology node integration, reduced time to market, and the ability to overcome reticle size limitations. However, the heterogeneous integration of an increasing number of chiplets and their interconnections poses a significant challenge in chiplet placement and routing.

The chiplet placement and routing problem is a critical challenge in the design of chiplet-based systems. This problem involves determining the optimal placement of chiplets on an interposer and routing the interconnections between them while minimizing the total route length and satisfying various hard constraints. Each chiplet has a unique size and port locations, and the interconnections

[*]Equal contribution

between chiplets are defined by a set of netlists, which specify the connections between the ports of two chiplets. The placement and routing of chiplets are interdependent, as the routing solution heavily relies on the initial placement of the chiplets. Moreover, the order in which the routing is performed significantly impacts the final solution quality.

The problem is further complicated by the fact that both the placement and routing subproblems are subject to hard constraints, including physical constraints (e.g., no overlaps between chiplets and interconnections), signal integrity specifications (e.g., maximum interconnection reach and data rate), power integrity specifications (e.g., IR drop and simultaneous switching noise), and thermal coupling considerations (e.g., the minimum spacing between chiplets). Chiplets must not overlap, and routings cannot intersect, which inherently leads to a high failure rate in finding feasible solutions. The feasibility of the problem and the final cost of the solution are highly dependent on the order in which the chiplets are placed and the order in which the routing is performed. This combinatorial nature of the problem adds an additional layer of complexity, as the search space for feasible and optimal solutions expands exponentially with the number of chiplets and netlists.

In this paper, we formulate the chiplet placement and routing problem as a constrained combinatorial optimization task, where chip placement is a prior action, and the routing of each netlist becomes the following action. We propose a novel representation and benchmark for the problem and implement various reinforcement learning approaches using open-source libraries to train auto-regressive policies. Our approach aims to address the complex challenges associated with chiplet placement and routing, considering the various hard constraints and objectives specific to chiplet-based system design. By developing a reinforcement learning-based solution, we seek to optimize chiplet placement and routing, enabling the design of high-performance, cost-efficient, and customizable chiplet-based systems. This work contributes to the advancement of chiplet technology and its application in various domains, including data centers and edge AI.

CONTENTS

1 Introduction **4**

2 Chiplet Placement and Routing Problem Statement **6**

2.1 General MDP Formulation 7

2.2 MDP for Chiplet Placement and Routing Problem 7

2.3 Problem Complexity 9

3 Reinforcement Learning for Chiplet Placement **11**

3.1 Autoregressive Policies 11

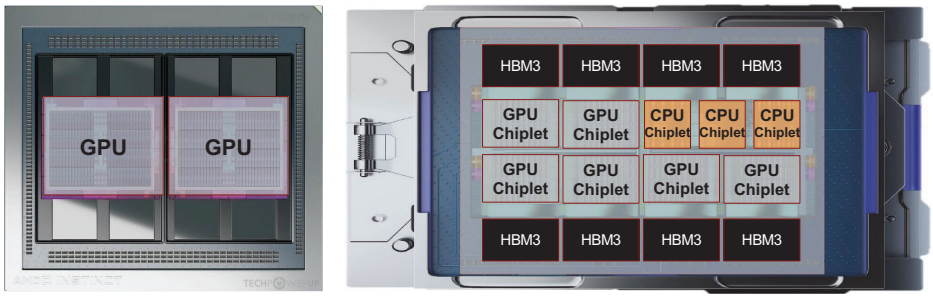
3.2 Deep Reinforcement Learning 11

3.3 Open-Source Libraries 12

4 Further Works for the Main DesignCon Paper **12**

1 INTRODUCTION

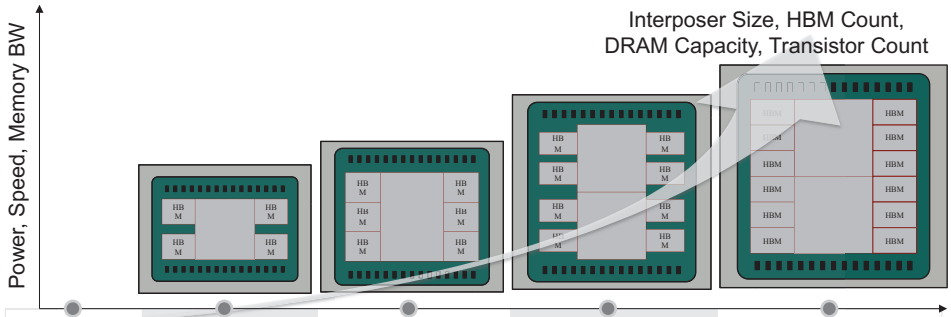
The semiconductor industry has been driven by the relentless pursuit of improved performance, guided by the principles of scaling as outlined in Moore's law. However, as the limits of traditional scaling are being reached, advanced packaging techniques have emerged as promising solutions to maintain the pace of performance improvement. In recent years, the demand for even greater performance enhancements, customization, and production cost efficiency has brought chiplet technology to the forefront of the semiconductor industry. In 1965, Moore predicted that building large systems out of smaller, separately packaged, and interconnected functions might prove more economical [1]. This concept, now known as chiplet technology, offers several key economic advantages over monolithic chips, including improved wafer yield, mixed process technology node integration, reduced time to market, and the ability to overcome reticle size limitations.



