ENGINEERING Information Technology & Electronic Engineering
https://www.academax.com/EITEE; https://www.engineering.org.cn
https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=11344065
E-mail: eng.itee@zju.edu.cn

Engineering Journals
Available on IEEE Xplore®

Research Article
https://doi.org/10.1631/ENG.ITEE.2025.0008

WSC optimizer: an optimization tool for wafer-scale chip architecture exploration

Wenbo ZHANG[1], Bo DING[2], Shuai WEI[1]✉, Qinrang LIU[3], Hong YU[1], Ke SONG[1], Wei GUO[1], Bo MEI[1], Rui ZHENG[1]

[1]*Information Engineering University, Zhengzhou 450001, China*
[2]*Songshan Lab, Zhengzhou 450001, China*
[3]*Institute of Big Data, Fudan University, Shanghai 200433, China*

Abstract: In recent years, mature advanced packaging technologies have increasingly enabled the integration of multiple small dies into larger chips, while retaining chip-scale density and high-bandwidth interconnects. To address the inefficiencies of manual design and the challenges of heterogeneous optimization in wafer-scale chip (WSC) development, we systematically explore key factors in WSC architecture design. We integrate chip layout, operator mapping, and hardware–software co-design, and formulate the WSC architecture exploration problem as a multi-objective optimization task. First, we establish a hierarchical architecture model for WSCs, unifying the quantification of core constraints and interconnect topology constraints; second, we propose a hierarchical multi-objective collaborative optimization framework to jointly optimize physical constraints and task mapping communication patterns; finally, we develop a WSC optimizer toolchain that supports mixed-granularity simulation and generates optimal configurations for representative workloads. Experimental results demonstrate that compared with traditional computer architectures, the optimized architectures generated by our WSC optimizer achieve up to a 22× throughput improvement and a 5× latency reduction in application domains, such as cryptographic decryption and signal processing.

Key words: Wafer-scale chip; Hardware–software co-design; Chip layout; Design space exploration

1 Introduction

Recent years have seen the rapid development of technologies, such as large-scale language models (LLMs), resulting in an exponential growth in computational demand (Zhu et al., 2025). This trend is strikingly epitomized by the remarkable expansion in model parameters, which have escalated from millions in bidirectional encoder representations from Transformers (BERT) to a staggering 1.8 trillion in GPT-4, a growth of over a thousand times (Brown et al., 2020; Achiam et al., 2023; Chowdhery et al., 2023; Touvron et al., 2023; Yenduri et al., 2024). Concurrently, training costs have escalated dramatically. For instance, GPT-4 has up to 1.8 trillion parameters (Baktash and Dawodi, 2023; Patel and Wong, 2023). At a cloud cost of roughly $1 per hour using A100, the training cost for a single run would amount to approximately $63 million (Patel and Wong, 2023). This growth trend is expected to continue, as large models continue to demonstrate superior performance in natural language understanding and content generation tasks (Raffel et al., 2020; Zhang SS et al., 2022; Baktash and Dawodi, 2023).

As microelectronics manufacturing technology approaches fundamental physical limits (IEEE, 2024), traditional single-chip integration faces significant size constraints, leading to reduced yields and increased costs (Bohr, 2009; Han et al., 2024). To address these challenges, the wafer-scale chip (WSC) (Burns et al., 2006) has emerged as a promising solution. This technology leverages advanced packaging and innovative chiplet design methods to integrate numerous processing cores within a single package (Shao et al., 2019), thereby enhancing computational density and intercommunication bandwidth. Furthermore, WSC offers a high degree of design flexibility, allowing customization of functional modules based on specific

✉ Shuai WEI, weis0906@163.com
Wenbo ZHANG, https://orcid.org/0009-0000-6542-9797
Shuai WEI, https://orcid.org/0000-0002-4902-6645
CLC number: TP336

Received Aug. 30, 2025; Revision accepted Dec. 5, 2025;
Crosschecked Feb. 28, 2026; Published online: Mar. 16, 2026

application requirements. This ensures an optimal balance between cost-effectiveness and manufacturing efficiency, while preserving superior performance.

Meanwhile, as microelectronic fabrication technology approaches the fundamental 1-nm physical threshold, the traditional paradigm of relying solely on process node scaling for performance enhancement has become increasingly untenable. In response to this technological impasse, semiconductor manufacturers have adopted a parallelization strategy, transitioning from single-core architectures to heterogeneous multi-core designs. This architectural evolution is exemplified by NVIDIA's graphics processing unit (GPU) architectures (Markidis et al., 2018), which integrate thousands of streaming multiprocessors (SMs) as computational units, and Intel's Core microarchitecture series (Hammarlund et al., 2014), which features sophisticated multi-core configurations with shared cache hierarchies. However, as die sizes approach the reticle limit of extreme ultraviolet (EUV) lithography systems, this scaling approach has led to significant challenges in defect density management. This, in turn, has resulted in diminished yield rates and escalating manufacturing costs (Han et al., 2024).

The chiplet design methodology (Shao et al., 2019), which decomposes complex systems into small, independent modules and uses advanced packaging technologies to enable efficient interconnections between these modules, has emerged as a crucial approach to addressing the limitations of monolithic integration, such as physical size constraints and yield issues. This method not only resolves these limitations but also allows for flexible integration of diverse functional units tailored to specific application scenarios, thereby providing the most cost-effective solutions across a wide range of applications. This flexible integration of functional units can be adapted to the requirements of different scenarios, optimizing cost and production efficiency without compromising performance.

Die-to-die interconnects (Turner et al., 2018) and high-density 3D stacking (Loh et al., 2007) enable high core integration within a single package, improving energy efficiency by shortening the length of the wire, reducing latency, and increasing data throughput. Wafer-scale technologies, exemplified by Cerebras's WSE and Tesla's DOJO (Talpes et al., 2022), demonstrate significant advancements in computational density, on-chip memory, and communication bandwidth. For instance, the Cerebras's WSE-3 integrates 4 trillion transistors and 900 000 cores on a 46 225 mm^2 silicon die, delivering 125 quadrillions floating point operations per second (PFLOPS) of compute power, 44 GB of on-chip static random access memory (SRAM), unprecedented memory bandwidth (21 PB/s), and inter-communication bandwidth (214 PB/s). These features make it ideal for deep learning workloads. In a similar vein, Tesla's DOJO integrates 25 custom D1 chips, with each chip offering 10 TB/s of internal bandwidth and 36 TB/s of tile-to-tile communication, achieving a peak performance of 9 PFLOPS. These examples underscore the viability and benefits of wafer-scale integration for high-performance computing.

Researchers developed a wafer-scale GPU system that achieves 18.9× performance gains and 143× energy efficiency compared to traditional printed circuit board-based GPUs (Pal et al., 2019). They also designed a 2048-chiplet, 14 336-core wafer-scale processor (Pal et al., 2021), proving its applicability not only for artificial intelligence (AI) workloads but also for broader high-performance computing tasks. These advancements highlight the potential of wafer-scale integration in performance and energy efficiency, offering a promising solution to modern computational demands and enabling more powerful AI systems. However, the current wafer-scale design methods primarily rely on homogeneous architectures, leading to issues such as energy inefficiency, uneven resource utilization, and high design and manufacturing complexity (Hu et al., 2024).

In this context, Wu Jiangxing's software-defined system on wafer (SDSoW) introduces an innovative approach (Wu et al., 2024). As shown in Fig. 1, by using silicon wafers as packaging substrates and integrating computing cores, memory cores, and interconnect cores through micro-bumps, the system significantly enhances the integration density and performance. However, the current WSC design remains heavily reliant on manual engineering processes, requiring meticulous consideration of chiplet selection, layout optimization, task partitioning, and performance evaluation (Xu et al., 2025). Key challenges include interconnect topology design, signal integrity assurance, and thermal management between cores, all of which demand precise optimization and verification (Li FP et al., 2022). To address these limitations and facilitate the broad adoption of wafer-scale technology, it is critical to develop a comprehensive and efficient set of tools for WSC architecture design exploration. This study is organized as follows:

Sections 1 and 2 systematically explore key concepts in WSC design, analyze limitations of existing technical solutions, and present motivations for improvements. Section 3 delves into chiplet abstraction and its corresponding wafer layout strategy. Section 4 integrates the chiplet model with a standardized operator library, proposing a WSC architecture design method based on hardware–software co-design, including reinforcement learning-driven iterative optimization and constraints-based chiplet combination. Section 5 verifies the methodology's effectiveness and superiority through specific application experiments. Finally, Section 6 summarizes the research findings and outlines future work directions.

2 Background and motivation

2.1 Chiplet design method

The chiplet design technology refers to an advanced packaging solution integrating multiple dies into a single substrate.

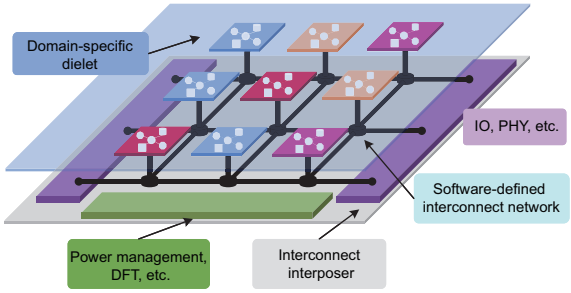

Fig. 1 Schematic diagram of the software-defined system on wafer (IO: input/output; PHY: physical layer; DFT: design for testability)

The substrate may be organic or silicon, and the integration is achieved through high-density interconnections, enabling the dies to function collectively as a single chip. This technology offers numerous benefits, such as increased design flexibility and shorter time-to-market, but it also incurs additional costs. In the realm of chiplet design methods, several strategies have been proposed to address various challenges and optimize system performance.

The general interposer architecture (GIA) method employs a linear programming algorithm to address the core selection problem and uses a simulated annealing algorithm to optimize the core layout, thereby enabling automatic task-oriented selection and placement (Li FP et al., 2022). However, this method does not consider core reuse, inter-task pipeline operations, or storage requirements, which remain critical challenges in advanced chiplet designs.

In contrast, Floorplet creates well-defined functional and specific chiplets as foundational building blocks (Chen SX et al., 2024). These chiplets are then evaluated using a gem5-based (Binkert et al., 2011) simChiplet simulator to analyze application workload, communication patterns, and memory hierarchy for layout optimization. Although this approach provides a detailed and flexible framework, it is sensitive to initial layout configurations and can suffer from a long simulation time for large-scale systems.

