Chiplet Placement and Routing Optimization: A Novel Benchmark and Neural Solver

Haeyeon Kim*, Federico Berto*, Chuanbo Hua, Minsu Kim, Joungho Kim and Jinkyoo Park
(* for equal contribution)
Korea Advanced Institute of Science and Technology (KAIST)
{haeyeonkim, fberto, cbhua, min-su, joungho, jinkyoo.park}@kaist.ac.kr

Abstract

The exponential growth of AI model sizes has amplified the demand for specialized hardware accelerators capable of efficiently managing complex workloads. To meet the customization needs of AI accelerators, recent advancements have enabled the use of chiplets – modular components of a larger integrated circuit that can be combined to create a complete system on a chip. Chiplet-based architectures offer a flexible and cost-effective solution by integrating modular chiplets with high-bandwidth memory (HBM), effectively addressing both computational power and memory capacity requirements. However, the increased complexity of chiplet designs introduces significant challenges in placement and routing. This paper presents a novel optimization benchmark and neural approach to the chiplet placement and routing problem, leveraging a hierarchical Markov decision process (MDP). We propose CHIPLETFORMER, a neural architecture that optimizes placement and routing by not only minimizing routing length but also improving datarate-dependent electrical system performance, aiming to enhance the efficiency and scalability of AI acceleration systems. We finally outline several promising directions for future work.

1 Introduction

AI model sizes are growing rapidly, leading to an increasing demand for specialized hardware accelerators that can efficiently handle specific AI workloads. This surge is driven by the need for efficient computation, model-specific memory capacity, and cost-effective solutions, as larger models require more resources to function optimally [12, 2, 4]. Traditional monolithic chip designs often struggle to balance the varying computational, memory, and precision requirements of these workloads. Chiplets – smaller, modular pieces of a larger integrated circuit (IC) that can be combined with other chiplets to form a complete system on a chip – present a new design solution by enabling flexibility and modularity [8, 17]. By integrating specialized cores or functions – such as Tensor Cores for deep learning, AI inference cores for edge applications, and high-performance memory controllers – chiplets can be tailored to meet the specific computational needs of AI models [22, 27]. This workload customization allows for optimized performance across different AI tasks, from heavy matrix computations to low-latency memory access [18, 19].

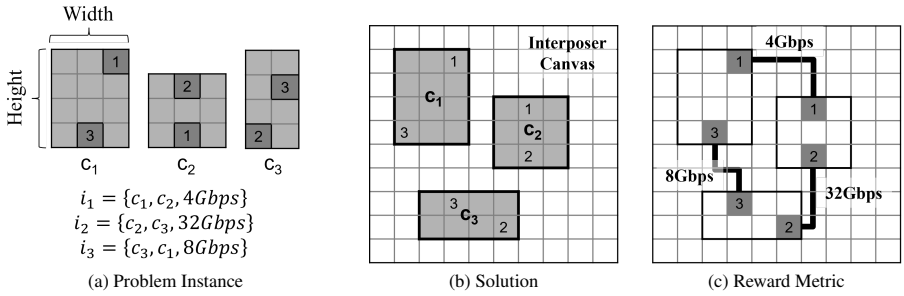
Moreover, the rapid growth of AI model sizes has outpaced improvements in GPU memory capacity, often requiring more GPUs to accommodate memory demands rather than computational needs [21]. Chiplet architectures address this by integrating more high-bandwidth memories (HBMs), allowing for customizable memory capacity without adding unnecessary computational power. Additionally, chiplet-based designs are cost-effective due to improved fabrication yield from reduced die sizes, while heterogeneous integration and the re-use of chiplets offer a more economical solution compared to traditional monolithic designs [24]. This combination of scalability, efficiency, and cost-effectiveness makes chiplets a compelling option for modern AI applications.

Submitted to the ML4Systems Workshop at NeurIPS 2024.

However, adopting chiplet-based designs introduces new challenges in chiplet placement and routing [28]. As systems integrate more chiplets and HBMs, the complexity of design increases exponentially. Efficient placement and routing are crucial as they impact system performance by minimizing signal delays, optimizing power distribution, and managing heat dissipation [6, 5, 23]. Unlike traditional chip placement [15, 13, 11], chiplet placement and routing must account for strict no-overlap constraints, precise routing between specific transmitter and receiver locations, and adherence to predefined netlists [7, 20, 10]. While conventional chip placement benchmarks use half-perimeter wirelength (HPWL) to estimate total wirelength by measuring half the perimeter of the bounding box enclosing terminals, our benchmark incorporates the actual routed path distance and the data rate within the interconnections. Varying data rates introduce specific relationships between routing length and signal integrity degradation for each netlist connection, necessitating novel methods to address the intricate interdependencies and constraints unique to chiplet-based designs.

This paper introduces a novel benchmark for the combinatorial optimization of chiplet placement and routing. In our benchmark, we not only consider the minimum routing path between chiplets, but also signal integrity of the interconnections between chiplets to ensure robust electrical performance to achieve better performance in AI accelerators[14]. We model the hierarchical optimization problem as a Markov decision process (MDP) to concurrently optimize placement and routing. To address the complexity and constraints of chiplet design, we propose ChipletFormer, an architecture that uses autoregressive decision-making for efficient chiplet placement.

2 Chiplet Placement and Routing Benchmark Formulation



(a) Problem Instance (b) Solution (c) Reward Metric

Figure 1: Chiplet placement and routing problem definition and objectives: (a) a problem instance defined in terms of a set of chiplets and netlists. (b) solution as the final design of chiplet placement on the interposer canvas. (c) reward as the measure of signal integrity of routed interconnections.

The chiplet placement and routing problem, illustrated in Fig. 1, is defined as a constrained combinatorial optimization problem with the following specifications: 1) Place the given chiplets of varying dimensions on an interposer canvas without any overlaps 2) Route the interconnections between transmitters and receivers on the chiplets according to the pre-defined netlists, ensuring no intersections occur 3) Routing paths are restricted to avoid traversing beneath chiplets, except the source and destination chiplets for each interconnection 4) The datarate between each transmitter and receiver pair is uniquely defined 5) Ensure the eye-opening for each netlist meets the eye mask specifications outlined in the UCIe[1] standards.

Given the increased complexity of chiplet designs, we propose to leverage machine learning. We formulate the complex problem of optimizing the search space of chiplet placement and routing as a hierarchical Markov decision process, which allows us to decompose the problem into manageable sub-tasks while maintaining the interdependencies crucial to finding optimal solutions.

As illustrated in Fig. 2, the chiplet placement and routing problem is modeled as a hierarchical Markov decision process (MDP), comprising two interlinked MDPs: one for chiplet placement and another for routing. This approach captures the sequential nature of the problem while maintaining the inter-dependencies between these two crucial stages.