< AMD MI250 Architecture > < Chiplet-based AMD MI300 Architecture >

Figure 1: Comparison between the architectures of AMD MI250 and MI300

A recent example of a chiplet-based product is AMD's MI300 in Fig. 1, which integrates 9 TSMC 5nm chiplets and 8 high-bandwidth memory (HBM) modules. Compared to its predecessor, the MI250, the MI300 achieves an impressive 8× improvement in AI performance and a 5× improvement in AI performance-per-watt. As a result, the chipletization of systems-on-chip (SoCs) is expected to accelerate in the near future.



	2011	2016	2019	2021	2023~
Interposer Size (1X)		1.5X	2X	3X	4X (~3400mm²)
HBM Count		4 HBMs	6 HBMs	8 HBMs	12 HBMs
DRAM Capacity		16GB	48GB	128GB	192GB~
Tr. Count		1X	8X	20X	TBD

Figure 2: Interposer roadmap by TSMC

According to the Fig. 2, the number of chiplets heterogeneously integrated within a GPU module is expected to increase as the HBM count integrated within the module rises and as the size of interposer expands, facilitated by the advancements in CoWoS technology.

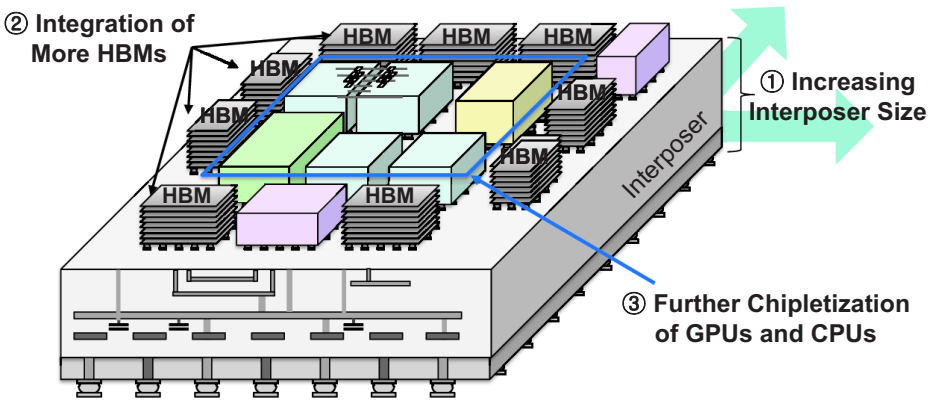


Figure 3: Chiplet-based 2.5D heterogeneous integration roadmap

As illustrated in Fig. 3, the chiplet-based 2.5D heterogeneous integration roadmap involves the integration of more HBMs, further chipletization of GPUs and CPUs, and an increase in interposer size. According to this roadmap, the number of chiplets integrated and their interconnections is projected to increase.

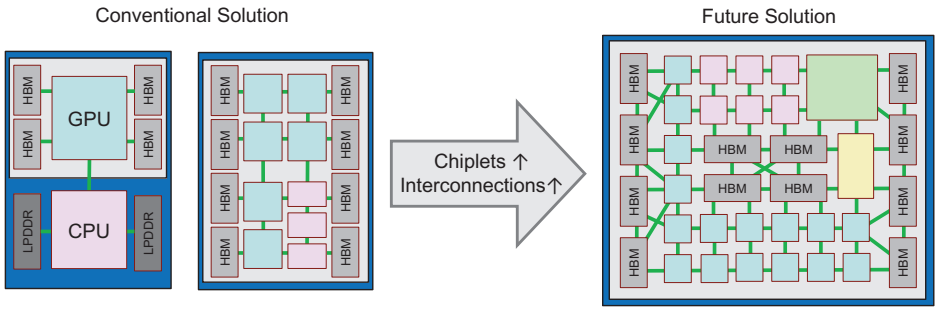


Figure 4: Future chiplet-based architecture with increasing number of chiplets and interconnections