Further enriching this field, Ahmad et al. (2022) explored the balance between cost and yield in multi-node heterogeneous integration, providing insights into optimizing economic efficiency and manufacturing reliability. Similarly, Feng and Ma (2022) proposed a quantitative cost model to help designers understand the financial implications of architectural choices, ensuring both performance and economic viability. Additionally, Zhang JM et al. (2024) demonstrated how fine-grained mapping can accelerate AI computations, highlighting the potential of chiplet technology in high-performance computing. Works focusing on multi-package co-design and fixed-outline floor planning (Zhuang et al., 2022) contribute to more efficient system-level optimization, addressing key challenges in signal integrity, power distribution, and thermal management.

2.2 Automated design framework and design tools

The AHA Agile Hardware Project at Stanford University has developed the Halide compiler, a domain-specific language and compiler for optimizing image processing and compute-intensive workloads, enabling agile hardware–software co-design. The AHA project team employs the Halide compiler (Deng et al., 2022) to decompose applications into computing, storage, and interconnection modules, subsequently mapping them into Verilog code to support iterative update designs. In contrast, NVIDIA's MAGNet tool (Venkatesan et al., 2019) generates synthesizable register transfer level (RTL) code and efficient mapping schemes for neural networks by leveraging three-level architectural templates to provide an automatic optimization framework. Li ZS et al. (2017) proposed a comprehensive software–hardware co-design framework, which captures parallelism in irregular applications and aggressively schedules pipelined execution on reconfigurable platforms. DSAGen (Weng et al., 2020) is a separable spatial architecture generator that uses primitive components to form flexible, connected architecture description graphs (ADG). Despite its moderate scale, DSAGen is well-suited for both general compilation and design space exploration, providing valuable insights for WSC design.

2.3 Challenges

The system-on-wafer (SoW) design method presents three fundamental challenges: (1) layout optimization under specific constraints; (2) task scheduling; (3) development of design toolchains. First, existing methods prioritize single-constraint dimension optimization. They fail to effectively address the challenge of multi-physical constraints. These strategies not only neglect signal integrity, thermal distribution, and stress balance, but also exhibit high sensitivity to initial layout configurations, resulting in $\mathcal{O}(n^2)$ time complexity in design space exploration for large-scale systems (here, n is the number of chips). Second, current solutions lack adaptability to heterogeneous task characteristics. They do not fully consider inter-task data dependencies, differences in computational intensity, and dynamic changes in resource requirements, making it difficult to match the adaptation needs between heterogeneous tasks and chiplet resources. Third, existing electronic design automation (EDA) toolchains lack a unified modeling method for WSC, which hinders efficient mapping from tasks to operators. Although current EDA toolchains, such as GARNet (Deng et al., 2022), MAGNet (Venkatesan et al., 2019), and DSAGen (Weng et al., 2020), attempt to mitigate these issues through graph neural network-based topology synthesis and polyhedral compiler techniques, they often increase design complexity. Additionally, the design space exploration mechanisms are insufficiently adaptable, failing to support efficient and automated design of WSC.

This study draws on the design idea of flexible packaging for SDSoW, follows the design concept of structure adapted to the application, and starts from the application while considering WSC process constraints, area constraints, and other constraints.

The contributions of this study are summarized as follows:

1. We propose a chiplet combination method and a hierarchical WSC construction method that integrates physical and design constraints. Through the design workflow of chiplet selection → component integration → wafer layout, we standardize the logic for functional and interconnection adaptation, thereby providing a foundational framework that balances feasibility and flexibility for SoW implementation.

2. We present a reinforcement learning-based evaluation and iterative optimization method. First, we develop mathematical models for chiplets and different applications, and compile physical parameters of chiplets and operator-specific attributes of applications to establish a standardized operator library. Second, to address the challenge of differentiated optimal scheduling across diverse layout results, we realize collaborative iteration of layout and scheduling within the reinforcement learning framework. This approach dynamically adapts to the matching requirements of chiplet and applications, efficiently explores optimal architecture configurations, and overcomes the trade-off between layout and scheduling

exploration space.

3. We propose a collaborative hardware–software co-design methodology for WSC. We comprehensively account for the coupled relationships between bandwidth, latency, storage capacity, throughput, and task execution time. Through the coordinated design of operator clustering, task mapping, and pipeline scheduling, we achieve dynamic balance across multiple performance metrics. Meanwhile, we clarify the rationale for prioritizing SRAM in WSC architectures: SRAM's low access latency and high reliability fulfill the system's demand for fast data response, and its tight integration with computing chiplets minimizes interconnection overhead. In contrast, DRAM, employed as a discrete chiplet, exhibits inherent drawbacks of large area cost and high cross-chiplet transmission latency, which conflict with the core design objectives of high-density integration and low-latency communication in WSC systems.

3 Model description and operator scheduling

To address the aforementioned challenges, this section presents a systematic design approach for WSCs. The core research objectives encompass system layout optimization, high-performance task scheduling, dedicated design toolchain development, operator clustering generation, a reinforcement learning-based optimization framework, and hardware–software co-design. These objectives collectively aim to achieve efficient architecture exploration and performance optimization for WSCs.

3.1 Application classification and evaluation indicators

Depending on the application's sensitivity to latency (Ali et al., 2021), tasks can be divided into two categories: real-time applications (Tatar et al., 2024) and non-real-time applications (Leon et al., 2023, 2024; Panousopoulos et al., 2024). It is critical to understand the characteristics of these two types of applications for designing efficient and reliable WSC architectures.

3.1.1 Real-time applications

Signal processing, autonomous driving (Tatar et al., 2024), and medical monitoring (Chakaravarthy et al., 2021) are typical time-sensitive real-time applications requiring data processing within strict timeframes. For example, a signal processing task involves pipeline operations, such as pulse compression, space–time adaptive processing (STAP), and constant false alarm rate (CFAR), where 20 parallel data groups must be fused and target-detected within a specified window; in in-vehicle systems (Tatar et al., 2024), timely target detection is equally critical—any timeout compromises accuracy, reliability, and the overall performance. These traits require the WSC optimizer to prioritize low latency and synchronization reliability, adapt to heterogeneous chiplet scheduling and high-bandwidth transmission, and thus effectively verify its layout optimization and dynamic resource allocation capabilities under specific time constraints.

3.1.2 Non-real-time applications

In contrast, non-real-time applications—such as message-digest algorithm 5 (MD5), video conferencing (Leon et al., 2024), surveillance systems, and information service systems (Leon et al., 2023)—can tolerate certain latency without compromising functionality. These general computing scenarios have no strict latency restrictions, with their core requirement being to improve task throughput via multi-chiplet parallelism. For example, mild frame delays in videoconferencing only reduce output quality rather than disabling the system. Such characteristics allow the WSC optimizer to prioritize chiplet density and interconnection efficiency in layout design, verifying its ability to optimize throughput and resource utilization under weak time constraints, as well as its adaptability to general computing chiplet combinations.

3.1.3 Throughput

In information service systems, throughput refers to the number of transactions processed by a system or server per unit of time. It reflects the system's ability to process requests and is a key indicator to evaluate the system's performance. In this study, the throughput of the application is used as the primary objective of system optimization. Specifically, for real-time applications, the execution time of individual tasks is considered a constraint. By strictly controlling the execution time of each task, all tasks are ensured to be completed within the specified time; for non-real-time applications, the execution time threshold of individual tasks is set as an upper bound. Through reasonable task scheduling, the system throughput is maximized while ensuring the quality of service.

3.1.4 Crucial aspects of WSC design

In WSC design, thermal management, yield enhancement, and fault tolerance are critical to addressing the challenges of high-density integration. To align with WSC's inherent demand for reliable, high-performance operation, we adopt a symmetrical and balanced layout strategy that mitigates stress concentration from component placement, laying a foundational framework for yield improvement; we integrate fault-tolerant design in the development process by refining task scheduling and resource allocation, and leverage our proposed reinforcement learning (RL) framework to mitigate faulty-node impacts, enabling the identification of robust scheduling strategies that physically isolate critical tasks from failure regions.

Based on a general-purpose design philosophy, we select two typical tasks—MD5 and signal processing—that represent non-real-time and real-time application scenarios, respectively. Meanwhile, we construct a WSC architecture supporting flexible resource configuration through the combination of general-purpose chiplets. This design is particularly well-suited for compact high-performance computing scenarios: such devices require strict compliance with size and weight constraints while delivering high computational throughput. The inherent high-density computing and storage capabilities of WSC, coupled with the architectural design proposed in this study, effectively resolve the trade-off between computing power and form factor in traditional solutions, accurately meeting the dual requirements of portability and high performance for field-deployed

or space-constrained systems.

3.2 WSC design architecture

The proposed WSC architecture aims to address the complex challenge of integrating multiple cores into a single chip (as shown in Fig. 2). Adopting a layered and progressive design strategy, it not only ensures efficient design and optimization at each stage, but also effectively reduces the overall design complexity. Among its key features, the integration of independent switch components stands as a core design decision: by decoupling routing/switching functions from compute chiplets, it not only avoids the rigidity limitations and high redesign costs of domain-specific integrated architectures, but also supports the direct reuse of general-purpose chiplets. This not only enhances the system's flexibility in adapting to various workloads but also enables efficient inter-component communication.

3.2.1 Cores

Core types are divided into computing, storage, and switching variants; their key physical attributes—size, power consumption, and storage capacity—are summarized in Table 1.

3.2.2 Components

The components are essential in the hierarchical wafer-scale architecture, linking cores and establishing efficient communication paths while respecting physical limits such as wafer-scale stress distribution. Internally, each component must feature a uniform, symmetric cell layout, and its footprint must stay within four times the reticle size (such as 26 mm×33 mm).