Let the problem instance be denoted as $\boldsymbol{x}$, which consists of a fixed-sized interposer canvas with dimensions $(h_{\text{int}}, w_{\text{int}})$, a set of chiplets $C = \{c_1, c_2, c_3, ..., c_n\}$ to be placed, and a set of intercon-

[1]UCIe (Universal Chiplet Interconnect Express) is an open industry standard for high-speed, die-to-die interconnects between chiplets in advanced semiconductor packaging[25].

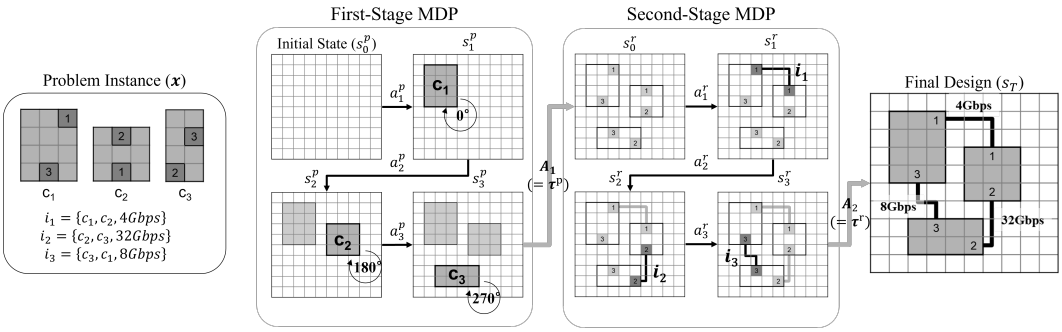


Figure 2: Hierarchical Markov decision process (MDP) of the chiplet placement and routing problem.

nection netlists $I = \{i_1, i_2, i_3, ..., i_k\}$. As illustrated in Fig. 1, each chiplet c_n is defined as an object with its height, width, and the location of transmitting (Tx) and receiving (Rx) ports according to the pre-defined netlists on itself. Each netlist $i_j = \{c_{\text{Tx}}, c_{\text{Rx}}, \lambda_j\}$ is defined with the transmitter chiplet, receiving chiplet, and datarate λ_j. The hierarchical MDP unfolds in two stages. In the first-stage MDP, dedicated to chiplet placement, the initial state (s_0^p) is set to the problem instance $\boldsymbol{x}$. This stage outputs placement trajectory τ^p with sequential actions ($\tau^p = (a_1^p, \ldots, a_n^p)$), resulting in a transition T^p to the next state ($s_0^r = T^p(s_0^p, \tau^p)$), which represents the post-placement configuration. The second-stage MDP, focused on routing, then takes the placement state (s_0^r) as its initial state and generates routing trajectory τ^r with sequential actions of routing ($\tau^r = (a_1^r, \ldots, a_k^r)$. These actions lead to the final state ($s_T = T^r(s_0^p, \tau^r)$) with routing transition T^r, representing the complete placement and routing configuration.

Upon completion of both stages, a reward $R = \sum_{i=1}^{k} \lambda_i \cdot |(\tau^r)^i|$ is calculated based on the final routing trajectories, reflecting the quality of the overall chiplet placement and routing solution in terms of electrical performance influenced by the netlist datarates λ_i. This hierarchical approach allows for a structured decision-making process that captures the sequential nature of placement followed by routing while maintaining the interdependencies between these two crucial stages. See Appendix B for further details of the hierarchical MDP.

3 Methodology

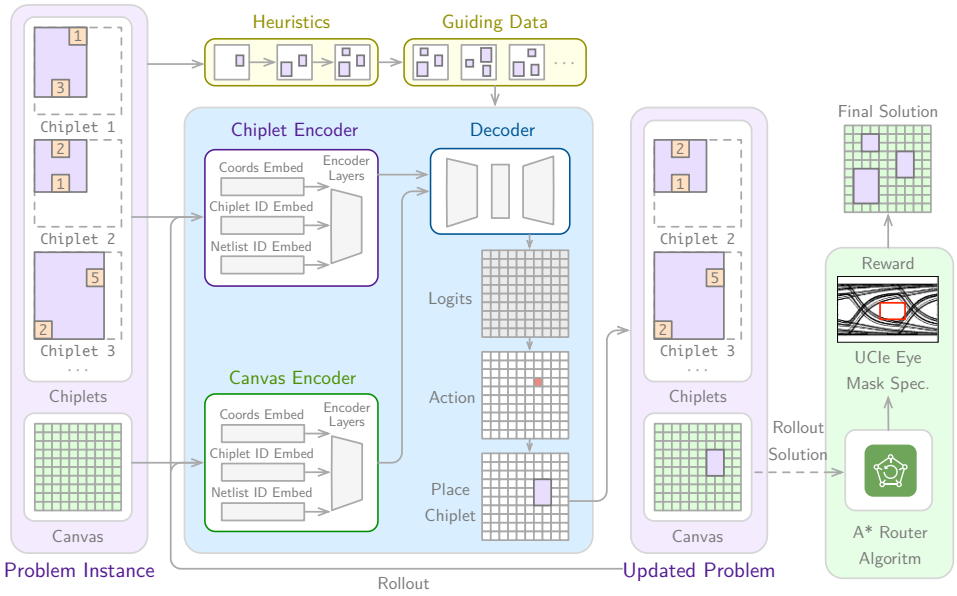


Figure 3: Overview of the Training Procedure with the ChipletFormer Architecture for Chiplet Placement and Routing Optimization.

We introduce ChipletFormer, a novel architecture that outputs a placement trajectory $\tau^p = (a_1^p, \ldots, a_n^p)$ by autoregressively producing placement actions a_t^p given the updated problem state s_{t+1}^p: $p(a_t^p \mid s_t^p)$. See Fig. 3 for abstract-level understanding. The initial state s_0^p consists of the problem instance with chiplets and the target canvas, while s_t^p represents the updated canvas with placed chiplets through action a_t^p. We employ two attention-based encoders to represent s_t^p: the Chiplet Encoder for representing chiplets and the Canvas Encoder for representing the canvas. Our decoder is a standard transformer decoder with context provided by the chiplet encoder and canvas encoder. Chiplet and Canvas, which contains useful information not only for placement but also for routing congestion using netlist information that can affect the entire optimization process (indicating that the encoder can learn representations for improved routing performance as well). We refer Appendix C for additional details on input representation and modeling.

Training of ChipletFormer We use imitation learning, training with the negative log-likelihood loss on a dataset of guiding demonstrations. Let $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, \tau^{p(i)})\}_{i=1}^N$ be our dataset of N problem instances and their corresponding guiding placement trajectories. Formally, our training process for parameters θ in ChipletFormer involves minimizing the following loss function $\mathcal{L}(\theta)$:

$$\mathcal{L}(\theta) := \mathbb{E}_{(\boldsymbol{x},\tau^p)\sim\mathcal{D}} \left[-\log p_\theta(\tau^p \mid \boldsymbol{x}) \right] \tag{1}$$

Here, $\boldsymbol{x}$ represents a problem instance (initial state), τ^p is the guiding trajectory of placement actions, and $p_\theta(\tau^p \mid \boldsymbol{x})$ is the probability assigned by our model to the trajectory τ^p given the initial state $\boldsymbol{x}$.