The heterogeneous integration of an increasing number of chiplets and their interconnections poses a significant challenge in chiplet placement and routing. As shown in Fig. 4, the conventional solution only involves a few components, whereas the future solution is expected to involve more chiplets and interconnections. The number of interconnections within a system can range from $O(N)$ to $O(N^2)$, where N is the number of chiplets. Considering the limited interposer size, placing chiplets and routing their interconnections becomes a very challenging problem. Moreover, chiplet placement ultimately determines the overall architecture. It is subject to various hard constraints, including physical constraints (e.g., no overlaps between chiplets and interconnections), signal integrity specifications (e.g., maximum interconnection reach and data rate), power integrity specifications (e.g., IR drop and simultaneous switching noise), and thermal coupling considerations (e.g., the minimum spacing between chiplets).

Although the objectives of chip placement and chiplet placement may appear similar, they have distinct problem contexts, rewards, and constraints. Chiplet placement must consider multiple constraints from various electrical requirements and specifications (SI/PI/TI), while chip placement primarily focuses on the trade-off between the wirelength and congestion [7]. The comparison between chip placement and chiplet placement problems is summarized in Table 1.

In this paper, we formulate the chiplet placement and routing problem as a novel constrained combinatorial problem, where chip placement is a prior action, and the routing of each netlist becomes the following action. We hypothesize that routing is significantly influenced by the initial placement actions. Therefore, we approach placement and routing as a unified problem and aim to train a policy that strategically places chiplets with consideration of subsequent routing actions. We pro-

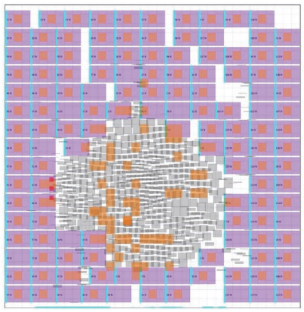
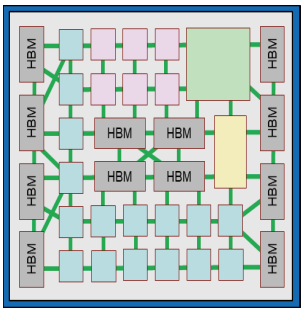
	Chip (Standard Cell, Macro) Placement Problem	Chiplet Placement Problem
		
Number of Components	$\sim 10^6$	10∼1000
Component Overlap Constraint	×	✓
Routing Overlap Constraint	×	✓
Maximum Wirelength Constraint	×	✓
Reward Type	Wirelength, congestion	SI/PI/TI

Table 1: Comparison between Chip and Chiplet Placement Problems

pose a novel representation and benchmark for the problem and implement a reinforcement learning approach using open-source libraries to train auto-regressive policies.

The main contributions of this paper are as follows:

- We define a novel chiplet placement and routing optimization problem.
- We implement the chiplet placement and routing optimization problem as a benchmark.
- We utilize state-of-the-art open-source libraries [11, 10] to make the problem accessible to the broader research community so that it may bootstrap further future works.

2 Chiplet Placement and Routing Problem Statement

Chiplet Placement and Routing Problem Definition:

- Optimally place chiplets on an interposer and route interconnections between them.
- Objective: Minimize the total datarate-weighted route length while satisfying hard constraints.
- Chiplet characteristics:
  - Unique size and port locations.
  - Interconnections defined by netlists, specifying connections between ports of two chiplets.
- Interdependence of placement and routing:
  - Routing solution heavily relies on the initial placement of chiplets.
  - Order of routing significantly impacts final solution quality.
- Hard constraints:
  - Chiplets must not overlap.
  - Routings cannot intersect.
  - High failure rate in finding feasible solutions due to hard constraints.
- Combinatorial nature of the problem:

  - Feasibility and final cost depend on the order of placement and routing.
  - Search space expands exponentially with the number of chiplets and netlists.
- Importance of the problem:
  - Directly influences system performance, power consumption, and overall cost.
  - Challenging to find feasible and optimal solutions due to hard constraints and combinatorial nature.
  - Requires advanced optimization techniques and algorithms to tackle the complexity.