3.2.3 Wafer-scale chip

In the context of the wafer-scale layout stage, employing a systematic design methodology can significantly streamline the process by breaking it down into manageable stages. Otherwise, the design space becomes excessively broad, which makes it challenging to identify feasible solutions and leads to inefficient allocation of search time on ultimately unviable options. A progressive strategy for WSC design begins with the core

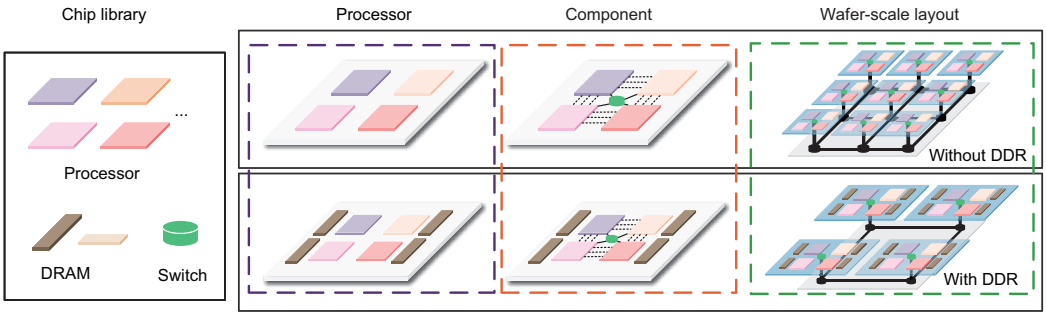

Fig. 2 An example of a hierarchical WSC design method

Table 1 Nomenclature

Notation	Meaning
$C=\{c_i \mid 1\le i\le m\}$	The set of processors
$V=\{v_i \mid 1\le i\le n\}$	The set of tasks
$E=\{e_{i,j} \mid 1\le i,j\le n, i\ne j\}$	The set of edges with $e_{i,j}$ being the edge between task i and task j
$W=\{w_{i,j} \mid 1\le i,j\le n, i\ne j\}$	The set of bandwidths with $w_{i,j}$ being the edge between processor i and processor j
$M=\{m_i \mid 1\le i\le n\}$	The set of task storage requirement with m_i being the storage required for task i
$Q=\{q_{i,j} \mid 1\le i,j\le n, i\ne j\}$	The set of inter-processor communication data with $q_{i,j}$ being the communication data between processor i and processor j
W_i, H_i	The width and height of processor core i
P_i	The power consumption of processor core i
T, R	The execution time interval and the computational capacity
$D_{\mathrm{O}}, D_{\mathrm{R}}$	The delay of sending and receiving data (without DRAM)
$D_{\mathrm{S}}, D_{\mathrm{W}}$	The delay of switching and transmitting
MM, SM	The capacity of DRAM and SRAM
M_i, C_j	Core i and component j
C_{K}	The operator time ratio of SRAM and DRAM
S_{c}, S_{w}	The area of the component and wafer
L_{chip}, $L_{\mathrm{component}}$	Minimum spacing between chips and components

processing elements and components, and gradually progresses to integration at the wafer level. This staged approach ensures that each component is optimized before moving on to the next stage, thereby enhancing the overall design efficiency and reducing unnecessary exploration of impractical solutions. The proposed design concept adopts the layer-by-layer approach and includes elements such as cores, processors, components, and WSCs.

3.3 Task partitioning and scheduling

3.3.1 Processor

The selected core serves as the initial input to the framework. The processor primarily consists of computing and memory cores. Each computing core is capable of connecting to memory cores. However, the relative positions of the computing core and memory core must remain fixed due to interface constraints, such as the requirement that the connection lines between the central processing unit (CPU) and memory be as short as possible and of equal length. This fixed positioning allows the entire processing element to be treated as a single entity during the space exploration phase of the design process. In traditional design architectures, each processing element is usually equipped with a large capacity of off-chip DRAM, so there is no additional consideration for its storage requirements. However, the performance of computation using DRAM is significantly lower than that of on-chip SRAM because of the long latency of off-chip data transfer. To this end, we introduce C_{K} as a metric to quantify the execution efficiency of operators on SRAM and double data rate SDRAM (DDR).

Meanwhile, the findings (Jung et al., 2022; Chen YW et al., 2024) revealed that the majority of mature dies are equipped with integrated SRAM, while DRAM typically functions as a discrete die, necessitating its consideration as an independent entity about area and cost. SRAM is predominantly employed as a cache for the compute die. The reason is that although SRAM is not cost-competitive, its low-latency characteristics are critical to improving system performance. In scenarios where the rapid access and processing of data is imperative, the use of SRAM has been shown to enhance system responsiveness and overall performance to a considerable degree. Consequently, the utilization of SRAM is deemed the optimal choice for this study.

3.3.2 Components

To achieve the desired functionality, it is customary to integrate a switch inside the components. This facilitates efficient data transfer while taking into account physical constraints such as the stress distribution of the wafer system. The layout should be designed to be uniform and symmetric, and the area of the component should generally not be too large. The maximum area of the component is subject to the constraint of satisfying a 4-fold photomask area (26 mm×33 mm).

WSCs typically adopt homogeneous components throughout the entire wafer to balance manufacturing cost and yield. The design paradigm is illustrated in Fig. 2. This method stresses high-throughput links among components. In the conventional architecture, DRAM is attached to each processing element with a large capacity, but the off-chip data transfer delay affects the computational performance compared to the on-chip SRAM. For simulation approximation purposes, a ratio C_{K} is assumed to represent the relative efficiency of executing operators in SRAM and DDR. This assumption makes it more accurate to model the impact of the memory hierarchy on the overall chip performance at the wafer level, highlighting the importance of efficient on-chip memory utilization.

3.3.3 Compilation

During compilation, the front-end takes in the application algorithm's description. It then performs dependency analysis and abstracts the algorithm into operators of varying granularities. However, if the operator graph is overly fine-grained, the scheduling space can become too large, making it harder to find a good scheduling strategy. Additionally, excessive graph complexity can reduce its effectiveness as a scheduling tool. Traditional development methods demand a lot from compilation front-ends, requiring them to directly translate programs into suitable operator graphs. In contrast, using domain-specific languages (DSL) offers a more efficient approach. Front-end tools can efficiently convert DSL into operator graphs, ensuring compatibility with existing applications. Multi-level intermediate representation (MLIR) is a framework for compilers and toolchains, aiming to provide a flexible foundation for a wide range of compiler optimizations and transformations. As shown in Fig. 3, the MLIR approach generally includes several key phases and elements.

3.3.4 Multi-level intermediate representation

The high-order intermediate representation (IR) module subsequently optimizes the computation graph. Then, the code optimization and transformation module converts the low-order IR and performs hardware parameter transformation. The memory management module optimizes data access to generate IR for specific target platforms. Finally, the code generation module produces executable code for the hardware, which is executed by the target machine. To address the portability issue for heterogeneous wafer-chip-based platforms, the

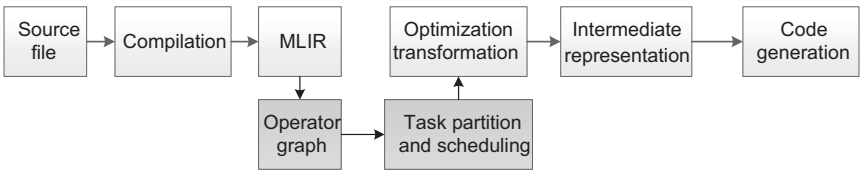

Fig. 3 Application compilation framework

compilation scheduling module uses a scalable heterogeneous parallel compilation framework. This includes application programming interfaces, intermediate representations, heterogeneous code generation modules, and task scheduling modules. Applications developed with this framework's interface can generate executable programs for platforms with diverse hardware, such as CPUs, GPUs, and reconfigurable devices, enabling efficient execution on WSCs.

3.3.5 Operator graph

As shown in Fig. 4, a simple operator graph consists of four operators, denoted as v_1, v_2, v_3, and v_4. In WSC, different tasks exhibit different performances across various execution times due to micro-architecture design, cache size, and compiler-generated code optimization (Table 2 presents an example of task execution time on different chips). Following the example shown in Fig. 4, the critical path L is $1 \to 3 \to 4$. In this example, operator v_1 is mapped to core A, and operator v_3 is mapped to core C. It can be demonstrated that a direct path exists between cores A and C. Employing an analogous calculation method, the transmission delay from operator v_1 to operator v_3 can be determined as

$$D_{13} = D_{\mathrm{O}} + D_{\mathrm{R}} + \frac{2q_{13}}{W_{\mathrm{b}}} + D_{\mathrm{W}}. \tag{1}$$

Similarly, the transmission delay between operator v_3 and operator v_4 can be determined as

$$D_{34} = D_{\mathrm{O}} + D_{\mathrm{R}} + \frac{2q_{34}}{W_{\mathrm{b}}} + D_{\mathrm{W}}. \tag{2}$$

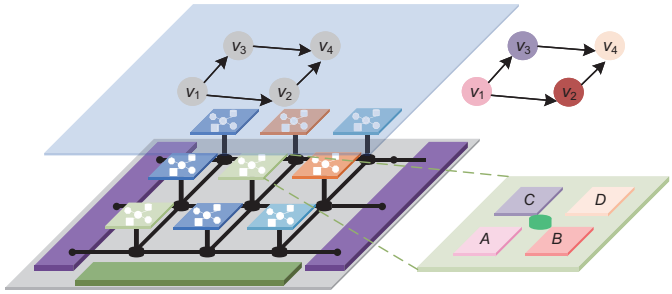

Fig. 4 Task graph mapping relationship and chiplet resource allocation in a wafer-scale system

Table 2 An example of task execution time on four chips

ID	Task execution time (ms)			
	chip_0	chip_1	chip_2	chip_3
0	75	83	59	64
1	48	29	64	46
2	54	19	44	83
3	15	56	98	98
4	71	89	71	67
5	61	13	46	69
6	47	21	83	89
7	79	75	14	68
8	44	73	20	52
9	12	66	27	49

3.3.6 Task partition and scheduling

After forming the operator graph, a direct mapping to the processing elements can be established, creating the corresponding mapping relationships. T_{ij} represents the corresponding execution time of operator i mapped to processing element j. Based on the execution time of each operator cluster and the transmission delay, scheduling algorithms, such as modulo scheduling, can be used to schedule tasks. Two scheduling methods are considered: conventional and pipelined. In conventional scheduling, data are processed only after reaching the execution units. In contrast, pipelined scheduling enables execution units to perform tasks and store data concurrently, effectively hiding storage latency and optimizing performance. For pipelined scheduling, inequalities (3)–(5) must be satisfied, and the link bandwidth must handle more bytes in time T than the processing element can process, as shown in Table 1.

$$M_{\mathrm{s}} \geq m_i + R_i, v_i \in V, \tag{3}$$

$$M_{\mathrm{s}} \geq m_i + R_i \left\lceil \frac{D}{T} + 1 \right\rceil, v_i \in V, \tag{4}$$

$$W_{\mathrm{b}} \geq R/T, \tag{5}$$

where $\lceil \cdot \rceil$ denotes the smallest integer that is greater than or equal to the enclosed value.

