HEX-SIM: Evaluating Multi-modal Large Language Models on Multi-chiplet NPUs

Xinquan Lin
School of Physics and Information Engineering
Fuzhou University
Fuzhou, China
1874034619@qq.com

Haobo Xu
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
xuhaobo@ict.ac.cn

Yinhe Han
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
yinhes@ict.ac.cn

Yiming Gan*
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
ganyiming@ict.ac.cn

***Abstract*—Deep learning models are scaling in both parameters and modalities. Multi-modal large language models are increasingly used in robotic applications, driving the need for large-scale deep learning accelerators. Multi-chiplet heterogeneous neural network accelerators are an effective solution for today's multi-modal large language models. Different types of chiplets provide diverse functionalities, enabling large data storage, high on-chip bandwidth, and significant computing capability. While single-core or multi-core NPU accelerators can be validated through simulation, there is still a lack of software-level cycle-accurate simulators for multi-chiplet NPUs.**

In this work, we propose HEX-SIM, a configurable multi-chiplet deep learning accelerator simulator. HEX-SIM offers designers various macro architectures and system parameters to better evaluate accelerator designs. We conduct extensive simulation experiments using HEX-SIM, demonstrating the effects of parallelism, bandwidth, buffer size, and the number of computing engines on inference latency. These insights can significantly aid users in optimizing their designs. Our project code is open-sourced and available at https://github.com/jimrelief/HEX-SIM.

I. INTRODUCTION

Deep learning algorithms, as well as deep learning accelerators, are widely used in robotic applications [1–3]. Accurate simulation of deep learning accelerators's inference performance before deployment in real robots is crucial, as it helps estimate the performance and energy consumption of robots [4–6], which in turn determines the required computation resources. Researchers have also explored the development of deep learning accelerator simulators.

However, traditional deep learning accelerator simulators are no longer sufficient for simulating the inference procedures of current algorithms used in robotic applications due to two fundamental reasons. **Firstly, neural network models are progressing towards enhanced multi-modal capabilities and larger parameter size.** The models have evolved from single-modality, single-functionality networks, such as keyword spotting networks [7–9], natural language understanding [10–12], and object detection and classification [13–15], to multi-modality, end-to-end large language models like robotic transformer (RT) [16] and PaLM-E [17]. These networks are much larger and incorporate multiple types of layers. For example, PaLM-E includes both convolutional layers and transformers within a single network and contains billions of parameters. The existing simulators are inadequate in simulating multi-modal models and large-scale models effectively. **Secondly, as the scale and modalities of models increase, the scale and diversity of hardware are also expanding.** As the model size and modality increase, the hardware also scales. Deep learning accelerators have evolved from single-core [18–20] to multi-core [21–23] and further to multi-chiplet architectures [24–26]. The emerging chiplet-based DNN accelerator involves integrating multiple multi-core chiplets into a high-throughput, large chip using advanced packaging techniques. This introduces a distinct memory hierarchy and interconnect topology.

Despite extensive studies on multi-core and multi-chiplet Neural Processing Units (NPUs) [27], there is no existing complete simulator for multi-chiplet NPUs that can enable full-system simulation of the NPU, High Bandwidth Memory (HBM), and CPU. Evaluating multi-modality large language models on a multi-chiplet NPU faces two fundamental bottlenecks. Firstly, the complex memory hierarchy and data loading pose new challenges. Unlike SCALE-SIM [28] with a single on-chip buffer, a multi-chiplet NPU stores data across chiplets, multi-core NPUs, and individual cores, with multiple levels of on-chip buffers and off-chip DRAM. Secondly, large models exceed the memory capacity of a single NPU, and different scheduling and partitioning methods result in varied performance.

In this work, we design a cycle-accurate simulator for a multi-chiplet NPU, aiming to analyze and evaluate key parameters that impact resource allocation, performance, and

* indicates the corresponding author of the paper.