Rollout for Routing The performance after placement, provided by ChipletFormer, is finalized at the end of each episode by routing the netlists of placed chiplets using the A* algorithm. To prevent intersections, we implement a sequential routing strategy prioritized by the Manhattan distance between netlist pairs. These rollout results can potentially be used for training a model as ChipletFormer, representing promising future work.

4 Discussion

4.1 Preliminary Results

We trained ChipletFormer on the guiding dataset with rule-based heuristics as in Appendix D for 10 epochs with the Adam optimizer, learning rate of 10^{-4} and batch size of 64. We noted that the model correctly learned the general behavior of placing the chiplets in a clustered fashion around the center of the canvas. However, it often did not correctly place the corresponding ports close to each other, resulting in feasible placements (no overlaps) but several infeasible downstream routing.

4.2 Further works

While the model could place components much faster than heuristics (i.e., seconds compared to minutes or even hours), we believe the results can be greatly improved, and there are several directions to pursue further work. Firstly, we plan to determine the reason for the incorrect placement of ports, which we believe might be due to the inability of our architecture to model edges correctly. These are only paired by using the same embeddings at the moment. In such a sense, integrating powerful graph representations as in Lai et al. [11] could help the model correctly understand. Another promising direction is employing additional strategies for (self-)augmentation of the dataset, including action and domain invariances [9]. Finally, exploring online and offline reinforcement learning [1, 3] is another promising avenue for future work that could overcome the limitation of sub-optimal pre-collected datasets.

5 Conclusion

In this paper, we introduce a novel benchmark for chiplet placement and routing, and present ChipletFormer, a neural approach that captures the interdependencies of the two hierarchical tasks while addressing the stringent constraints of chiplet design. Our research not only provides a valuable benchmark but also highlights the potential of neural methods in solving interdependent design problems. As the industry shifts toward advanced chiplet architectures, we hope our work will help advance future innovations in semiconductor design and optimization.

References

[1] F. Berto, C. Hua, J. Park, L. Luttmann, Y. Ma, F. Bu, J. Wang, H. Ye, M. Kim, S. Choi, N. G. Zepeda, A. Hottung, J. Zhou, J. Bi, Y. Hu, F. Liu, H. Kim, J. Son, H. Kim, D. Angioni, W. Kool, Z. Cao, J. Zhang, K. Shin, C. Wu, S. Ahn, G. Song, C. Kwon, L. Xie, and J. Park. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. *arXiv preprint arXiv:2306.17100*, 2024. https://github.com/ai4co/rl4co.

[2] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, 8:225134–225180, 2020. doi: 10.1109/ACCESS.2020.3039858.

[3] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[4] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020. ISSN 2095-8099. doi: https://doi.org/10.1016/j.eng.2020.01.007. URL https://www.sciencedirect.com/science/article/pii/S2095809919306356.

[5] K. Cho, Y. Kim, H. Lee, H. Kim, S. Choi, S. Kim, and J. Kim. Design and analysis of power distribution network (pdn) for high bandwidth memory (hbm) interposer in 2.5 d terabyte/s bandwidth graphics module. In *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, pages 407–412. IEEE, 2016.

[6] K. Cho, Y. Kim, H. Lee, J. Song, J. Park, S. Lee, S. Kim, G. Park, K. Son, and J. Kim. Signal integrity design and analysis of differential high-speed serial links in silicon interposer with through-silicon via. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 9(1):107–121, 2018.

[7] S. W. Ho, M. Z. Ding, P. S. Lim, D. I. Cereno, G. Katti, T. C. Chai, and S. Bhattacharya. 2.5d through silicon interposer package fabrication by chip-on-wafer (cow) approach. In *2014 IEEE 16th Electronics Packaging Technology Conference (EPTC)*, pages 679–683, 2014. doi: 10.1109/EPTC.2014.7028352.

[8] R. Kaur, A. Asad, and F. Mohammadi. A comprehensive review of processing-in-memory architectures for deep neural networks. *Computers*, 13(7):174, 2024.

[9] H. Kim, M. Kim, F. Berto, J. Kim, and J. Park. Devformer: A symmetric transformer for context-aware device placement. 2023.

[10] M. Kim, H. Park, S. Kim, K. Son, S. Kim, K. Son, S. Choi, G. Park, and J. Kim. Reinforcement learning-based auto-router considering signal integrity. In *2020 IEEE 29th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 1–3. IEEE, 2020.

[11] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao, and P. Luo. Chipformer: Transferable chip placement via offline decision transformer, 2023. URL https://arxiv.org/abs/2306.14744.

[12] F. Li, Y. Wang, M. Lu, Y. Zhu, H. Wang, Z. Zhao, J. Huang, X. Wei, X. Liang, Y. Wang, H. Xu, H. Li, X. Li, Q. Liu, M. Liu, N. Sun, and Y. Han. The decomposition and combination paradigms of chiplet-based integrated chips. *Integrated Circuits and Systems*, pages 1–13, 2024. doi: 10.23919/ICS.2024.3451428.

[13] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(4):748–761, 2021. doi: 10.1109/TCAD.2020.3003843.

[14] X. Ma, Y. Wang, Y. Wang, X. Cai, and Y. Han. Survey on chiplets: interface, interconnect and integration methodology. *CCF Transactions on High Performance Computing*, 4(1):43–52, 2022.

[15] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean. A graph placement methodology for fast chip design. *Nature*, 594: 207–212, 06 2021. doi: 10.1038/s41586-021-03544-w.

[16] G. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86 (1):82–85, 1998. doi: 10.1109/JPROC.1998.658762.

[17] G. Mounce, J. Lyke, S. Horan, W. Powell, R. Doyle, and R. Some. Chiplet based approach for heterogeneous processing and packaging architectures. In *2016 IEEE Aerospace Conference*, pages 1–12, 2016. doi: 10.1109/AERO.2016.7500830.

[18] M. Odema, L. Chen, H. Kwon, and M. A. A. Faruque. Scar: Scheduling multi-model ai workloads on heterogeneous multi-chiplet module accelerators, 2024. URL https://arxiv.org/abs/2405.00790.

[19] S. Pal, D. Petrisko, R. Kumar, and P. Gupta. Design space exploration for chiplet-assembly-based processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(4): 1062–1073, 2020. doi: 10.1109/TVLSI.2020.2968904.

[20] H. Park, B.-I. Kim, D. G. Choi, H. Kyu Kim, T. Wook Kang, and Y. Jung Lee. Layer and length-deviation limit aware interposer routing for bend and wirelength minimization. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 14(6):993–1006, 2024. doi: 10.1109/TCPMT.2024.3393930.

[21] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–14, 2021.