2.1 General MDP Formulation

We formulate the process of generating solutions for a problem as a Markov Decision Process (MDP). Given a problem instance $\boldsymbol{x}$, the MDP is defined by the following components:

- **State Space** $\mathcal{S}$: A set of states that encapsulate the information about the given chiplets and netlists $\boldsymbol{x}$ and the partial solution at each step of the solution-generating process.
- **Action Space** $\mathcal{A}$: A set of actions a_t available at each step t, which can modify the current partial solution to progress towards a complete solution.
- **Transition Function** $\mathcal{T}$: A deterministic function $s_{t+1} = \mathcal{T}(s_t, a_t)$ that maps a state-action pair (s_t, a_t) to the next state s_{t+1}, defining how the partial solution evolves based on the chosen action.
- **Reward Function** $\mathcal{R}$: A function $\mathcal{R}(s_t, a_t)$ that assigns an immediate scalar reward to each state-action pair (s_t, a_t), providing feedback to the agent about the desirability of the chosen action in the current state.
- **Discount Factor** γ: A parameter $\gamma \in [0, 1]$ that balances the importance of immediate and future rewards, allowing the agent to consider the long-term consequences of its actions.

The solution to a chiplet placement and routing problem $\boldsymbol{x}$ is represented as a sequence of actions $\boldsymbol{a} = (a_1, \ldots, a_T)$ taken over T steps, thanks to the deterministic nature of the transition function. The objective is to find a sequence of actions that minimizes the total cost of the problem, which is equivalent to maximizing the cumulative discounted reward $\sum_{t=1}^{T} \gamma^t \mathcal{R}(s_t, a_t)$ in the MDP formulation.

2.2 MDP for Chiplet Placement and Routing Problem

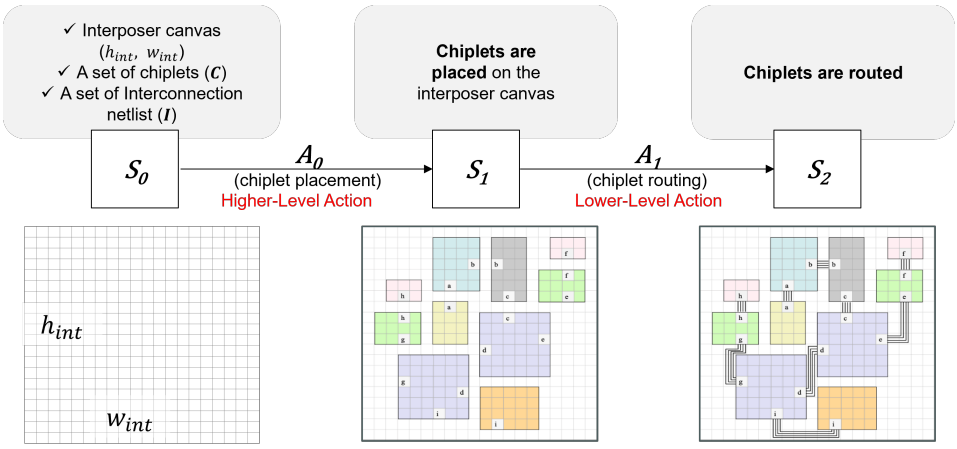


Figure 5: Hierarchical MDP for chiplet placement and routing problem

As we formulate the chiplet placement and routing problem as a unified problem, we define the MDP as a hierarchical MDP that consists of a higher-level placement action set and lower-level routing action set as described in Fig. 5. Given a fixed-sized interposer canvas (h_{int}, w_{int}), a set of chiplets (C) to be placed and a set of interconnection netlist (I), there follows two different types of actions.