Inequalities (3)–(5) outline the constraints on storage and bandwidth: inequality (3) ensures that storage M_{s} meets the sum of task storage m_i and communication data R_i, inequality (4) requires storage M_{s} to handle m_i, R_i, and the ratio of delay D to execution time T, and inequality (5) ensures that bandwidth W_{b} can transfer communication data R within time T.

4 A framework for WSC architecture design method

4.1 Overview of the automatic design framework

This study aims to develop an efficient automated design framework for WSC architectures, enabling specific applications to achieve high performance. As illustrated in Fig. 5, the process starts with compiling the application program to abstract operators into task graphs. Then, operators are selected from the operator library, and their information is extracted for component layout and expansion. Subsequent steps include determining the interconnect topology, bandwidth allocation, operator clustering, and mapping strategies. Throughout this process, evaluating key metrics such as throughput and latency is critical. Since decisions on chiplet selection, placement, and bandwidth topology are interdependent and affect task scheduling methods, the framework adopts a cyclic iterative approach to optimize these parameters. Specifically, feedback loops facilitate iterative optimization, where design outcomes are continuously refined using a deep learning model. This ensures that the final WSC design achieves an optimal balance between performance and cost, maximizing the overall efficiency.

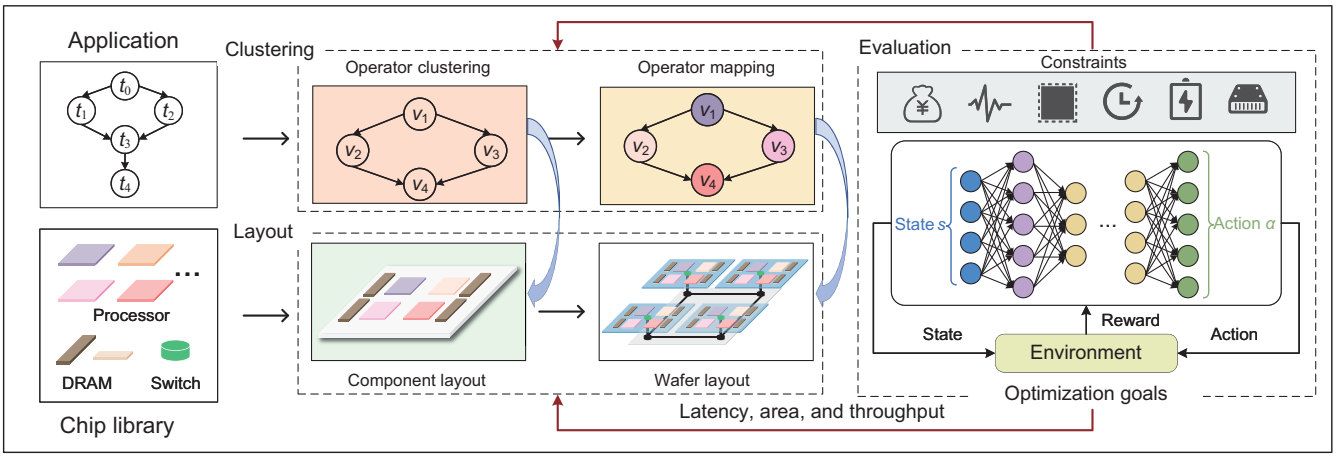

Fig. 5 A comprehensive optimization framework for operator clustering and task mapping in WSC systems

4.1.1 Core selection

Core selection is critical to the efficiency of WSC, as outlined in Section 2. Components on a wafer typically adopt a homogeneous design, with two primary architecture generation schemes. The first involves users copying devices and associated layouts based on conventional rack computing architectures. The second enables users to select a range of devices—such as multiple processors, various interconnect topologies, or different interconnect bandwidths, which are then automatically optimized by the architecture generator. This flexibility ensures the selection of the most suitable configuration tailored to specific application requirements.

4.1.2 Layout stage

The objective of layout optimization is to arrange as many components as possible on the wafer while adjusting the position and rotation angle of the cores to minimize area. Irregular placement has been shown to cause issues such as warping and local overheating. Therefore, it is crucial to carefully consider the connection relationships between components. The processing core should be as close as possible to the storage core to minimize latency and avoid crossing connections. The optimization of layout is intended to enhance the overall system performance and reliability.

In pursuing the above layout optimization goals, it is crucial to represent and optimize the layout scheme efficiently and accurately. Traditional layout representation methods often have limitations in handling large-scale complex systems and fail to meet the demand for efficient optimization. Therefore, this study introduces the sequence pair representation method (Tang et al., 2001) to provide a more effective solution for layout optimization problems. To efficiently handle the fixed contour constraints in WSC design, we draw on the methods of Zou et al. (2016) in fixed contour layout planning. By using the simulated annealing strategy, we combine rotation, movement, and swap perturbations to provide a solution for WSC design.

A sequence pair is composed of two sequences X and Y, each containing n elements that represent n blocks. The sequence pair (X, Y) can be expressed as shown in Eq. (6), where x_i and y_i denote the positions of block i in sequences X and Y, respectively.

$$(X, Y) = (\langle x_1, x_2, \ldots, x_n\rangle, \langle y_1, y_2, \ldots, y_n\rangle). \tag{6}$$

By computing the longest common subsequence (LCS) of the sequence pair, we can determine the coordinates of each block and the layout dimensions as follows:

$$x_i = \mathrm{LCS}(X_1, Y_1), \quad y_i = \mathrm{LCS}(X_2^{\mathrm{R}}, Y_1), \tag{7}$$

where X_1 and Y_1 represent the subsequences before block i in the sequence pair (X, Y). X_2^{R} is the reverse of subsequence X_2. For example, the layout information represented by the sequence pairs $(\langle a, c, d, b, e\rangle, \langle c, d, a, e, b\rangle)$ is shown in Fig. 6, where each element has the following dimensions: $(\langle 8, 4\rangle, \langle 2, 3\rangle, \langle 4, 5\rangle, \langle 4, 3\rangle, \langle 4, 6\rangle)$.

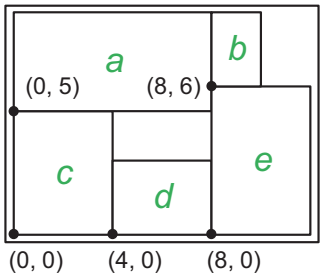

Fig. 6 An example of sequence-pair layout representation with corresponding module coordinate positioning

In WSC design, we adopt a rigorous hierarchical design approach. For each component, the following constraints must be strictly satisfied:

$$\max\left(\sum_{i=0}^{m} x_i + L_{\mathrm{chip}}\right) \cdot \max\left(\sum_{j=0}^{m} y_j + L_{\mathrm{chip}}\right) \le S_{\mathrm{c}}. \tag{8}$$

It is crucial to ensure that the replication of each component, along with the interval $L_{\mathrm{component}}$ between them, does not exceed the specified effective wafer area S_{w}.

4.1.3 Operator clustering

To facilitate mapping onto computing resources, the operator graph undergoes clustering, which aligns the number of nodes in the clustered graph (VJ) with the number of computing units in the component. This process preserves the original dependency relationships between operators, streamlining resource allocation and enhancing task execution efficiency. The clustering is implemented through a structured three-step approach, and the resulting VJ must satisfy the following core conditions:

$$\forall(v_i \in \mathrm{VJ}_k),\ v_i \in V, \tag{9}$$

$$\forall(v_i \in V),\ v_i \in \bigcup_k \mathrm{VJ}_k, \tag{10}$$

$$\forall\left(e_{i,j}, v_i \in \mathrm{VJ}_k, v_j \in \mathrm{VJ}_l\right),\ \exists e_{k,l}. \tag{11}$$

Specifically, conditions (9) and (10) ensure comprehensive partitioning of DAG, while condition (11) maintains the integrity of original dependencies throughout this partitioning process.

4.1.4 Task mapping

Following the formation of the operator cluster graph, a direct mapping to the processing element can be established to define the corresponding mapping relationship. In WSC design, different tasks exhibit different performances across various execution units due to micro-architecture design, cache size, and compiler-generated code optimization (Table 2 shows the difference in execution time of a task on different chips). The operator mapping strategy aims to minimize the overall task completion time by allocating interdependent task operators to chips with different computational capabilities. This involves determining the topological ordering of the operators, which defines the execution sequence, and the specific compute chip to which the operators should be assigned. These two key factors make up a multi-objective optimization problem.

4.1.5 Evaluation

The performance metrics of the system are derived via the performance evaluation (PE) stage in our framework (lines 2–4 of Algorithm 1). These metrics form the core of the current solution state, which guides the RL-based optimization loop (lines 5–15): after generating a new solution, we re-execute the PE stage to update metrics. These metrics align with the formulas in Eqs. (12) and (13), where N denotes the number of switch nodes, D_{O} and D_{R} represent the delay of sending and receiving data (without DRAM) respectively, D_{S} and D_{W} represent the delay of switching and transmitting respectively, and M_i and C_j represent the number of cores and groups respectively (the definitions of parameters are also given in Table 1).

$$\mathrm{Latency} = D_{\mathrm{O}} + D_{\mathrm{R}} + \frac{2R}{W_{\mathrm{b}}} + (N-1) \times D_{\mathrm{S}} + N \times D_{\mathrm{W}}, \tag{12}$$

$$\mathrm{Throughput} = \frac{\sum M_i \sum C_j}{L_{\mathrm{total}}}, i \in M, j \in C. \tag{13}$$

Algorithm 1 Pseudocode of the proposed framework

Input: ChipInfo, DAGInfo;
Output: layout and mapping solution;
/* generate the initial solution */
1: generate the initial layout and mapping solution;
/* performance evaluation (PE) stage */
2: $P, L \leftarrow$ calculate the critical path and length;
3: TS, TH, TP $\leftarrow$ calculate the latency, throughput, and power;
4: CurrentState = $\{\mathrm{TH}, \mathrm{TS}, \mathrm{TP}, P, L\}$;
/* RL-based action selection */
5: **for** $i \leftarrow 0$ to n_{iters} **do**
6: Action = model(CurrentState, ChipInfo, DAGInfo);
7: new solution $\leftarrow$ Action(layout, mapping);
8: **if** satisfy layout constraint & mapping constraint **then**
9: turn to the PE stage and calculate the cost;
10: $\mathrm{cost}_{\mathrm{new}} = \mathrm{cost}_{\mathrm{calculated}}(\mathrm{TH}, \mathrm{TS}, \mathrm{TP}, L)$;
11: **if** $\mathrm{cost}_{\mathrm{new}} < \mathrm{cost}_{\mathrm{old}}$ **then**
12: accept the new solution;
13: **end if**
14: **end if**
15: **end for**
16: **return** layout and mapping solution

4.1.6 Decision model

A decision-making model based on deep learning and reinforcement learning is employed. Core selection, bandwidth selection, and topology selection are regarded as integral steps in the process, similar to moves in a chess game. Each step presents a range of options, and a deep residual network is used to evaluate the current situation and suggest the next action. When searching reaches a leaf node, the final results, such as throughput, are used as rewards to propagate back to each node.