[22] H. Sharma, S. K. Mandal, J. R. Doppa, U. Ogras, and P. P. Pande. Achieving datacenter-scale performance through chiplet-based manycore architectures. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[23] K. Son, S. Kim, H. Park, T. Shin, K. Kim, M. Kim, B. Sim, S. Kim, G. Park, S. Park, et al. Thermal and signal integrity co-design and verification of embedded cooling structure with thermal transmission line for high bandwidth memory module. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 12(9):1542–1556, 2022.

[24] R. Swaminathan. The next frontier: Enabling moore's law using heterogeneous integration. *Chip Scale Review*, pages 11–22, 2022.

[25] UCIe Consortium. Universal Chiplet Interconnect Express (UCIe) Specification. Technical specification, UCIe Consortium, July 2022. URL https://www.uciexpress.org/specification/.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

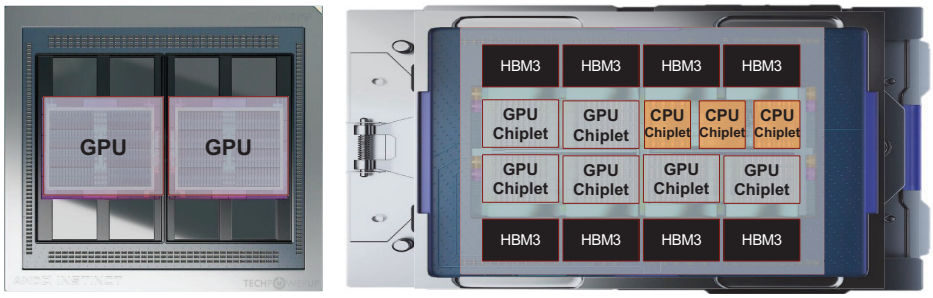
[27] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, C. Fuguet, I. Miro-Panades, G. Moritz, J. Durupt, C. Bernard, D. Varreau, J. Pontes, S. Thuries, D. Coriat, M. Harrand, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, Q. L. Meunier, A. Farcy, A. Arriordaz, S. Chéramy, and F. Clermidy. Intact: A 96-core processor with six chiplets 3d-stacked on an active interposer with distributed interconnects and integrated power management. *IEEE Journal of Solid-State Circuits*, 56(1):79–97, 2021. doi: 10.1109/JSSC.2020.3036341.

[28] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh. Modular routing design for chiplet-based systems. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 726–738, 2018. doi: 10.1109/ISCA.2018.00066.

[29] B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

A Further Background on Chiplets

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. A chiplet is a modular building block of a larger integrated circuit (IC) or system-on-chip (SoC). In 1965, Moore predicted that building large systems out of smaller, separately packaged, and interconnected functions might prove more economical [16]. 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 4: Comparison between the architectures of AMD MI250 and MI300

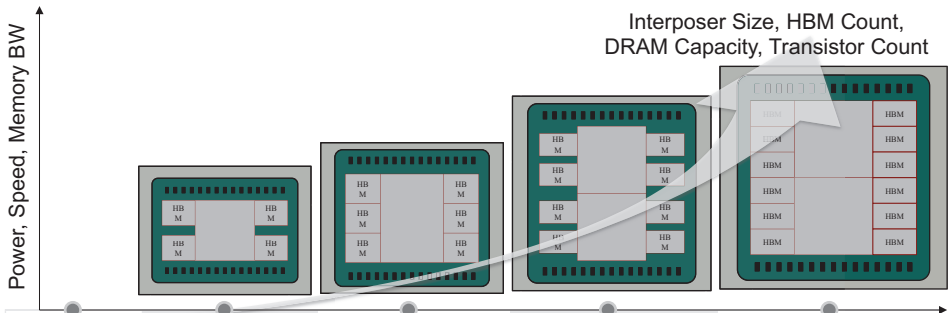
A recent example of a chiplet-based product is AMD's MI300 in Fig. 4, which integrates 9 TSMC 5nm chiplets and 8 high-bandwidth memory (HBM) modules. Compared to its predecessor, the MI250, the MI300 achieves an impressive 8× improvement in AI performance and a 5× improvement in AI performance-per-watt. As a result, the chipletization of systems-on-chip (SoCs) is expected to accelerate in the near future.

Model		**P100**	**V100**	**A100**	**H100**	**B100**
Year		2016	2018	2020	2022	2024
Process		16 nm	12 nm	7 nm	4 nm	4 nm (4NP)
# of SM		56 EA	80 EA	108 EA	144 EA	192 EA
Core type of SM		FP32, FP64	+ FP16 Tensor	+ FP32/64 Tensor + INT8 Tensor	+ INT8/4 Tensor + FP8 Tensor + Tensor Memory Accelerator	+ FP4/6 Tensor
Peak Performance	FP32	10.3 TFLOPS	15.7 TFLOPS	19.5 TFLOPS	73 TFLOPS	0.9 PFLOPS
	Tensor FP16	-	125 TFLOPS	312 TFLOPS	1080 TFLOPS	1.8 PFLOPS
	Tensor INT8	-	-	624 TOPS	2160 TOPS	3.5 POPS
	Tensor FP8	-	-	-	2160 TFLOPS	3.5 PFLOPS
	Tensor FP6	-	-	-	-	3.5 PFLOPS
	Tensor FP4	-	-	-	-	7 PFLOPS
L2 Cache		4 MB	6 MB	40 MB	60 MB	N/A
Memory	Capacity	16GB HBM2	32GB HBM2	80GB HBM2E	96GB HBM3	192GB HBM3e
	Bandwidth	720 GB/s	900 GB/s	2039 GB/s	4110 GB/s	8000 GB/s
	# of HBM	4 EA	4 EA	6 EA	6 EA	8 EA
Max Power		300 W	300 W	400 W	700 W	700 W

Table 1: Historical Development Roadmap of NVIDIA GPUs

With the exponentially increasing AI model size, there has been great demand for AI accelerators that offer enhanced performance, greater memory capacity, and higher bandwidth. As illustrated in Table 1, the number of streaming multiprocessor (SM) cores in a single GPU has risen significantly, from 56 EA in the P100 model to 192 EA in the latest B100 NVIDIA GPU module. Additionally, the integration of High Bandwidth Memory (HBM) has expanded, with the number of HBMs integrated increasing from 4 EA in the P100 to 8 EA in the B100, to meet the higher memory capacity and bandwidth requirements of large-scale AI models. This trend suggests that, if AI accelerator modules are fabricated using chiplet packaging, the number of chiplets integrated will increase substantially in future designs.

Moreover, as NVIDIA GPUs are designed for general-purpose computing, the diversity of core types they support has also broadened. By employing chiplet-based packaging for AI accelerators, the core types and quantities can be customized to suit specific application requirements. Likewise, memory capacity and bandwidth can be optimized, enabling tailored solutions for various AI workloads.