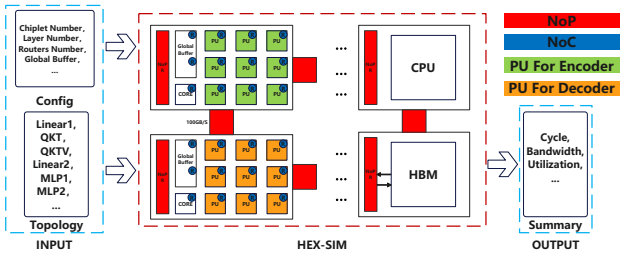


Fig. 1: Schematic depicting HEX-SIM

energy consumption. HEX-SIM establishes a basis architecture for a multi-chiplet integrated NPU and exposes various architectural details, including the number of chiplets, PUs, systolic array size, local buffer size, and system integration parameters like memory bandwidth at each level. Furthermore, HEX-SIM supports a wide range of workloads, especially multi-modality large language models used in robotic applications like robotic transformer and roboflamingo [29]. These models typically encode sensor inputs via convolutional neural networks and decode them into robot control instructions through transformer-based decoders. Even the smallest models have billions of parameters. In addition, HEX-SIM also supports other types of models, apart from the multimodal large language models used for robotic applications. Given the hardware parameters configuration file and workload configuration file, HEX-SIM reports the latency, utilization, SRAM access, and different levels of DRAM access. Fig. 1 shows the workflow of HEX-SIM.

Using HEX-SIM, we conduct a comprehensive analysis of the multi-chiplet NPU design space, uncovering key insights for hardware and software designers. These insights fall into three categories. First, we explore different model parallelism mechanisms, crucial as many large language models exceed single chiplet capacities. Second, we delve into hardware resource allocation for hybrid multi-chiplet NPUs running multi-modality large language models, highlighting performance variations with resource distribution changes. Lastly, we investigate memory size and bandwidth requirements across the memory hierarchy. The insights we propose using HEX-SIM are useful when designing a multi-chiplet NPU.

In summary, this paper makes the following contributions:

- We provide an open-source, cycle-accurate simulator for multi-chiplets NPUs.
- With the simulator, we evaluate multiple state-of-the-art multi-modality large language models used in the robotic domain and with different parallelism execution patterns.
- We provide insights on different trade-offs to help chip designers better allocate hardware resources.

The rest of the paper is organized as follows. We introduce the background and motivation of this paper in Sec. II. Sec. III describes detailed simulator design. Sec. IV evaluates multiple multi-modality large language models and uses HEX-SIM to highlight hardware design insights. Sec. V put this work in the context of related works and Sec. VII concludes the paper.

II. BACKGROUND

This paper is motivated by two trends in the domain of deep learning accelerators. Firstly, the integration of multiple chiplets has emerged as a promising approach for designing large-scale deep learning acceleration systems (Sec. II-A). Secondly, the accurate simulation of deep learning accelerator performance is crucial during the chip design process (Sec. II-B).

A. Large Scale Deep Learning Accelerators

Deep learning accelerators are growing larger as model sizes increase. There are two approaches to building large-scale neural network accelerators. The first approach is wafer-scale integration (WSI), which entails constructing a massive integrated deep-learning accelerator using an entire silicon wafer. For example, chip designers like Cerebras have developed a wafer-scale chip specifically designed to accelerate deep learning applications. This chip is $46225mm^2$ in size and boasts 900,000 cores [30]. However, this design approach comes with challenges related to chip yield and chip cost. The second approach is package-level multi-chiplet integration, which is a more cost-effective and widely adopted method. Each chiplet serves as a small building block designed and fabricated individually. These chiplets are then connected through a silicon interposer or an organic substrate, with efficient intra-package connection circuits also utilized [31].

The multi-chiplet NPU offers several benefits. Firstly, design and fabrication costs are significantly reduced due to the smaller size of chiplets compared to a large monolithic chip. Secondly, scalability is improved as different types of chiplets can be integrated into a single system. Multi-chiplet designs have already seen widespread use in CPUs [32] and GPUs [33]. Additionally, multi-chiplet NPUs have been a recent area of study [34].