4.2 The proposed reinforcement learning framework

As shown in Fig. 7, the proposed reinforcement learning framework combines a graph convolutional network (GCN) with neural networks to optimize chip layout and scheduling. As shown in Eqs. (14) and (15), the GCN comprises two layers: the first layer with parameters $\boldsymbol{W}^{(1)}$ for the initial feature extraction, and the second layer with parameters $\boldsymbol{W}^{(2)}$ for deeper feature extraction. The output of this process is an $(m \times 128)$-dimensional feature matrix $\boldsymbol{Z}$, which integrates global node information. This feature matrix is seamlessly incorporated into the operator neural network, ensuring that the weight parameters of the GCN are optimized along with the operator neural network during the optimization process. The input model first converts the task graph into a disjunctive graph, generating an $(m \times (n+2))$-dimensional feature matrix and an $(m \times m)$-dimensional adjacency matrix. These matrices include information such as chip execution time, finish time, and schedule status. For simplicity, assume $m = 10$ (the number of operators) and $n = 4$ (the number of chips).

$$\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}, \tag{14}$$

$$\boldsymbol{Z} = \mathrm{softmax}(\tilde{\boldsymbol{A}} \cdot \mathrm{ReLU}(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(1)})\boldsymbol{W}^{(2)}), \tag{15}$$

where $\boldsymbol{A}$ is the adjacency matrix, $\tilde{\boldsymbol{A}}$ is the normalized adjacency matrix, $\boldsymbol{D}$ is the indegree matrix, and $\mathrm{ReLU}(\cdot)$ is the rectified linear unit activation function. $\boldsymbol{X}$ is a matrix for incorporating the information of adjacent nodes.

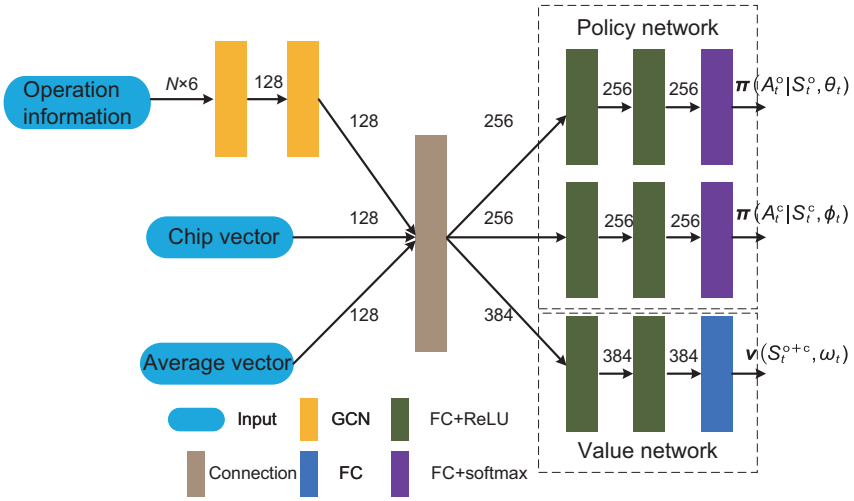

Fig. 7 Schematic of the reinforcement learning framework (FC: fully connected layer)

Then, the GCN processes these inputs to produce an $(m\times128)$-dimensional feature matrix that captures global node information. During feature preprocessing, candidate operator features are isolated and combined with an average vector to form refined inputs for the operator neural network. The operation network then takes this input and outputs a probability distribution over candidate operators. Based on these probabilities, the operation selection step chooses an operator, which is used for both gradient descent updates and as input to the core neural network. Finally, the value network evaluates stored samples to update the model under the guidance of a specific loss function, as shown in Eqs. (16)–(18):

$$\text{Operator\_Loss} = \text{mean}\left(-\log\left(\text{o\_probs}\right)\cdot D_{\text{r}}\right), \tag{16}$$

$$\text{Chip\_Loss} = \text{mean}\left(-\log\left(\text{c\_probs}\right)\cdot D_{\text{r}}\right), \tag{17}$$

$$\text{V\_Loss} = \text{mean}(\text{mse}(V_{\text{s}}, T)). \tag{18}$$

Here, mean$(\cdot)$ refers to computing the arithmetic mean of squared errors across all samples, and mse$(\cdot)$ (short for mean squared error) denotes calculating the squared difference between the predicted and true target values for each sample, followed by mean reduction to obtain the overall error.

Operator_Loss and Chip_Loss are losses of the actor network, representing the policy gradient losses of the operator and the chiplet, respectively. V_Loss is the loss of the critic network. o_probs denotes the output probabilities of the operator; c_probs denotes the output probabilities of the chiplet. D_{r} represents the reward per iteration and V_{s} represents the state value function.

Finally, the features extracted from the GCN are integrated through a connection layer. This layer combines outputs from different inputs— operation information, chip vector, and average vector—into a unified feature representation for subsequent neural network layers. This integrated approach enables efficient task scheduling by leveraging the strengths of GCN and neural networks within a reinforcement learning framework.

The advantage function $\boldsymbol{\delta}_t$ at model update $t+1$ is given by

$$\boldsymbol{\delta}_t = \boldsymbol{R}_t\gamma\boldsymbol{v}\left(S_{t+1}^{\text{o+c}},\omega_t\right) - \boldsymbol{v}\left(S_t^{\text{o+c}},\omega_t\right), \tag{19}$$

where $\boldsymbol{R}_t$ is the immediate reward vector collected at t, $\boldsymbol{v}\left(S_{t+1}^{\text{o+c}},\omega_t\right)$ is the value of the next state collected at t, and $\boldsymbol{v}\left(S_t^{\text{o+c}},\omega_t\right)$ is the value of the current state collected at t. In addition, ω_t, θ_t, and ϕ_t correspond to the value network, operator network, and chip network at time t, respectively. γ is a discount factor. The value network parameter ω_{t+1} is updated according to

$$\omega_{t+1} = \omega_t - \alpha_\omega\cdot\text{mean}\left(\boldsymbol{\delta}_t\nabla_\omega\boldsymbol{v}\left(S_t^{\text{o+c}},\omega_t\right)\right). \tag{20}$$

The operator network parameter θ_{t+1} is updated with

$$\theta_{t+1} = \theta_t - \alpha_\theta^{\text{o}}\cdot\ \text{mean}\ \left(\boldsymbol{\delta}_t\nabla_\theta\log\boldsymbol{\pi}\left(A_t^{\text{o}}\mid S_t^{\text{o}},\theta_t\right)\right). \tag{21}$$

Additionally, the chip network parameter ϕ_{t+1} is updated with

$$\phi_{t+1} = \phi_t - \alpha_\phi^{\text{c}}\cdot\text{mean}\left(\boldsymbol{\delta}_t\nabla_\phi\log\boldsymbol{\pi}\left(A_t^{\text{c}}\mid S_t^{\text{c}},\phi_t\right)\right). \tag{22}$$

α_ω, α_θ^{o}, and α_ϕ^{c} are the discount factors. The action probabilities $\boldsymbol{\pi}\left(A_t^{\text{o}}\mid S_t^{\text{o}},\theta_t\right)$ and $\boldsymbol{\pi}\left(A_t^{\text{c}}\mid S_t^{\text{c}},\phi_t\right)$ are defined as the predicted action probability under the condition of the given state vector and action vector.

The right side of Fig. 7 illustrates the policy network and value network. The policy network outputs action probability distributions, while the value network evaluates samples to update the model under a specific loss function, as shown in Eqs. (16)–(18). During each iteration, based on the current state, the operator neural network (actor_1) generates a probability distribution (prob_1) to select an action. Once an action is chosen, the core neural network (actor_2) produces another distribution (prob_2) for core selection, as shown in Eqs. (19)–(22).

Here, we give an example to illustrate the operator selection detail, in Fig. 8. From step 1 to step 5, the process of selecting an operator choice tree is as follows: in step 1, operator v_1 has a probability set p_1, p_2, p_3 and a reward set r_1, r_2, r_3 with corresponding candidate operator nodes v_2, v_3, and v_4. In step 2, when operator v_2 is chosen in the next

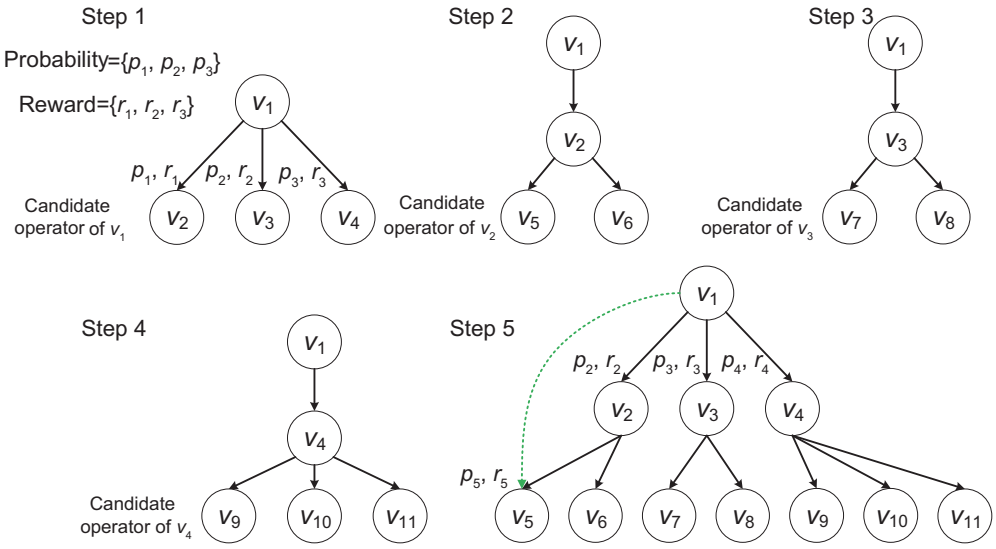

Fig. 8 An example of the operator search tree with probability and reward assignments

step, v_2 reveals its candidate operator nodes v_5 and v_6, and at the same time, a set of probabilities and a set of rewards are obtained. In the same way, in step 3, operator v_3 shows its candidate operators v_7 and v_8. Step 4 presents the candidate operators v_9, v_{10}, and v_{11} for operator v_4, similarly. Finally, in step 5, starting from operator v_1, different paths are chosen based on probabilities and rewards to determine the operator to be executed. For example, from v_1 to v_5, the path reward can be calculated as $R_{1-5} = r_2 + \gamma r_5$.