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

Figure 5: Interposer roadmap by TSMC

As shown in Fig. 5, not only have the number of SM cores and HBM stacks increased, but the size of the interposer has also expanded, driven by advancements in Chip-on-Wafer-on-Silicon (CoWoS) technology, primarily led by TSMC. This trend aligns with the hypothesis that the number of chiplets integrated onto a single silicon interposer is expected to grow exponentially. In addition to the increasing number of chiplets, further "chipletization"—where larger chiplets are subdivided into smaller units—is anticipated, leading to a substantial rise in the overall chiplet count.

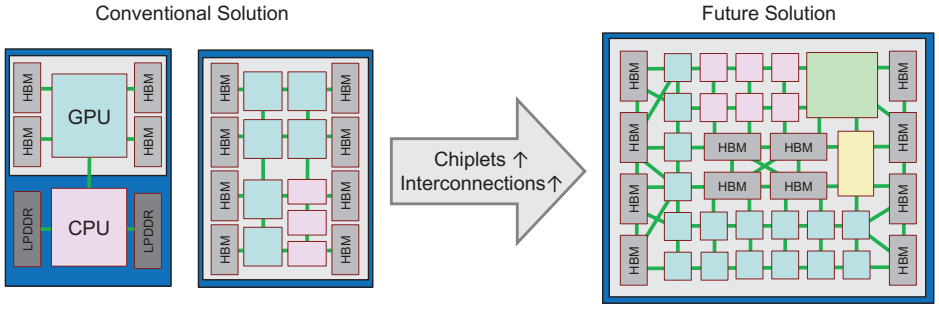


Figure 6: Future chiplet-based architecture with increasing number of chiplets and interconnections

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. 6, 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 traditional chip placement and chiplet placement may seem similar, they differ significantly in terms of problem contexts, rewards, and constraints. Chip placement involves positioning macros and standard cells within a chip (i.e., at the on-chip level), whereas chiplet placement refers to arranging chiplets (sliced chips) on an interposer package (i.e., at the off-chip level). Chiplet placement must adhere to multiple constraints, arising from various electrical requirements and specifications such as signal integrity (SI), power integrity (PI), and thermal integrity (TI). In contrast, chip placement primarily focuses on optimizing the trade-off between wirelength and congestion [13]. Additionally, in chiplet placement, the location of ports (i.e., the sources or destinations of netlist interconnections) on the chiplets is predetermined, meaning the rotation of the chiplets can significantly impact routing quality.

B Detailed Hierarchical MDP of Chiplet Placement and Routing

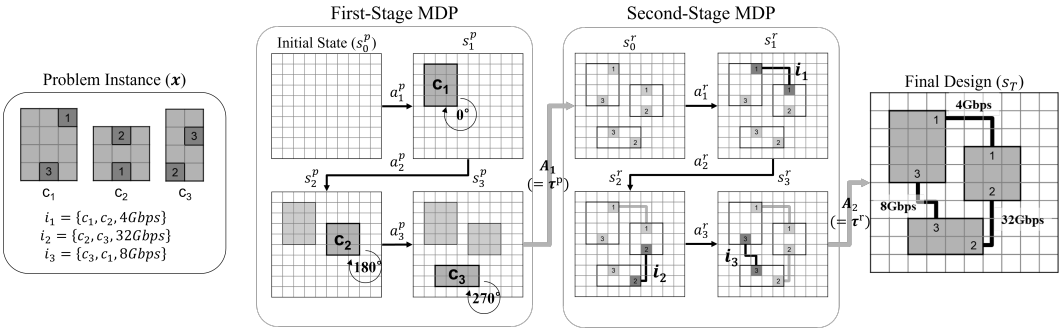


Figure 7: Hierarchical Markov decision process (MDP) of the chiplet placement and routing problem.

First-Stage MDP for chiplet placement consists of state, placement actions, and transition, n denotes for the number of chiplets.

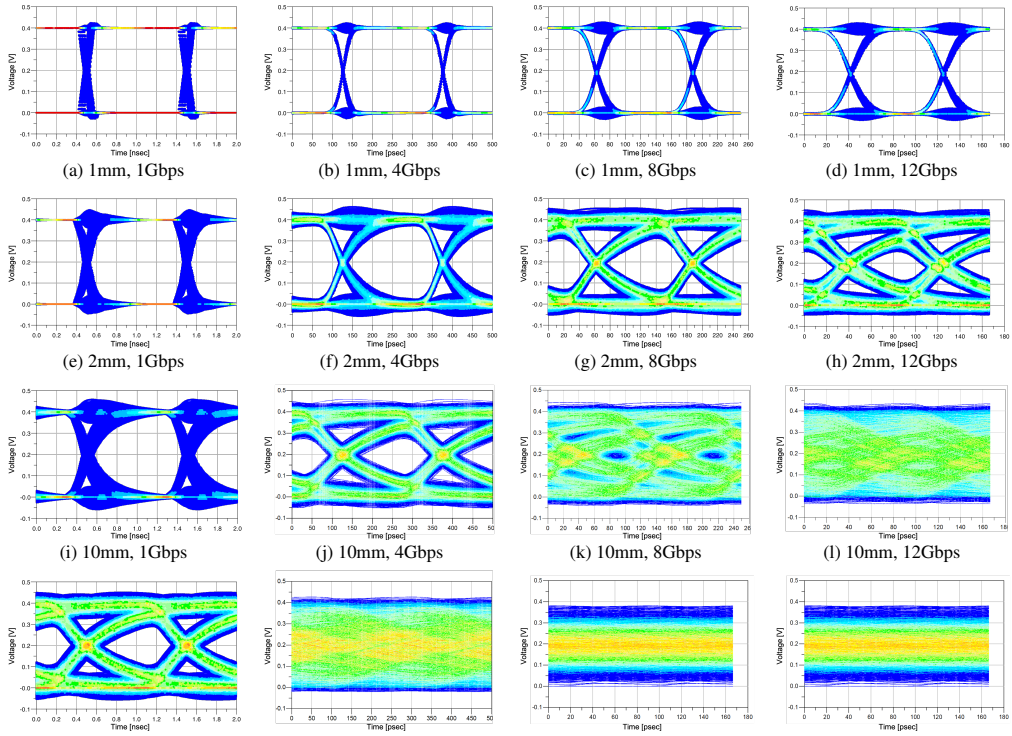
- **Initial State (s_0^p)**: An empty interposer canvas of size w_{int} and h_{int}.
- **Intermediate State (s_t^p)**: The partial placement configuration after t chiplets have been placed.
- **Final State (s_n^p)**: The complete placement configuration after all n chiplets have been placed.
- **Action (A_1)**: A set of first-stage sub-actions and is a trajectory of sub-actions $A_1 = \tau^p = \{a_1^p, a_2^p, a_3^p, ..., a_n^p\} = a_{1:n}^p$. Each sub-action represents the placement of a single chiplet $a_t^p = (x_t, y_t, \theta_t)$, where $(x_t, y_t) \in \{(1,1), ..., (h_{int}, w_{int}\} \setminus s_{t-1}^p$ are the coordinates for placement, making sure no overlaps are allowed. $\theta_t \in \{0°, 90°, 180°, 270°\}$ is the rotation angle of the chiplet.
- **Transition (T^p)**: This function $s_0^r = T^p(s_0^p, \tau^p)$ represents the post-placement configuration.