However, due to the highly modular design approach, the chiplet-based accelerator introduces a vast design space that encompasses both architecture and workload mapping considerations. From the architecture perspective, the constraints imposed by chiplet sizes and the use of silicon interposers in 2.5D-based accelerators create a trade-off between resource allocations for chiplets and the package. Additionally, the choice of interconnection designs can significantly impact efficiency discrepancies. Designers are faced with higher demands when it comes to exploring architecture options.

B. Deep Learning Acceleration Simulation

Using a system architectural simulator for architecture simulation is considered an effective means for conducting architecture design. Compared to hardware-level simulation, software-level cycle-accurate architectural simulators can provide faster feedback to designers, enabling quicker iterations [35–37].

Researchers have studied software-level cycle-accurate architectural simulators for both single-core and multi-core NPUs, yielding valuable design insights. For instance, results from the single-core simulator SCALE-SIM [28] indicate that no single data flow consistently outperforms others, with the

choice of data flow being tied to the size of the systolic array. mNPUsim [38] suggests that, given the same chip area, a multi-core NPU should incorporate more small-size systolic arrays rather than fewer large-size ones.

Building on the insights gained from previous examples, we introduce HEX-SIM, a cycle-accurate simulator for a multi-chiplet NPU. Our simulator considers diverse types of chiplets, non-uniform memory bandwidth, and varying model-to-chip mappings. Through this work, we aim to uncover valuable insights for designing multi-chiplet NPUs.

III. HEX-SIM DESIGN

In this section, we present our simulator design. We start by describing the architecture of the multi-chiplet NPU system (Sec. III-A). Next, we explain the parallelization of a multi-modal large language model on a multi-chiplet NPU and discuss how to calculate inference latency (Sec. III-B). Finally, we delve into the details of our simulator design (Sec. III-C).

A. Architecture of Multi-Chiplet NPU System

We illustrate the architecture of a basic multi-chiplet NPU in Fig. 2 [24]. This architecture comprises three layers: the chip layer, the chiplet layer, and the processing unit (PU) layer. When running a multi-modality large language model on a multi-chiplet NPU system, it requires at least four types of chiplets: CPU, High Bandwidth Memory (HBM), chiplets for vision encoding, and chiplets for decoding. The CPU dispatches instructions for inference globally, while HBM stores weights and inputs for the network. Given that most large language models in the robotic domain involve multiple modalities, we include two types of chiplets to meet these requirements. Each chiplet is interconnected through a network on package (NoP), with the maximum bandwidth determined by detailed interconnection technologies. In HEX-SIM, the number of CPUs, the number of other chiplets, and the communication bandwidth are configurable according to specific requirements.

Each chiplet designed for vision encoding and decoding is equipped with a local CPU, a global buffer (GB), and several processing units (PUs). These components are interconnected through a MESH topology network on chip (NoC). The maximum bandwidth of the NoC is determined by the design and is configurable. Additionally, the number and size of the PUs and the size of the GB are also configurable.

Each processing unit (PU) is a classic systolic array-based deep learning accelerator, with on-chip SRAM divided into an input buffer, a weight buffer, and an output buffer. The size of the systolic array and on-chip buffer, as well as the dataflow, are configurable. For instance, in our evaluation, PUs for vision encoding are equipped with a 768KB local buffer (LB) and a 16×16 systolic array, while PUs employed for decoding utilize a 1536KB LB and a 32×32 systolic array.

B. Modeling Multi-Chiplet NPU System

HEX-SIM offers cycle-accurate inference simulation for multi-chiplet NPUs, allowing users to validate and optimize their designs in the pre-silicon stage. To achieve this, HEX-SIM employs two accurate modeling approaches. Firstly, it accurately models computation and data flow, including data communication within the same level (e.g., between chiplets) and across different levels (e.g., between HBM chiplets and PUs inside another chiplet). Secondly, HEX-SIM models different types of model parallelism, as this is a fundamental principle when executing large language models.

Modeling Memory Hierarchy, Data Flow and Computation. Latency, utilization, SRAM access, and various levels of DRAM access are crucial parameters that indicate the effectiveness of a chip design, significantly influencing inference performance. We have identified that simulating a multi-chiplet NPU design begins with understanding its memory hierarchy details. Specifically, we accurately quantify the memory size, bandwidth and the data requirements for each layer. By combining these key parameters with data flow information and computations on each systolic array, HEX-SIM calculates the actual performance of a chip.