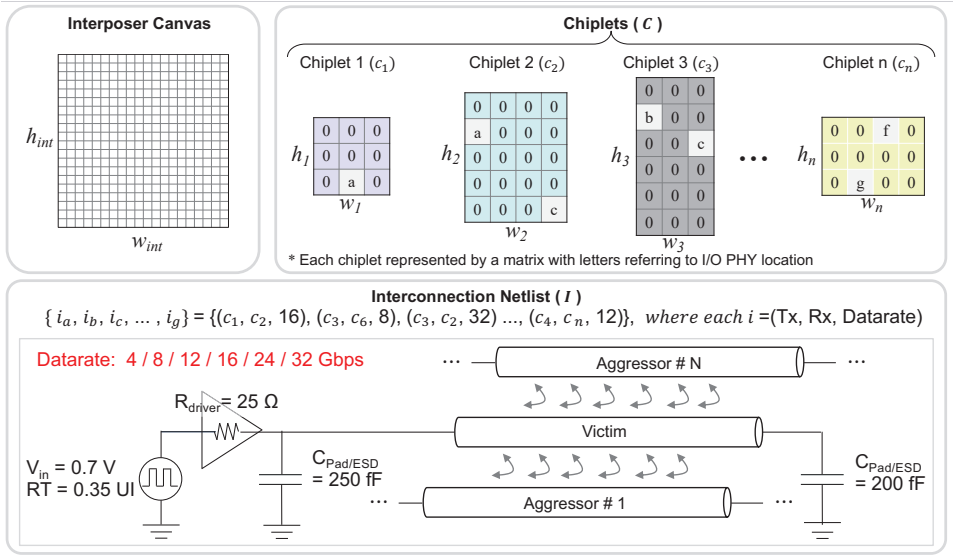


Figure 6: Initial state of the hierarchical MDP of chiplet placement and routing problem

In Fig. 6, the initial state (S_0) consists of interposer canvas size, chiplets (C), and interconnection netlists (I), each of which consists of multiple components. Interposer canvas is simply defined by its width and height. Chiplets (C) is composed of chiplets, each of which is defined by the size, and transmitter and receiver port locations. Netlists (I) is a set of interconnections, each of which is defined by the transmitter chiplet, receiver chiplet, and the datarate.

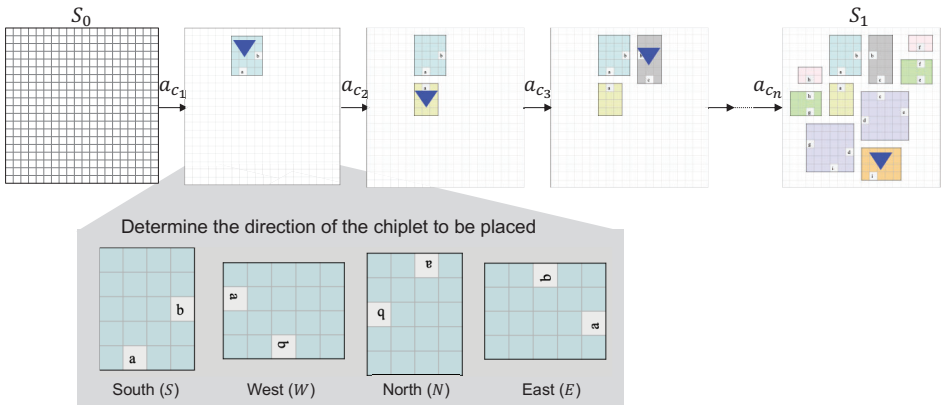


Figure 7: Graphical representation of higher-level chiplet placement action (A_0) as a set of sequential sub-actions

In Fig. 7, the higher-level chiplet placement action (A_0) is composed of the sequential placement of chiplets as $A_0 = \{a_{c_1}, a_{c_2}, a_{c_3}, ..., a_{c_n}\}$. Chiplets are auto-regressively placed on the interposer canvas as sub-actions, which are represented as the x,y-coordinates of the location of each chiplet with a specific rotational transformation. Subaction is defined as follows: $a_{c_t} = (rotation, location) = (r, (x_t, y_t))$, where $r \in \{S, W, N, E\}$.

According to the higher-level chiplet placement action (A_0), chiplet state is represented as a 3D-array as represented in Fig. 8. Each layer of interposer-sized array represents chiplet locations, tx locations and rx locations of each netlist, respectively. This state representation gives both chiplet placement context and routing context, whose embeddings can improve the generalization of the agent during the training.

Figure 8: 3D-array representation of state in chiplet placement environment

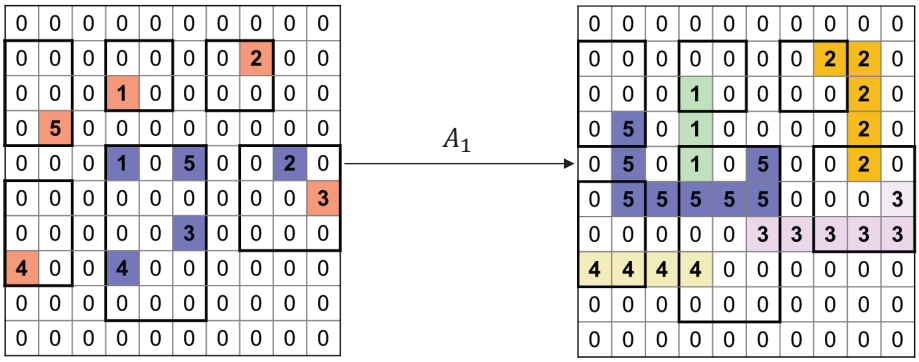


Figure 9: Graphical representation of lower-level routing action (A_1) as a set of sequential routing of each netlist

Followed by the higher-level placement action (A_0), as shown in Fig. 9, lower-level routing action consists of multiple routes according to the netlist (I), as $A_1 = \{a_{i_a}, a_{i_b}, a_{i_c}, ..., a_{i_m}\}$. A routing of the interconnection between two chiplets is referred to as a sub-action, which is the sequence of coordinates in the route between the two tx and rx nodes, $a_i = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), ...(x_l, y_l)\}$, where the first and the last coordinate are the tx and rx nodes for the two chiplets to be routed, and l is the tour length of the route. The datarate of each interconnection are independently determined.