Second-Stage MDP for netlist routing consists of state, routing actions, transitions, and reward, k denotes for the number of netlists.

- **Initial State (s_0^r)**: The final placement configuration from the first-stage MDP.
- **Intermediate State (s_t^r)**: The partial routing configuration after k netlists have been routed.
- **Final State (s_T)**: The complete placement and routing configuration after all k netlists have been routed s_k^r.
- **Action (A_2)**: A set of second-stage sub-actions and is a trajectory of routing path $A_2 = \tau^r = \{a_1^r, a_2^r, a_3^r, ..., a_k^r\} = a_{1:k}^r$ Each sub-action represents the routing of a single netlist $a_t^r = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$, where $(x_i, y_i) \in \{(1,0), ..., (h_{int}, w_{int})\} \setminus s_{t-1}^r$ are the coordinates of points along the route and m is the number of points along the route.
- **Transition (T^r)**: This function $s_T = T^r(s_0^r, \tau^r)$ represents the post-routing configuration, where the chiplets are placed and routed and is ready to calculate reward.

Reward of the hierarchical MDP is the weighted average routing distance of the netlists. Each netlist has a weight depending on the datarate of the interconnect.

- **Reward:** $R = \frac{1}{k} \sum_{i=1}^{k} \lambda_i \cdot |(\tau^r)^i|$
- λ_i is a datarate-dependent hyperparameter, determined by the electrical simulation-based look-up table. The eye diagram simulation was carried out for interconects with varying length and varying datarate. Then, they were evaluted based on the eye mask specification of UCIe standard.



(a) 1mm, 1Gbps (b) 1mm, 4Gbps (c) 1mm, 8Gbps (d) 1mm, 12Gbps

(e) 2mm, 1Gbps (f) 2mm, 4Gbps (g) 2mm, 8Gbps (h) 2mm, 12Gbps

(i) 10mm, 1Gbps (j) 10mm, 4Gbps (k) 10mm, 8Gbps (l) 10mm, 12Gbps

(m) 20mm, 1Gbps (n) 20mm, 4Gbps (o) 20mm, 8Gbps (p) 20mm, 12Gbps

Figure 8: Eye diagrams of interconnects with varying lengths and datarates.

B.1 Electrical Simulation-based Reward Function

The reward function in our study incorporates both the routing distance and the pre-defined datarate of the interconnect. This approach acknowledges that the overall electrical performance is influenced not only by the length of the interconnect but also by the datarate at which it operates. Consequently, simply minimizing the total routing distance in the chiplet package does not guarantee optimal system performance.

To address this complexity, we conducted electrical simulations to evaluate the eye diagrams of routed channels at various lengths and datarates. An eye diagram is a visual representation of a digital signal, created by superimposing multiple bit periods of the signal. It provides crucial information about signal quality, including timing variations, amplitude variations, and noise.

As reported in Fig. 8, we assessed the quality of these eye diagrams by measuring their eye aperture, which represents the open area of the eye pattern. The eye aperture is a key indicator of signal integrity, with a larger aperture generally indicating better signal quality. Eye diagrams like Fig. 8 (a) is considered widely open, while Fig. 8 (p) is completely closed. We then compared these measurements against the eye mask specifications defined by the UCIe standard, which stipulates minimum requirements of 40mVpp (millivolts peak-to-peak) for vertical eye opening and 0.75UI (Unit Interval) for horizontal eye opening.

The eye mask specification is evaluated by overlaying a predefined mask on the eye diagram. If any part of the signal intrudes into the mask area, it fails to meet the specification. This method ensures that the signal maintains sufficient amplitude and timing margins for reliable data transmission. The results of the eye mask evaluation is reported in Fig. 10 in terms of UI, the time interval that indicates the eye opening and Fig. 9 describes how the eye aperture is measured.

As shown in Fig. 10, signal quality deteriorates as interconnection length increases. Similarly, higher data rates also result in degraded signal quality. On top of that, different interconnection lengths exhibit varying rates of signal degradation as data rates increase. Based on our experimental results, we developed a look-up table for the hyperparameter λ, which assigns weights to specific data rates

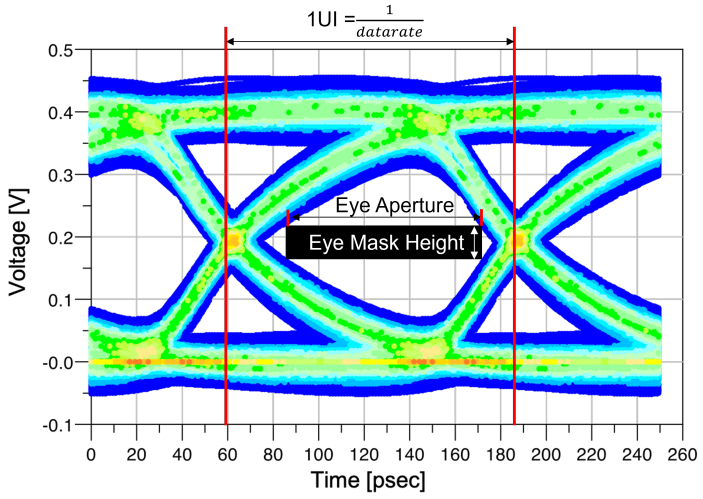


Figure 9: Evaluation of Eye Diagram in terms of Eye Mask Aperture.

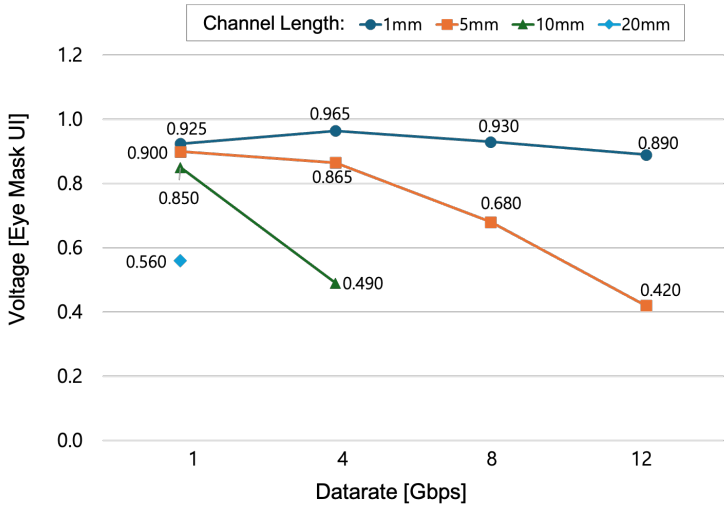


Figure 10: Eye Mask Evaluation of Interconnects with Varying Lengths and Datarates

and is applied to the routing length. We aim to construct this look-up table such that higher data rates are given more priority to ensure eye mask specification compliance.