The memory architecture of a multi-chiplet NPU is categorized into three levels. Within each chip, one or multiple HBM chiplets store weights and inputs. Below the HBM chiplets is a global buffer (GB) located inside each chiplet, and below the GB, there is a local buffer (LB) inside each PU. These three levels of memory are interconnected in a strict dependency: LBs only read or write data to GBs, GBs read from HBM chiplets and write to LBs. We illustrate this hierarchy in Fig. 3.

In HEX-SIM, both GB and LB are modeled in a double-buffer fashion to hide the data communication latency during computation latency. However, due to workload variations and bandwidth differences between NoP and NoC, systolic arrays often need to stall while waiting for data preparation.

For accurate performance simulation, we initially determine the parallelization, which will be discussed later. Once the model parallelization topology is established, HEX-SIM generates a data stream containing the sizes of input and weight matrices for operations at the PU layer. By considering the size of data loaded from HBM to GB and from GB to LB, along with the predetermined data flow of each PU (e.g., input stationary, weight stationary, output stationary), the computation cycle of each systolic array can be simulated. Following the computation, partial sums or final outputs are aggregated back to the HBM chiplet.

In addition to latency, HEX-SIM also reports the average bandwidth of each layer, indicating the potential for improvement in interconnection technology within each layer, as well as the utilization rate of the systolic array. Breakdowns of each layer's data can also be reported upon user request. This information provides valuable insights during the chip design process.

The design principles of HEX-SIM enable not only hardware reconfiguration and expansion but also provide generalizability. Currently, we support four types of chiplets, with two dedicated to vision encoding and language model decoding. With the trend of increasing modalities used in large language models, HEX-SIM can be easily integrated with more types

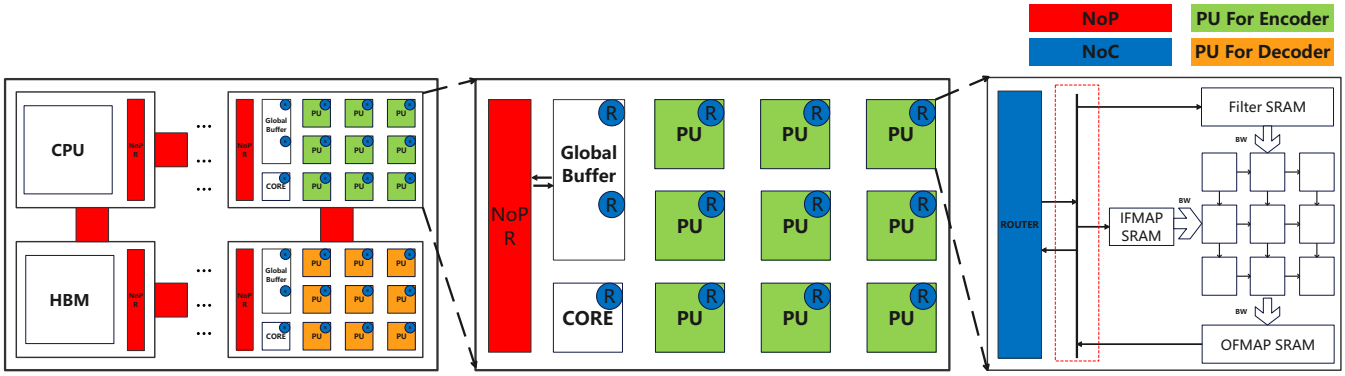


Fig. 2: Architecture of Multi-Chiplet NPU System. The chip consists of heterogeneous chiplets. Four chiplets are used in this work. Inside each chiplet, multiple PUs are connected through a NoC network. PU here means processing unit, each PU is a systolic array based deep learning accelerator. NoP means network on package, which is the connection between different chiplets, NoC means network on chip.