In summary, the hierarchical MDP of the chiplet placement and routing is defined as follows:

- **Initial State:** $S_0 = \{(h_{int}, w_{int}), C, I\}$, where C is a set of chiplets and I is a set of netlists.
- **Placement State:** $S_1 = \{(h_{int}, w_{int}), C, I, A_0\}$
- **Routing State:** $S_2 = \{(h_{int}, w_{int}), C, I, A_0, A_1\}$
- **Higher-level Action:** $A_0 = \{a_{c_1}, a_{c_2}, a_{c_3}, ..., a_{c_n}\}$, where each chiplet placement sub-action is defined as $a_{c_t} = (rotation, location) = (r, (x_t, y_t))$, where $r \in \{S, W, N, E\}$.
- **Lower-level Action:** $A_1 = \{a_{i_a}, a_{i_b}, a_{i_c}, ..., a_{i_m}\}$, where each netlist routing sub-action is defined as $a_i = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), ...(x_l, y_l)\}$.
- **Reward:** $R = \sum_{i=1}^{m} \lambda \cdot |a_{i_m}|$, where λ is a datarate-dependent hyperparameter.

2.3 Problem Complexity

The chiplet placement and routing problem presents a significant challenge due to its combinatorial nature, which results in a vast solution space, and its high problem complexity arising from the interdependent and hard-constrained characteristics of the task. As illustrated in Table 1, although the chip placement problem [8] involves a considerably larger number of components compared to the chiplet placement problem, the latter exhibits distinct complexity dimensions. This is attributed to

the presence of various hard constraints and the requirement for simulation-based reward evaluation, setting the chiplet placement problem apart from its chip placement counterpart.

To illustrate the complexity of the chiplet placement and routing problem, we implemented the hierarchical Markov Decision Process (MDP) described in Section 2.2. This implementation was based on random solution generation, and we evaluated the success rate of obtaining a feasible solution as well as the failure rates of placement and routing separately. In this context, failure refers to the inability to satisfy the hard constraints imposed by the problem. By analyzing these metrics, we aim to demonstrate the inherent difficulty in finding even a single feasible solution for this problem, highlighting the significant challenges associated with chiplet placement and routing optimization.

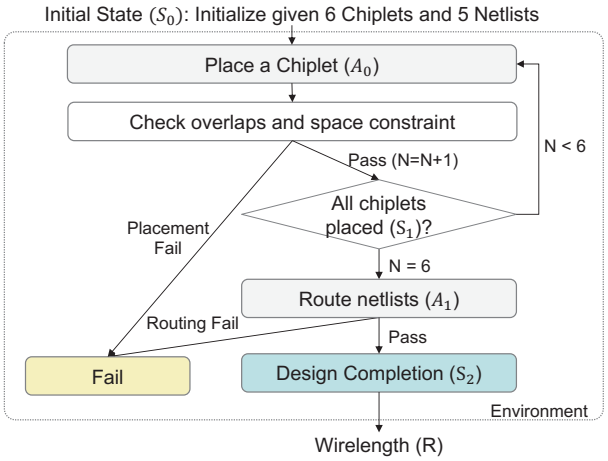


Figure 10: Single iteration/episode procedure of random solution generation for chiplet palcement and routing MDP

We formulated a problem with 6 chiplets and 5 netlists and the procedure for generating a random solution for an iteration/episode is shown in Fig. 10.

Iterations (N)	Wirelength (R)	Completion Rate	Placement Fail	Routing Fail
100	52	19.00%	50.60%	49.40%
300	51	13.00%	55.60%	44.40%
500	47	13.80%	56.80%	43.20%
1,000	41	13.40%	52.10%	47.90%
2,000	43	13.30%	51.70%	48.30%
10,000	39	13.29%	53.10%	46.90%
100,000	37	12.88%	52.80%	47.20%

Table 2: Iteration results for wirelength, completion rate, placement fail, and routing fail.