By conducting these comprehensive analyses, we were able to more accurately assess the viability and performance of various routing solutions, taking into account both physical layout and electrical signal integrity constraints.

C CHIPLETFORMER: Input Representation and Model

C.1 Input Representation

CHIPLETFORMER's input comprises two main components: the current canvas state and unplaced chips.

Canvas State. Represented as a tensor $X_c \in \mathbb{R}^{B\times N^2\times 4}$, where B is the batch size, N^2 is the total number of cells in the square canvas, and the 4 features for each cell are:

1. $[x, y]$: Coordinates of each cell
2. Chip ID: ID of the placed chip (0 for empty cells)
3. Port ID: Port number of the chip (0 for empty cells)

Unplaced Chips. Represented as a tensor $X_u \in \mathbb{R}^{B\times C\times K\times 5}$, where C is the number of unplaced chips, K is the maximum size of any chip (e.g., 64 for an 8×8 chip), and the 5 features are:

1. $[x, y]$: Coordinates representing chip dimensions
2. Chip ID: ID of the unplaced chip
3. Port ID: Port number of the chip
4. Current chip indicator: Binary flag (1 for current chip to be placed, 0 otherwise)

C.2 Model Architecture

The main architecture is represented in Fig. 3.

C.2.1 Canvas Encoder

The canvas encoder projects features of the current canvas state X_c into the latent dimension D, i.e. Canvas Encoder: $Z_c \in \mathbb{R}^{B\times N^2\times 4} \rightarrow Z_c \in \mathbb{R}^{B\times N^2\times D}$

Table 2: Components of the Canvas Encoder

Component	Layer Type	Dimension (in,out)
Coordinate Embedding	Linear	$(2, D/2)$
Chip ID Embedding	Embedding	$(\text{max\_chips} + 1, D/4)$
Port ID Embedding	Embedding	$(\text{max\_chips} + 1, D/4)$
Canvas Projection	Linear	(D, D)

C.2.2 Chiplet Encoder

The Chiplet Encoder projects features of the current (remaining) state X_u into the latent dimension D – this contains crucial information for determining the next location of the current chip at hand: Chiplet Encoder: $Z_u \in \mathbb{R}^{B\times(C\times K)\times 5} \rightarrow Z_u \in \mathbb{R}^{B\times(C\times K)\times D}$

Table 3: Components of the Chiplet Encoder

Component	Layer Type	Dimension (in,out)
Coordinate Embedding	Linear	$(2, D/2)$
Chip ID Embedding	Embedding	$(\text{max\_chips} + 1, D/4)$
Port ID Embedding	Embedding	$(\text{max\_chips} + 1, D/4)$
Unplaced Chip Projection	Linear	$(D + 1, D)$

C.2.3 Input Projection Layer

After passing through the respective decoders, we then pass encoded features to the projection layer is the concatenation along the dimension 1 (right after the batch dimension B) of the outputs of both encoders:

$$Z = [Z_c; Z_u] \in \mathbb{R}^{B\times(N^2+C\times K)\times D}$$

C.2.4 Transformer Decoder

The transformer decoder is adapted from Vaswani et al. [26] and processes the input representation $Z \in \mathbb{R}^{B\times(N^2+C\times K)\times D}$ and outputs an updated representation $Y \in \mathbb{R}^{B\times(N^2+C\times K)\times D}$, maintaining the same dimensionality. The decoder consists of L layers, each composed of the following key components:

Multi-Head Attention (MHA) Multi-head attention allows the model to jointly attend to different parts of the input sequence with multiple attention mechanisms (or "heads"). Each attention head is computed by performing scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K, and V are the query, key, and value matrices, respectively, and d_k is the dimensionality of the key vectors. The input Z is projected into multiple sets of query, key, and value vectors for each attention head:

$$Q = ZW_i^Q, \quad K = ZW_i^K, \quad V = ZW_i^V$$

The outputs from all attention heads are concatenated and linearly transformed:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, and W^O is the output projection matrix.

Feed-Forward Network (FFN) The FFN is applied independently to each position in the sequence. It consists of two linear transformations with a ReLU activation in between, which allows the model to introduce non-linearity and capture more complex patterns:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Here, W_1, W_2, b_1, and b_2 are learnable parameters.

Layer Normalization (RMSNorm) RMSNorm (Root Mean Square Normalization) is used to normalize the input vectors, ensuring stable training and helping the network handle the scale of the inputs [29]. RMSNorm is computed as:

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} \cdot \gamma$$

where x is the input vector, n is the dimensionality of x, and γ is a learnable scaling parameter.

Full Transformer Pass Each transformer layer applies the above components in sequence with residual connections. The updated hidden state y for each layer is computed as follows:

$$x' = x + \text{MHA}(\text{RMSNorm}(x)) \tag{2}$$

$$y = x' + \text{FFN}(\text{RMSNorm}(x')) \tag{3}$$

Here, x is the input to the layer, x' is the intermediate output after attention, and y is the final output after the FFN.

Output Projection Layer The output of the transformer encoder, $Y \in \mathbb{R}^{B\times(N^2+C\times K)\times D}$, is then passed through a linear projection layer to produce the final output $O \in \mathbb{R}^{B\times(N^2+C\times K)}$. This is implemented using a linear transformation:

$$O = \text{Linear}(D, 1)$$

The output is then appropriately masked to ensure valid chip placements and connections, accounting for any constraints in the task. For instance, nodes greater than N^2 in their index are always masked, since decision-making happens on the canvas.

Complete Model: CHIPLETFORMER. The complete CHIPLETFORMER model is built by sequentially applying the input projection, transformer encoder, and output projection layers:

$$\text{CHIPLETFORMER}(X) = \text{OutputProj}(\text{TransformerEncoder}(\text{InputProj}(X)))$$

Here, X represents the raw input data (canvas and chiplet features), which is first processed by the input projection layer to map the inputs to the same latent space. The transformer encoder then captures spatial relationships and dependencies, and finally, the output projection layer computes the valid chip placements and outputs the final decision.

C.3 Hyperparameters

Table 4 summarizes the key hyperparameters used in the CHIPLETFORMER architecture.

Hyperparameter	**Value**
Model dimension (D)	128
Number of attention heads (h)	8
Number of transformer layers (L)	3
Canvas size ($N \times N$)	50×50
Maximum number of chips	10
Maximum chip dimension	8×8

Table 4: CHIPLETFORMER Hyperparameters

D Guiding Data Collection with Rule-based Heuristics

To mitigate the extensive combinatorial solution space inherent in the chiplet placement and routing problem, we employ imitation learning. This approach leverages near-optimal guiding data to replace the exploration phase required for policy parameterization, potentially accelerating training convergence. The guiding data is obtained through a rule-based heuristic method.