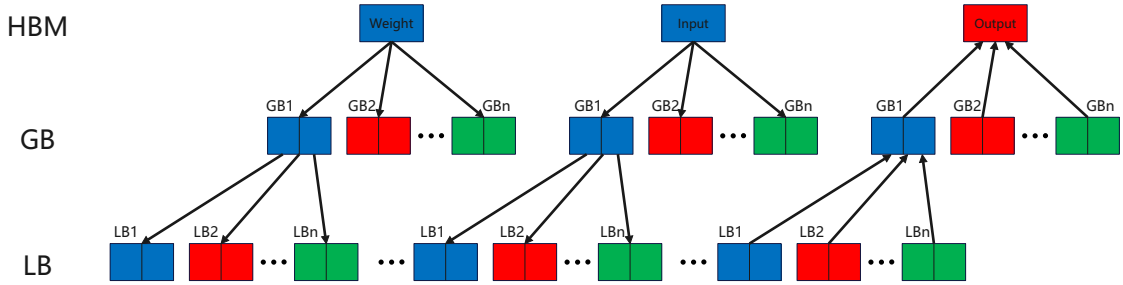


Fig. 3: Memory hierarchy modeling by HEX-SIM. HBM is used to store both weights, inputs and outputs. Each chiplet has its own global buffer (GB), each PU has its local buffer (LB).

of chiplets, such as chiplets for LiDAR processing and neural networks processing point cloud data.

Modeling Parallelism. Multi-chiplet NPUs offer abundant computational resources that enable users to employ various types of model-level parallelism [39], which is crucial for achieving real-time inference with large language models [40]. There are two main types of parallelism: layer-level parallelism, known as pipeline parallelism, and intra-layer parallelism, known as tensor parallelism [41, 42].

In HEX-SIM, we support both pipeline parallelism and two types of tensor parallelism: row parallelism and column parallelism. We explain how tensor parallelism operates in HEX-SIM, as it requires a customized data flow. The data flow for different types of tensor parallelism is illustrated in Fig. 4 and Fig. 5. In particular, the input matrix is denoted by X, the weight matrix by A, and the output matrix by Y.

In Fig. 4, we illustrate the workflow of row parallelism for large models in HEX-SIM. The input matrix is evenly split by column based on the number of chiplets corresponding to the type of the neural network, and similarly, the weight matrix is evenly split by row based on the number of chiplets. These segmented input matrices and weight matrices are transmitted to the chiplet layer through NoP. Within the chiplet layer, the input matrix and weight matrix are further divided

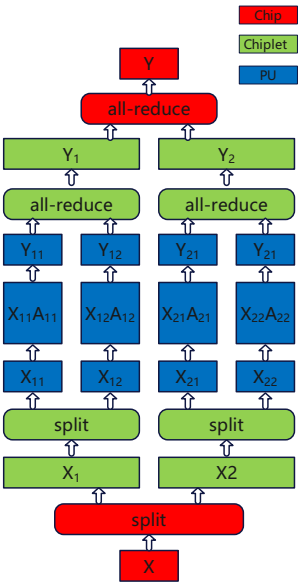


Fig. 4: Row Parallelism, where inputs and weights are distributed to different chiplets in a parallel way.

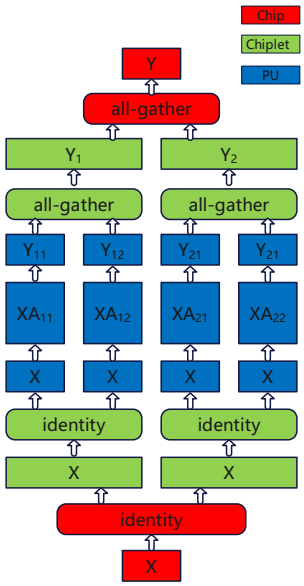


Fig. 5: Column Parallelism, where only weights are distributed to different chiplets in a parallel way.

uniformly based on the number of processing units (PUs) in the chiplet, and then transmitted to each PU through NoC. The input matrix and weight matrix in the PU layer undergo GEMM operations using a systolic array to produce the output matrix. The chiplet layer collects the output matrices from the PU layer through NoC for all-reduce operations. Finally, the output matrix is gathered from the chiplet layer through NoP in the chip layer, and an all-reduce operation is performed to obtain the final result of the GEMM calculation.