In the process of optimizing chiplet selection and layout, we fully consider factors such as performance, area, and power consumption. We can customize the weights of these factors based on the specific application's needs to create an overall optimization goal, as expressed in Eq. (23). For instance, when selecting chiplets, we consult a chip library containing key parameters like size, power consumption, storage, and latency. During layout and scheduling, these factors are integrated into the iterative optimization process based on the reinforcement learning model.

$$\text{Total} = \alpha_{\text{t}} \times \text{Throughput} + \beta_{\text{l}} \times \text{Latency} + \gamma_{\text{p}} \times \text{Power}, \quad (23)$$

where α_{t}, β_{l}, and γ_{p} denote the weighting parameters for throughput, latency, and power, respectively. These parameters are tunable based on specific application requirements (e.g., prioritizing high throughput).

5 Application examples and experiment

5.1 Experimental setup

The WSC architecture-automated design framework is implemented in Python and integrates the PyTorch-based RL module. The framework consists of an analysis input module, a layout module, an operator clustering module, and a reinforcement learning-driven iterative module. The chip operator library provides support for CPU, digital signal processing (DSP), field programmable gate array (FPGA), SRAM, and DRAM, whose physical parameters and capacity constraints are strictly defined following inequalities (3)–(5). During training iterations, heterogeneous task streams (bandwidth: 1–64 Gb/s, covering PCIe/UCIe protocols) are dynamically generated, with core physical information summarized in Table 3.

Table 3 Chip library examples

Type	Size (mm^2)	Power (mW)	Data	Latency (ms)
DSP	9.8×10.7	5	14M	10
CPU	24.5×20.1	30	64M	10
DDR	8.75×8.75		1000M	
Switch	1.0×1.0	0.8		200

It should be noted that existing WSC design solutions—such as Tesla's DOJO—achieve extreme TB/PB-level bandwidth through domain-specific customization. Such designs deeply bind specific chip architectures, protocols, software stacks, and task types, sacrificing flexibility and scalability despite meeting the needs of exclusive scenarios. In contrast, the 64 GB/s bandwidth constraint in our experiments only reflects the current limitations of general-purpose commercial chiplets, not the technical upper bound of the SDSoW framework. By avoiding reliance on domain-specific chiplet customization or exclusive interconnection protocols, SDSoW natively supports both general-purpose commercial and dedicated high-bandwidth chiplets; this design enables flexible multi-scenario adaptation and rapid verification, which aligns with the heterogeneous task settings of our experiments.

5.2 Task description

MD5 is a widely used password hash function. MD5 accepts a 64-byte multiple of the integer as input (with the possibility of a smaller integer), and the output is a 16-byte hash value. This is employed to ensure that the information

transmission is complete and consistent. MD5 can thus be regarded as an operator, the graph of which is shown in Fig. 9a. The external input, which is the key to be cracked, is represented by 0xAECF, while the plaintext spaces are represented by 0x00000000–0xFFFFFFFF; the relevant task data are shown in Table 4.

In the domain of signal processing tasks, taking the radar signal processing flow as an example, as shown in Fig. 9b, the amount of data in a single task is $1 \times 715 \times 93 \times 16$ bytes, and there are 20 groups of data that can be executed in parallel. After executing the 20 groups of data, they are merged into a 20×715×93-dimensional matrix to input the STAP task, and finally, CFAR target detection is performed. The relevant task data are shown in Table 4.

The comparison scheme incorporates both the architecture (Li FP et al., 2022) of fixed DDR external memory and the mesh interconnection architecture without external memory. In the verification stage, the closed-loop feedback mechanism is used to transmit the timing violation and thermal distribution results back to the layout network (as shown in Fig. 5) to facilitate the parameter iteration.

5.3 Layout comparison

5.3.1 MD5 procedure

In the MD5 task, the layout with external memory can place four components, each equipped with four processing elements, within the effective area of an eight-inch wafer (as shown in Fig. 10a). In contrast, the optimized architecture, which eliminates DDR at the layout stage, can place nine components, each with nine processing elements, on the same wafer size (see Fig. 10b). This significantly improves the component density and processing capacity within the same footprint.

5.3.2 Signal processing procedure

Under the signal processing task, each processing element in the layout with external memory must be equipped with DDR, as each processing element must perform the entire task and handle significant data interaction. In such a layout, 18 components, each with four execution units, can be placed within the effective area of an eight-inch wafer (as shown in Fig. 10c). Fig. 10d shows the optimized layout results achieved by our proposed WSC framework. By removing DDR from the layout stage, the new architecture can accommodate 30 components, each consisting of 4 processing elements, within the same wafer area. This optimization significantly enhances the component density without compromising functionality.

5.4 Latency and throughput comparison

5.4.1 MD5 procedure

The performance of the architecture under consideration is compared to that of a layout with external memory in terms of throughput and latency. The wafer layout results are shown in Fig. 10. The optimized architecture significantly reduces the area of each core group, which allows a larger number of core groups to be placed on a wafer, thus reducing computational latency. From a throughput perspective, the optimized

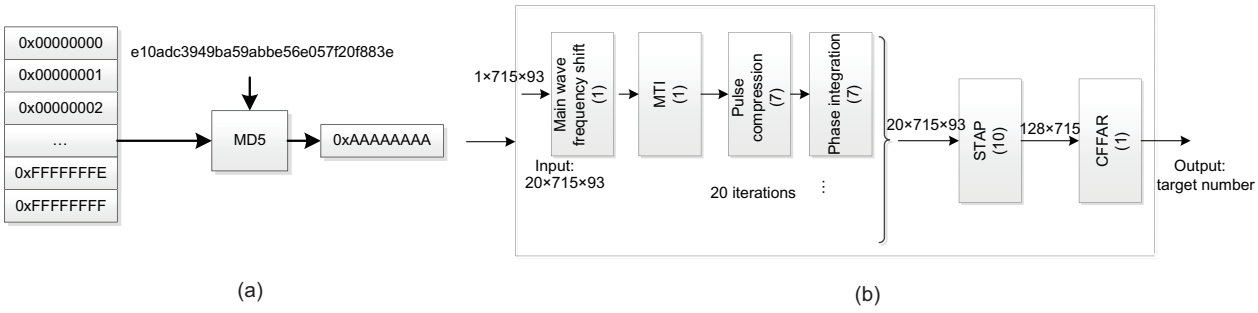

Fig. 9 An example of a decryption and encryption task (a) and a signal processing task (b)

Table 4 Resource and computational requirements of the task

Task	Procedure	Processor	Memory	Communication	Computation
MD5	MD5	CPU		2000	8000
		DSP		32 000	12 000
		CGRA		128 000	2000
Signal processing	Reading data	DSP	30M	0.5M × 20	100
	Main clutter frequency shift		30M	0.5M × 20	100
	MTI		30M	0.5M × 20	100
	Pulse compression		30M	0.5M × 20	100
	Pulse integration		45M	0.5 × 20 × 1.4	700
	STAP		45M	0.5 × 20 × 1.4	1000
	CFAR		45M	0.5 × 20 × 1.4	100

layout employs an unbound storage structure that allows more resources to be allocated, significantly increasing processing speed. In the layout with external memory, where each DSP performs the entire task individually, the throughput of the system is calculated as Eqs. (12) and (13). The computational latency and the system throughput are shown in Fig. 11a and Fig. 11c, respectively. The experimental results show that the throughput is improved by a factor of 22 compared to the layout with external memory.

5.4.2 Signal processing procedure

As shown in Figs. 11b and 11d, increasing bandwidth reduces latency in both architectures, but the improvement saturates beyond 4 GB/s for the external-memory (traditional) design. The proposed buffer-less architecture achieves an 11× throughput gain by exploiting data-level parallelism and eliminating off-chip DRAM. However, when the bandwidth falls below 8 GB/s, the traditional layout temporarily outperforms the proposed architecture, as its task-level parallelism achieves higher resource utilization under severe bandwidth constraints. This crossover highlights the critical need for hardware–software co-design.