In this heuristic approach, the problem instance, consisting of a set of chiplets and netlists, is represented as a graph. Within this graph, each node corresponds to a chiplet, and each edge represents a netlist connection between two chiplets. This graph representation allows to figure out a closed loop in the given chiplet and netlist combinations and is used to determine the order of chiplet placement. Here's the procedure of the heuristic method which has three phases: (1) chiplet placement ordering, (2) chiplet rotation and (3) chiplet placement.

(1) Chiplet Placement Ordering: The process of determining the chip placement order involves a systematic approach based on the structure and connectivity of the graph representing the chip arrangement. The method can be summarized as follows:

Cycle Analysis and Initial Chip Selection: Cycles within the graph are analyzed to identify loops where chips are interconnected. The chip with the highest number of edges (i.e., the highest degree) is selected as the initial chip for placement. This chip is chosen for its central role in the graph, providing a robust starting point for subsequent placements.

Priority Queue Creation: A priority queue is constructed to determine the placement order of the remaining chips. Priority is assigned based on several criteria:

- Direct Connection to Initial Chip: Chips directly connected to the initial chip receive higher priority.
- Inclusion in Cycles with Initial Chip: Chips that are part of the same cycle as the initial chip are given intermediate priority.
- Both Connection and Cycle Inclusion: Chips that are both connected to the initial chip and included in the same cycle are given lower priority.
- Vertex Degree: Chips with a higher degree (more connections) receive a higher priority within the same level of priority.

Selection and Ordering: Chips are then selected from the priority queue based on their assigned priorities. Chips with higher priorities are placed earlier in the order. For chips with the same priority, the size of the chips is considered to break ties, with larger chips being placed earlier.

Final Placement Sequence: The placement order is constructed by sequentially adding chips from the priority queue, starting with the highest priority and moving to lower priorities. This approach ensures that chips are placed in a sequence that optimizes connectivity and minimizes potential conflicts.

(2) Chiplet Rotation: The determination of chiplet directions involves aligning the orientation of each chiplet to ensure optimal connectivity and minimal signal misalignment. This process is carried out as follows:

Initial Placement and Netlist Order: Chips are placed according to the predetermined placement order. Netlists, which define the connections between chips, are then processed in sequence based on their IDs. The goal is to ensure that chips are oriented correctly to match the direction of their connections as specified by the netlists.

Rotation and Alignment:

- Identify Rotation Needs: For each netlist, the chips involved (transmitter and receiver) are checked to determine if they need to be rotated to align their ports correctly. This is based on the netlist's specified port directions.
- Determine Opposite Sides: The orientation of each chip is compared to the target port direction. The opposite side of a port direction is used to determine the required rotation. For example, if a port is on the 'top' side, it should align with the 'bottom' side of the connected chip.

- Apply Rotation: If a chip's current orientation does not match the required direction, it is rotated accordingly. The rotation is performed based on predefined mappings that specify the degree of rotation needed to achieve the desired alignment.

Process and Adjust Remaining Chips:

- Handling Unprocessed Chips: Chips that have not yet been processed are evaluated in the context of their remaining connections. If a chip needs to be rotated to match the alignment of an already placed chip, this rotation is performed based on the netlist connections.
- Verify Final Alignment: After all rotations are applied, the final alignment is checked to ensure that all chips are correctly oriented and that connections are properly aligned. Misaligned nets, if any, are identified and reviewed.

Misalignment Check: Identify and Report Misaligned Nets: Any connections that remain misaligned after the final rotations are identified. Misaligned nets are reported, and their sources and destinations are documented to assess potential adjustments or corrections.

(3) Chip Placement: To place chips optimally, the heuristic method incorporates several key strategies and criteria to ensure the best configuration. Here's a detailed breakdown of the process used to achieve optimal chip placement:

Initialization: The environment is defined with a canvas of specified dimensions where chips will be placed. The placement order and directions of chiplets, determined earlier, guide the placement process.

Initial Placement:

- Place First Chip: The first chip in the placement order is placed at a central or predefined position on the canvas. If the initial placement is invalid (e.g., overlapping with existing placements), the method searches for the nearest valid location.

Heuristic Placement Strategy:

- Search for Valid Locations: The method scans potential locations on the canvas to place the chip. Each candidate location is evaluated for validity, ensuring no overlap with already placed chips and compliance with spatial constraints.
- Calculate Distances: The method calculates the Manhattan distance between the chip's ports and the ports of already placed chips. The goal is to minimize this distance to optimize the connectivity and overall performance of the chip configuration.

Placement Adjustment:

- Adjust Locations: If a valid placement cannot be found or if the initial placement does not yield optimal results, the method adjusts the placement by considering alternative positions or reconfiguring previous placements.
- Maximize Reward: A reward function evaluates each placement based on factors such as minimized distances between connected ports, adherence to constraints, and overall configuration efficiency. The method aims to maximize the total reward.

Validation and Constraints:

- Check for Overlaps: Each placement is checked for overlaps with other chips. This ensures that the spatial arrangement does not violate constraints and that the chips fit within the canvas boundaries.
- Verify Connectivity: The method verifies that the placement of each chip maintains proper connectivity with previously placed chips, based on the netlists and required connections.

Optimization and Final Selection:

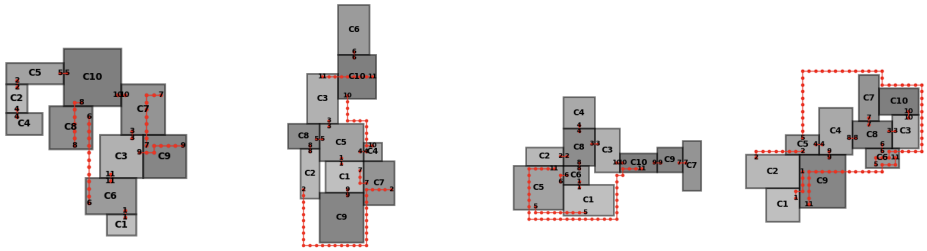
- Evaluate Configurations: After multiple iterations, the configuration with the highest total reward is selected. This configuration represents the optimal placement in terms of connectivity, spatial constraints, and overall performance.

- Select Best Placement: The final placement is chosen based on the best achievable reward, reflecting the most efficient and effective chip configuration.

D.1 Hyperparameters of Guiding Data Collection

We collected a total of 1,620 guiding data using the rule-based heuristic method described above. Of these, 1,420 were used for training, 100 for validation, and 100 for testing. Each data was generated through 100 iterations of the heuristic method. Each problem instance consists of 10 chiplets and 11 netlists, placed on a 50x50 grid canvas, representing a silicon interposer in real-world applications.

Illustrations of collected guiding data are shown in Fig. 11.



(a) Reward: -4.82 (b) Reward: -6.64 (c) Reward: -6.18 (d) Reward: -9.09

Figure 11: Results of the Chiplet Placement and Routing by Rule-based Heuristics