Table 2 presents the results of our evaluation of the random solution generation to the hierarchical MDP implementation for the chiplet placement and routing problem. We conducted experiments with varying numbers of iterations, ranging from 100 to 100,000, to assess the performance of the random solution generation approach. The table reports several key metrics, including the total wirelength, completion rate, placement fail rate, and routing fail rate.

The total wirelength represents the sum of the lengths of all the interconnections between the chiplets, serving as a measure of the overall solution cost. As the number of iterations increases, the total wirelength tends to decrease.

The completion rate refers to the percentage of iterations that successfully yield a feasible solution, successful placement of all chiplet and routing of all netlist satisfying all the hard constraints. Notably, the completion rate remains relatively low, ranging from 12.88% to 19.00%, highlighting the difficulty in finding feasible solutions for the chiplet placement and routing problem.

The placement fail rate and routing fail rate provide insights into the specific stages where failures occur. The placement fail rate represents the percentage of iterations where the placement of chiplets violates the hard constraints, while the routing fail rate indicates the percentage of iterations where the routing of interconnections fails to meet the constraints. Interestingly, the placement fail rate is consistently higher than the routing fail rate, suggesting that the placement stage is more prone to constraint violations in this problem.

Moreover, as the number of iterations increases, the placement fail rate and routing fail rate remain relatively stable, indicating that simply increasing the number of random solution attempts does not significantly improve the chances of finding feasible solutions. This observation underscores the need for more sophisticated optimization techniques, such as reinforcement learning, to effectively tackle the complexity of the chiplet placement and routing problem.

Overall, the results presented in Table 2 demonstrate the challenging nature of the chiplet placement and routing problem, with low completion rates and high failure rates even with a large number of iterations. These findings motivate the development of advanced optimization approaches to efficiently explore the vast solution space and find high-quality, constraint-satisfying solutions.

3 Reinforcement Learning for Chiplet Placement

3.1 Autoregressive Policies

In the reinforcement learning framework, a policy π_θ is employed to construct a solution from scratch for a given problem instance $\boldsymbol{x}$. The policy is parameterized by a set of learnable parameters θ, which are optimized during the training process. We consider the problem of generating a solution as an autoregressive sequence generation task.

The policy π_θ consists of two main components: an encoder f_θ and a decoder g_θ. The encoder f_θ is a function that maps the problem instance $\boldsymbol{x}$ into a latent embedding space, generating a context vector $\boldsymbol{h} = f_\theta(\boldsymbol{x})$. The decoder g_θ is a function that iteratively determines a sequence of actions $\boldsymbol{a} = (a_1, \ldots, a_T)$ based on the context vector $\boldsymbol{h}$ and the previous actions. Formally, the policy can be defined as:

$$a_t \sim g_\theta(a_t|a_{t-1}, \ldots, a_0, s_t, \boldsymbol{h}), \quad \pi_\theta(\boldsymbol{a}|\boldsymbol{x}) \triangleq \prod_{t=1}^{T} g_\theta(a_t|a_{t-1}, \ldots, a_0, s_t, \boldsymbol{h}), \tag{1}$$

where s_t represents the state at step t, and $\boldsymbol{h}$ is the context vector obtained from the encoder. Note that we abuse the notation slightly by using θ to represent the parameters of the policy π, the encoder f, and the decoder g.

The policy generates actions sequentially, conditioning each action on the previous actions and the current state. This allows the policy to capture dependencies between actions and adapt to the evolving solution. The probability of a complete action sequence $\boldsymbol{a}$ given the problem instance $\boldsymbol{x}$ is expressed as the product of the individual action probabilities.

By treating the solution generation process as an autoregressive sequence generation task, the policy can learn to generate actions that incrementally build a complete solution. The encoder f_θ extracts relevant features from the problem instance, while the decoder g_θ uses this information along with the previously generated actions to make informed decisions about the next action to take.

3.2 Deep Reinforcement Learning

The primary objective in reinforcement learning is to train a policy π_θ that maximizes the expected cumulative reward (or, equivalently, minimizes the total cost) over the distribution of problem instances. This is achieved by optimizing the parameters θ of the policy using deep reinforcement learning techniques.

The optimization problem can be formulated as follows:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})} \left[\mathbb{E}_{\pi_\theta(\boldsymbol{a}|\boldsymbol{x})} \left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}(s_t, a_t) \right] \right], \tag{2}$$

where θ^* represents the optimal parameters that maximize the expected cumulative reward, $P(\boldsymbol{x})$ is the distribution of problem instances, and $\gamma \in [0, 1]$ is the discount factor that balances the importance of immediate and future rewards.