As illustrated in Fig. 12, a signal processing task is assigned to a component including four processors: A, B, C, and D. Each element conducts five rounds of single-channel signal processing, with final results aggregated at processor D. In the timeline, gray areas indicate inter-core communication

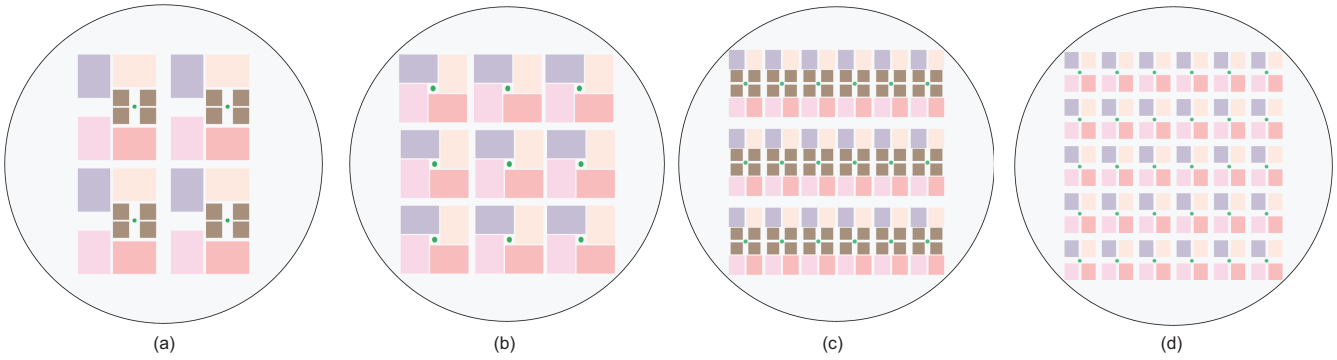

Fig. 10 Comparison of wafer layout for the MD5 task (a and b) and the signal processing task (c and d): (a) attached storage (MD5); (b) optimized layout (MD5); (c) attached storage (signal processing); (d) optimized layout (signal processing)

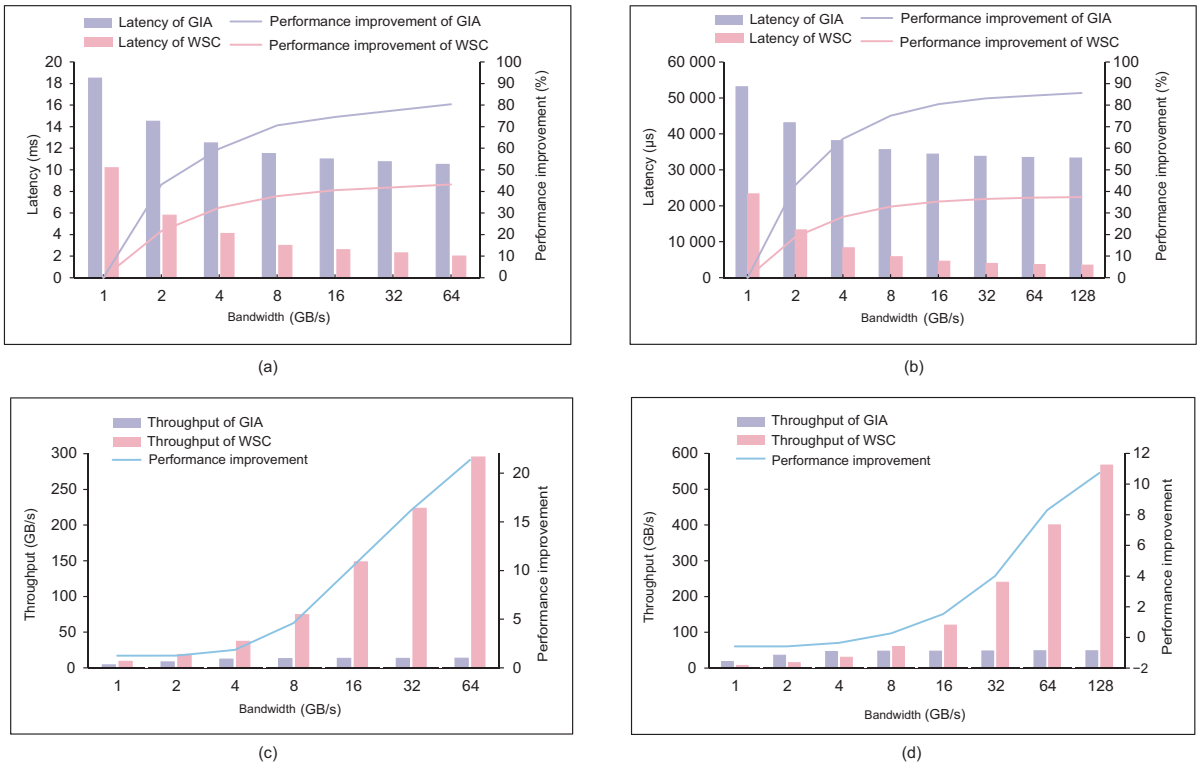

Fig. 11 Latency and throughput comparison of different design methods across MD5 procedure and signal processing procedure: (a) latency (MD5 procedure); (b) latency (signal processing procedure); (c) throughput (MD5 procedure); (d) throughput (signal processing procedure)

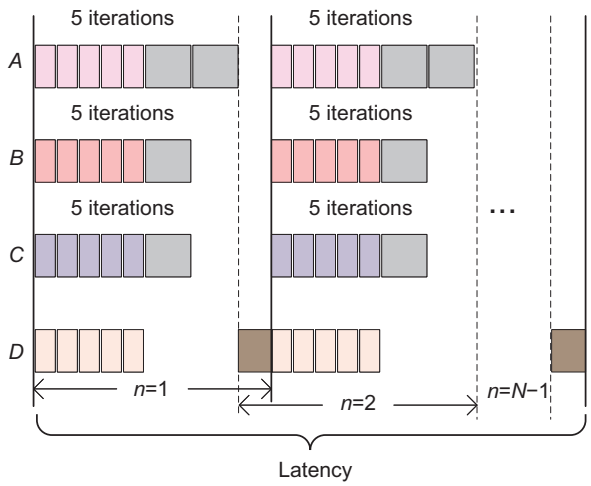

Fig. 12 Pipelined execution of the signal processing procedure

latency, and colored segments show operation execution on processors. The horizontal axis measures latency, showing the task pipeline's temporal progression. Each processor operates in iterative cycles with a set number of processing cycles per iteration. Dotted lines mark pipeline stages, showing data flowing from one processor to the next. This visualization offers insights into the signal processing task pipeline's parallelism and latency. It clarifies how each processor performs its operations in a pipelined way, and results are aggregated at D, while clearly showing inter-core communication latency.

5.5 Analysis of different optimization methods

We conducted experiments to validate the superiority of our RL-based method. Detailed experimental settings are as follows: the simulated annealing (SA) algorithm and enumeration (Enum) method were selected as baselines. MD5 was adopted as the test task, with constraints derived from an eight-inch wafer, and the number of tasks was set to 10, 15, 20, 25, 30, and 50. Performance was evaluated from two key perspectives: the throughput of scheduling schemes generated within a specified time and the search time under predefined constraints.

As shown in Fig. 13a, the experimental results demonstrate that our proposed method reduces the search time by an average factor of 12 compared with Enum and by 44% compared with SA. Fig. 13b further validates its superiority in throughput under time-constrained scheduling: compared with Enum and SA, the proposed method achieves notable throughput improvements, delivering a consistent performance across multi-task scenarios. These results confirm significant performance gains in both search efficiency and scheduling throughput.

6 Conclusions

The development of WSC design holds significant promise for the future. Our proposed hardware–software co-design methodology offers a comprehensive solution to address the complexities and high-performance requirements in modern

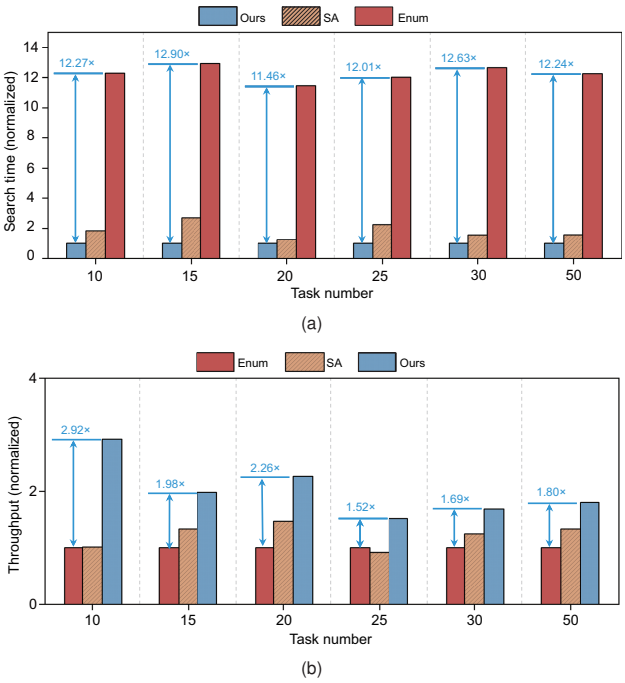

Fig. 13 Overall performance comparison of different optimization methods in multi-task scenarios: (a) search time comparison results in multi-task scenarios; (b) throughput comparison results in multi-task scenarios

microelectronics. We have systematically explored the WSC development process, focusing on optimizing the architecture space for hardware–software co-design. Despite challenges such as limited tools and core libraries in WSC design, we have made great progress by building a comprehensive core library and refining the design flow, which has greatly boosted the design efficiency.

Moving forward, we plan to expand our exploration into diverse application fields, such as large-scale AI training, autonomous driving, and scientific computing to verify the framework's adaptability. We will conduct in-depth evaluations of multi-task execution on a single fixed layout, validating its cross-task generality to address layout rationality and cost concerns. Additionally, we will integrate thermal and power management into optimization goals, supplement system-level power analysis, and perform quantitative evaluation of yield and stress factors for more reliable designs. Furthermore, we will enhance support for heterogeneous components and finalize the open-source toolchain with detailed tutorials to ensure reproducibility, boosting the framework's flexibility and practicality.

Acknowledgments

This work was supported by the National Key R&D Program of China (No. 2022YFB4401400) and the Songshan Laboratory (No. 221100211100).

Author contributions

Wenbo ZHANG and Bo DING conducted the experiments, analyzed and processed the data, and drafted the paper. Shuai WEI conceived and designed the study. Hong YU developed the software and visualization tools and validated the results. Qinrang LIU and Ke SONG provided the project data, supervised the project, secured

funding, and offered administrative support. Wei GUO critically reviewed the paper and provided the overall project guidance. Bo MEI and Rui ZHENG revised the paper, provided the technical guidance, and edited the final version.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declaration on the use of generative AI tools

During the preparation of this work, the authors used ChatGPT to improve language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

Achiam J, Adler S, Agarwal S, et al., 2023. GPT-4 Technical Report. https://api.semanticscholar.org/CorpusID:257532815 [Accessed on Dec. 1, 2025].