In Fig. 5, we depict the process of column parallelism in HEX-SIM. In the chip layer, the input matrix remains unsplit, while the weight matrix is evenly divided by columns based on the number of chiplets. These segmented weight matrices and the input matrix are then transmitted to the chiplet layer through NoP. Within the chiplet layer, the input matrix remains unchanged, while the weight matrix is segmented by column based on the number of processing units (PUs) in the chiplet. These segmented weight matrices are then transmitted to each PU through NoC. Similarly, in the PU layer, the output matrix is calculated using a systolic array. The chiplet layer collects the output matrix from the PU layer through NoC for all-gather operations. In the chip layer, the output matrix is gathered from the chiplet layer through NoP for all-gather operations to obtain the final GEMM calculation result.

The key distinction between column parallelism and row parallelism lies in the partitioning of the input matrix and weight matrix, as well as the aggregation of the output matrix. Row parallelism involves a finer granularity in matrix partitioning compared to column parallelism, resulting in lower computational requirements. However, row parallelism necessitates more complex all-reduce operations for the output matrix and demands a larger data output bandwidth during collection.

Overall, HEX-SIM supports two tensor parallel methods, which influence the GEMM computation of the systolic array in the final PU layer based on the selected parallel method. These different types of parallelism involve trade-offs in terms of inference latency, data movement, and even hardware design considerations, as we will further elaborate on in the evaluation.

C. Simulation Details

HEX-SIM takes both hardware configuration specifications and neural network topology as input for simulation. In this section, we discuss both inputs in detail.

1) Hardware Configuration: We provide an example of hardware configuration in Tbl. I. Since we are modeling a multi-chiplet NPU, users need to input parameters for each layer.

Chip Layer. At the chip layer, users need to specify three parameters: the number of chiplets used for vision encoding, the number of chiplets used for decoding, and the bandwidth of the connection between different chiplets.

Chiplet Layer. For each chiplet, we require information such as the type of chiplet, the number of router ports, the interconnection topology (e.g., mesh), the number of processing units (PUs), the bandwidth, and the size of the global buffer.

PU Layer. For each processing unit (PU), users need to provide the size of the systolic array, the size and partition of the on-chip buffer, the bandwidth, and the data flow. The type of PU should match the type of chiplet; in other words, if there are two types of chiplets in a chip, users should define PU information for both types.

Run Presets. In addition to the above information, users can also choose their preferred type of parallelism. Currently, we support both row and column parallelism, as well as pipeline parallelism.

2) Network Topology: In addition to hardware configuration, HEX-SIM also requires another configuration file describing the network topology. We provide an example of the network configuration in Tbl. II. The network is described in a layer-by-layer fashion, following a format that is similar to most DNN accelerator simulators. To ease the burden on users, we employ the same format for all layers, including convolutional layers and attention layers.

Meanwhile, HEX-SIM also supports mapping matrix operations from attention to the topology structure of convolution. For example, The GEMM operation of $m \times k$ and $k \times n$ can be mapped to a single channel $m \times k$ input matrix and n $1 \times k$

TABLE I: Hardware configuration, including modifiable specifications for chip layer, chiplet layer and PU layer and interconnections.

Package
Parallelism
Number of Encoder Chiplet
Number of Decoder Chiplet
Chiplet(Encoder)
Number of LAYER
Global Buffer Size(kB)
Routers per Global Buffer
Chiplet(Decoder)
Number of LAYER
Global Buffer Size(kB)
Routers per Global Buffer
PU(Encoder)
Array Size
IfmapSram(kB)
FilterSram(kB)
OfmapSram(kB)
Bandwidth
Dataflow
PU(Decoder)
Array Size
IfmapSram(kB)
FilterSram(kB)
OfmapSram(kB)
Bandwidth
Dataflow
Run Presets
Encoder Bandwidth
Decoder Bandwidth

TABLE II: Network topology, indicating the workloads that are running on HEX-SIM.