The inner expectation $\mathbb{E}_{\pi_\theta(\boldsymbol{a}|\boldsymbol{x})}[\cdot]$ represents the expected cumulative reward obtained by following the policy π_θ to generate a sequence of actions $\boldsymbol{a}$ for a given problem instance $\boldsymbol{x}$. The outer expectation $\mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}[\cdot]$ averages the expected cumulative reward over the distribution of problem instances.

To optimize the policy parameters θ, various deep reinforcement learning algorithms can be employed, such as policy gradient methods (e.g., REINFORCE [3], Actor-Critic [2], PPO [6]) or value-based methods (e.g., Q-learning, Deep Q-Networks [5]). These algorithms utilize deep neural networks to parameterize the policy π_θ and learn the optimal parameters through iterative updates based on the observed rewards and state transitions.

During training, the policy generates sequences of actions for a batch of problem instances sampled from the distribution $P(\boldsymbol{x})$. The resulting states, actions, and rewards are used to estimate the gradients of the expected cumulative reward with respect to the policy parameters θ. The gradients are then used to update the parameters using optimization techniques such as stochastic gradient descent (SGD) or its variants such as Adam [4].

By iteratively generating solutions, collecting rewards, and updating the policy parameters, the training process allows the policy to learn the underlying patterns and strategies for constructing high-quality solutions across the distribution of problem instances. The trained policy can then be used to generate solutions for new, unseen problem instances efficiently.

The choice of the specific deep reinforcement learning algorithm and the architecture of the neural networks used to parameterize the policy depends on the characteristics of the problem domain and the available computational resources. The training process aims to find the optimal policy parameters θ^* that maximize the expected cumulative reward, enabling the generation of high-quality solutions for the given problem domain.

3.3 Open-Source Libraries

To facilitate the development and benchmarking of deep reinforcement learning approaches for combinatorial optimization, we leverage and contribute to open-source libraries. Our primary goal is to provide a comprehensive and user-friendly framework that enables researchers and practitioners to easily implement, compare, and extend various algorithms in this domain. We adopt the RL4CO library [10] as the main template for our benchmark codebase. RL4CO is a unified interface for defining and solving combinatorial optimization problems using reinforcement learning techniques, such as EDA [9]. It offers a modular and extensible architecture, allowing seamless integration of different problem domains, policy architectures, and training algorithms. Our codebase is also designed to be compatible with TorchRL [11], a popular deep reinforcement learning codebase built on top of PyTorch. TorchRL provides a collection of state-of-the-art reinforcement learning algorithms, including policy gradient methods and value-based methods. By integrating with TorchRL, our codebase can leverage its efficient implementations and benefit from ongoing updates and improvements. By providing an open-source implementation that integrates with existing frameworks and supports standardized evaluation, we aim to accelerate research and development in the context of fat and effective chiplet placement and routing. We believe our codebase library will serve as a valuable resource for the community, promoting collaboration, reproducibility, and the advancement of state-of-the-art techniques in this exciting area.

4 Further Works for the Main DesignCon Paper

- Reward estimator based on eye diagram and simultaneous switching noise will be implemented.
- Extensive experimental results with various reinforcement learning schemes and neural networks, as well as heuristics solvers, will be reported.
- Chiplet placement and routing problem benchmark will be released in GitHub.

REFERENCES

[1] G.E. Moore. "Cramming More Components Onto Integrated Circuits". In: *Proceedings of the IEEE* 86.1 (1998), pp. 82–85. DOI: 10.1109/JPROC.1998.658762.

[2] Vijay Konda and John Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

[3] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).

[4] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[7] Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, Brucek Khailany, and David Z. Pan. "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.4 (2021), pp. 748–761. DOI: 10.1109/TCAD.2020.3003843.

[8] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. "A graph placement methodology for fast chip design". In: *Nature* 594 (June 2021), pp. 207–212. DOI: 10.1038/s41586-021-03544-w.

[9] Haeyeon Kim, Minsu Kim, Federico Berto, Joungho Kim, and Jinkyoo Park. "DevFormer: A Symmetric Transformer for Context-Aware Device Placement". In: 2023. arXiv: 2205.13225 [cs.LG].

[11] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. "TorchRL: A data-driven decision-making library for PyTorch". In: *International conference on learning representations*. 2024. arXiv: 2306.00577 [cs.LG].

[10] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Lin Xie, and Jinkyoo Park. "RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark". In: *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2025. URL: https://rl4.co/.