Ahmad M, DeLaCruz J, Ramamurthy A, 2022. Heterogeneous integration of chiplets: cost and yield tradeoff analysis. Proc 23rd Int Conf on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems, p.1-9. https://doi.org/10.1109/EuroSimE54907.2022.9758914

Ali H, Tariq UU, Hardy J, et al., 2021. A survey on system-level energy optimisation for MPSoCs in IoT and consumer electronics. *Comput Sci Rev*, 41:100416. https://doi.org/10.1016/j.cosrev.2021.100416

Baktash JA, Dawodi M, 2023. GPT-4: a review on advancements and opportunities in natural language processing. *J Elect Electron Eng*, 2(4):548-549. https://doi.org/10.33140/jeee.02.04.19

Binkert N, Beckmann B, Black G, et al., 2011. The gem5 simulator. *ACM SIGARCH Comput Archit News*, 39(2):1-7. https://doi.org/10.1145/2024716.2024718

Bohr M, 2009. The new era of scaling in an SoC world. Proc IEEE Int Solid-State Circuits Conf—Digest of Technical Papers, p.23-28. https://doi.org/10.1109/ISSCC.2009.4977293

Brown TB, Mann B, Ryder N, et al., 2020. Language models are few-shot learners. Proc 34th Int Conf on Neural Information Processing Systems, Article 159.

Burns JA, Aull BF, Chen CK, et al., 2006. A wafer-scale 3-D circuit integration technology. *IEEE Trans Electron Dev*, 53(10):2507-2516. https://doi.org/10.1109/TED.2006.882043

Chakaravarthy RV, Kwon H, Jiang H, 2021. Vision control unit in fully self-driving vehicles using Xilinx MPSoC and opensource stack. Proc 26th Asia and South Pacific Design Automation Conf, p.311-317. https://doi.org/10.1145/3394885.3431616

Chen SX, Li SY, Zhuang Z, et al., 2024. Floorplet: performance-aware floorplan framework for chiplet integration. *IEEE Trans Comput-Aid Des Integr Circ Syst*, 43(6):1638-1649. https://doi.org/10.1109/TCAD.2023.3347302

Chen YW, Wang RH, Cheng YH, et al., 2024. SUN: dynamic hybrid-precision SRAM-based CIM accelerator with high macro utilization using structured pruning mixed-precision networks. *IEEE Trans Comput-Aid Des Integr Circ Syst*, 43(7):2163-2176. https://doi.org/10.1109/TCAD.2024.3358583

Chowdhery A, Narang S, Devlin J, et al., 2023. PaLM: scaling language modeling with pathways. *J Mach Learn Res*, 24(1):240.

Deng CH, Li XY, Feng Z, et al., 2022. GARNet: reduced-rank topology learning for robust and scalable graph neural networks. https://doi.org/10.48550/arXiv.2201.12741

Feng YX, Ma KS, 2022. Chiplet actuary: a quantitative cost model and multi-chiplet architecture exploration. Proc 59th ACM/IEEE Design Automation Conf, p.121-126. https://doi.org/10.1145/3489517.3530428

Hammarlund P, Martinez AJ, Bajwa AA, et al., 2014. Haswell: the fourth-generation Intel Core Processor. *IEEE Micro*, 34(2):6-20. https://doi.org/10.1109/MM.2014.10

Han YH, Xu HB, Lu MX, et al., 2024. The big chip: challenge, model and architecture. *Fund Res*, 4(6):1431-1441. https://doi.org/10.1016/j.fmre.2023.10.020

Hu Y, Lin XH, Wang HZ, et al., 2024. Wafer-scale computing: advancements, challenges, and future perspectives. *IEEE Circ Syst Mag*, 24(1):52-81. https://doi.org/10.1109/MCAS.2024.3349669

IEEE, 2024. International Roadmap for Devices and Systems™. https://irds.ieee.org/images/files/pdf/2024/2024IRDS_MET.pdf [Accessed on Dec. 1, 2025].

Jung S, Lee H, Myung S, et al., 2022. A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature*, 601(7892):211-216. https://doi.org/10.1038/s41586-021-04196-6

Leon V, Minaidis P, Lentaris G, et al., 2023. Accelerating AI and computer vision for satellite pose estimation on the Intel Myriad X embedded SoC. *Microprocess Microsyst*, 103:104947. https://doi.org/10.1016/j.micpro.2023.104947

Leon V, Minaidis P, Soudris D, et al., 2024. MPAI: a co-processing architecture with MPSoC & AI accelerators for vision applications in space. Proc 31st IEEE Int Conf on Electronics, Circuits and Systems, p.1-2. https://doi.org/10.1109/ICECS61496.2024.10848988

Li FP, Wang Y, Cheng YQ, et al., 2022. GIA: a reusable general interposer architecture for agile chiplet integration. Proc IEEE/ACM Int Conf on Computer Aided Design, p.1-9. https://doi.org/10.1145/3508352.3549464

Li ZS, Liu LB, Deng YD, et al., 2017. Aggressive pipelining of irregular applications on reconfigurable hardware. Proc 44th Annual Int Symp on Computer Architecture, p.575-586. https://doi.org/10.1145/3079856.3080228

Loh GH, Xie Y, Black B, 2007. Processor design in 3D die-stacking technologies. *IEEE Micro*, 27(3):31-48. https://doi.org/10.1109/MM.2007.59

Markidis S, Der Chien SW, Laure E, et al., 2018. NVIDIA tensor core programmability, performance & precision. Proc IEEE Int Parallel and Distributed Processing Symp Workshops, p.522-531. https://doi.org/10.1109/IPDPSW.2018.00091

Pal S, Petrisko D, Tomei M, et al., 2019. Architecting waferscale processors—a GPU case study. Proc IEEE Int Symp on High Performance Computer Architecture, p.250-263. https://doi.org/10.1109/HPCA.2019.00042

Pal S, Liu JY, Alam I, et al., 2021. Designing a 2048-chiplet, 14336-core waferscale processor. Proc 58th ACM/IEEE Design Automation Conf, p.1183-1188. https://doi.org/10.1109/DAC18074.2021.9586194

Panousopoulos V, Papaloukas E, Leon V, et al., 2024. HW/SW co-design on embedded SoC FPGA for star tracking optimization in space applications. *J Real-Time Image Proc*, 21(1):16. https://doi.org/10.1007/s11554-023-01391-8

Patel D, Wong G, 2023. GPT-4 architecture, infrastructure, training dataset, costs, vision, MoE. Proc Demystifying GPT-4: the Engineering Tradeoffs that Led OpenAI to Their Architecture, p.1-17.

Raffel C, Shazeer N, Roberts A, et al., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J Mach Learn Res*, 21(140):1-67.

Shao YS, Clemons J, Venkatesan R, et al., 2019. Simba: scaling deep-learning inference with multi-chip-module-based architecture. Proc 52nd Annual IEEE/ACM Int Symp on Microarchitecture, p.14-27. https://doi.org/10.1145/3352460.3358302

Talpes E, Williams D, Sarma DD, 2022. DOJO: the microarchitecture of Tesla's exa-scale computer. Proc IEEE Hot Chips 34 Symp, p.1-28. https://doi.org/10.1109/HCS55958.2022.9895534

Tang XP, Tian RQ, Wong DF, 2001. Fast evaluation of sequence pair in block placement by longest common subsequence computation. *IEEE Trans Comput-Aid Des Integr Circ Syst*, 20(12):1406-1413. https://doi.org/10.1109/43.969434

Tatar G, Bayar S, Çiçek İ, 2024. Real-time multi-learning deep neural network on an MPSoC-FPGA for intelligent vehicles: harnessing hardware acceleration with pipeline. *IEEE Trans Intell Veh*, 9(6):5021-5032. https://doi.org/10.1109/TIV.2024.3398215

Touvron H, Martin L, Stone K, et al., 2023. Llama 2: open foundation and fine-tuned chat models. https://doi.org/10.48550/arXiv.2307.09288

Turner WJ, Poulton JW, Wilson JM, et al., 2018. Ground-referenced signaling for intra-chip and short-reach chip-to-chip interconnects. Proc IEEE Custom Integrated Circuits Conf, p.1-8. https://doi.org/10.1109/CICC.2018.8357077

Venkatesan R, Shao YS, Wang MR, et al., 2019. MAGNet: a modular accelerator generator for neural networks. Proc IEEE/ACM Int Conf on Computer-Aided Design, p.1-8. https://doi.org/10.1109/ICCAD45719.2019.8942127

Weng J, Liu SH, Dadu V, et al., 2020. DSAGen: synthesizing programmable spatial accelerators. Proc 47th Annual Int Symp on Computer Architecture, p.268-281. https://doi.org/10.1109/ISCA45697.2020.00032

Wu JX, Liu QR, Shen JL, et al., 2024. From SoC to SDSoW: a new paradigm for microelectronics development. *Sci Sin Inform*, 54:1350-1368. https://doi.org/10.1360/SSI-2023-0219

Xu QZ, Wang CH, Li ZQ, et al., 2025. A wafer-scale heterogeneous integration thermal simulator. *Appl Therm Eng*, 264:125459. https://doi.org/10.1016/j.applthermaleng.2024.125459

Yenduri G, Ramalingam M, Selvi GC, et al., 2024. GPT (generative pre-trained transformer)—a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access*, 12:54608-54649. https://doi.org/10.1109/ACCESS.2024.3389497

Zhang JM, Wang XY, Ye YY, et al., 2024. M2M: a fine-grained mapping framework to accelerate multiple DNNs on a multi-chiplet architecture. *IEEE Trans VLSI Syst*, 32(10):1864-1877. https://doi.org/10.1109/TVLSI.2024.3438549

Zhang SS, Roller S, Goyal N, et al., 2022. OPT: open pre-trained transformer language models. https://doi.org/10.48550/arXiv.2205.01068

Zhu JC, Xue CH, Chen YQ, et al., 2025. Theseus: exploring efficient wafer-scale chip design for large language models. *IEEE Trans Comput-Aid Des Integr Circ Syst*, 44(12):4793-4806. https://doi.org/10.1109/TCAD.2025.3566297

Zhuang Z, Yu B, Chao KY, et al., 2022. Multi-package co-design for chiplet integration. Proc 41st IEEE/ACM Int Conf on Computer-Aided Design, Article 4. https://doi.org/10.1145/3508352.3549404

Zou DX, Wang GG, Pan G, et al., 2016. A modified simulated annealing algorithm and an excessive area model for floorplanning using fixed-outline constraints. *Front Inform Technol Electron Eng*, 17(11):1228-1244. https://doi.org/10.1631/FITEE.1500386