Parameter	Description
Layer Name	Defined tag
Input Height	Dimension of input matrix
Input Width	Dimension of input matrix
Filter height	Dimension of filter matrix
Filter Width	Dimension of filter matrix
Channels	Number of input channels
Num Filter	Number of filter
Strides	Strides in convolution

TABLE III: Validation, indicating the difference between the RTL computation cycle and HEX-SIM computation cycle.

Workload(matrix size)	RTL(cycle)	HEX-SIM (cycle)
10×10	72	72
20×20	82	82
30×30	92	92
40×40	204	204
50×50	224	224
60×60	244	244
70×70	396	396
80×80	426	426
90×90	456	456
100×100	648	648

convolution kernels when performing convolution with a stride of 1.

IV. EVALUATION

HEX-SIM opens up a huge design space for developing a multi-chiplet NPU. We describe the evaluation setup (sec:eval:set) and evaluate HEX-SIM in this section with two goals. First, we verified the correctness of HEX-SIM and evaluated the performance of different parallelisms given the same software and hardware (Sec. IV-C). We then explore the impact of varying important parameters in the hardware configurations and highlight the performance differences. We change the percentage of decoder chiplets (Sec. IV-E), number of PUs each chiplet owns, global buffer size (Sec. IV-F), and interconnection bandwidth (Sec. IV-G).

A. Validation

We validate HEX-SIM against an in-house RTL implementation of a real multi-chiplet system. We use matrix multiplication as the workload to test the inference cycles. The dataflow is in column parallelism. We have 4 chiplets, each with 2 PU inside. Network topology is mesh. As shown in Tbl. III, the figure the cycle counts obtained by both methods are in good agreement with each other.

B. Evaluation Setup

Software workloads setup. To evaluate HEX-SIM, we utilize three multi-modality large language models that are commonly employed in the robotic domain. Notice that although we mainly evaluate models that are widely used in the robotic domain, HEX-SIM can also serve as a versatile tool for simulating any multi-modality large language models.

- **Robotics Transformer (RT-1).** RT-1 [16] is one of the first several works training a transformer-based model to transfer knowledge from a large multi-modality dataset into downstream robot applications. It uses a neural network mixed with EfficientNet [43] to encode instruction and images and uses 8-layer Transformers [44] to decode robot actions.
- **RoboFlamingo (RF).** RF [29] is another vision language foundation model that can understand language instructions and perform accurate robot control. RF has achieved state-of-the-art performance on task-level manipulation tasks.
- **GR-1.** GR-1 [45] further enriches the type of tasks it allows the robot to execute. Both RF and GR-1 use Vision Transformer as encoders, with different network details. As for the decoder, they both use GPT-2.

Hardware setup. The chip we simulate in total has four different chiplets. We employ a RISC-V CPU for instruction dispatch, and a third-generation HBM is utilized for storing weights, inputs, and activations. For computing, we have two types of chiplets for encoding and decoding separately, each with 5 chiplets. The NoP bandwidth and NoC bandwidth are the same as the first multi-chiplet NPU Simba [24]. The bandwidth of HBM is extracted from its introduction paper [46].

For encoding, each chiplet has 16 PUs, a global buffer with a size to be 1536KB, and a RISC-V core [47]. Each PU is equipped with a 16×16 systolic array, and the size of the local buffer is 768KB evenly split for inputs, weights, and outputs.

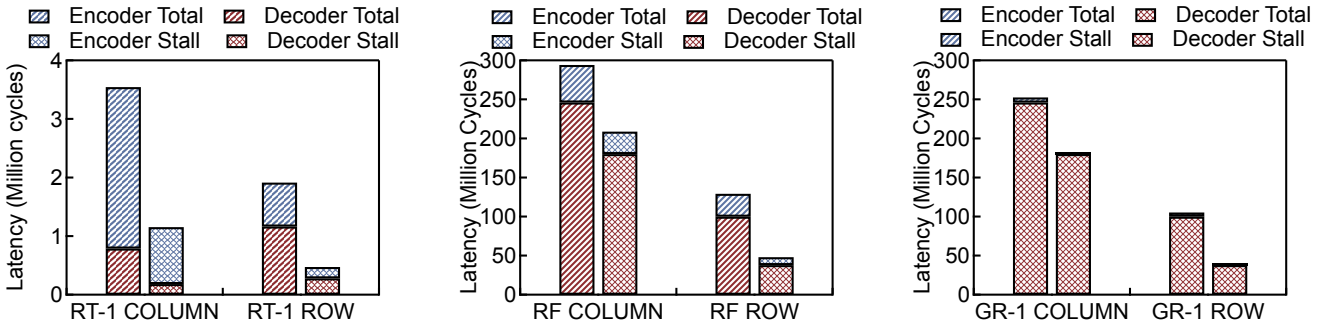
For decoding, each chiplet has 16 PUs, a global buffer with a size to be 3072KB, and a RISC-V core. Each PU is equipped with a 32×32 systolic array, and the size of the local buffer is 1536KB evenly split for inputs, weights, and outputs.

For the behavior of each systolic array, we use an RTL-validated single-core NPU simulation tool SCALE-SIM [28]. Although being cycle-accurate, the entire simulation runs fast. On average, we get the simulation results in around 6 hours running on an intel CORE i5-13600K.

C. Tensor Parallelism

As the scale of models continues to increase, the significance of large-scale multi-chiplet NPUs becomes more evident due to their potential for parallelism. We first compare total cycles consumed and stall cycles in Fig. 6. We find in all three cases, column parallelism costs longer latency compared to row parallelism. The difference in total cycles is 46.0%,56.1%, and 58.3%. Column parallelism also shows longer stall cycles. It stalls on average 281.7% longer compared to row parallelism.

Results suggest that in the workload we evaluate, row parallelism is a better choice compared to column parallelism. We find the key reason lies in the amount of data transmitted. In column parallelism, HEX-SIM only divides the number of filters, while in row parallelism, HEX-SIM divides the



(a) Inference latency of RT-1 with two different parallelisms.

(b) Inference latency of RoboFlamingo with two different parallelisms.

(c) Inference latency of RT-1 with two different parallelisms.

Fig. 6: Latency comparison on different tensor parallelisms. Both total cycles and stall cycles are reported.

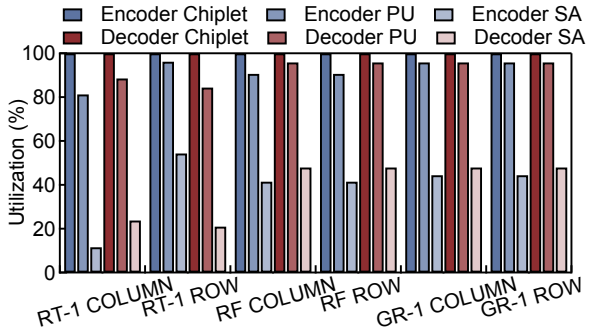


Fig. 7: Utilization for three levels of hardware, chiplet, PU, and systolic array (SA). Statistics of the encoder and decoder are shown separately.

number of channels in the input matrix and weight matrix, while large models usually have a large number of channels. This results in the input matrix size using column parallelism being N times larger than using row parallelism. Among them, N is the product of the number of chiplets and the number of PUs. However, row parallelism requires complex all-reduce operations on the output matrix and also requires a larger output bandwidth. This means when encountering neural networks with fewer channels and more filters, the effect of column parallelism is better than that of row parallelism. The reason is that on the one hand, the segmentation effect of column parallelism is greater than that of row parallelism, and on the other hand, the output bandwidth cannot meet the bandwidth required by row parallelism. Therefore, we believe that the choice of parallelism should be tailored to specific network types rather than adopting a fixed approach.

The utilization statistics support our findings. As shown in Fig. 7, both two parallelisms are having a high chiplet utilization. In all cases, the chiplet utilization is 100%, this means the scheduling works well and all the chiplets are used. PU utilization is also high. On average, the PU utilization is 91.3% for encoder stage and 92.6% for decoder stage. The utilization of PU is not perfect, as in some cases, there is no data scheduled on PU. In row parallelism, for example, weights are splited on the basis of channels, when the channels scheduled for each chiplet is smaller than the number of PUs, there will be idle PUs in this round of computation. We find this phenomenon is common in attention computation.

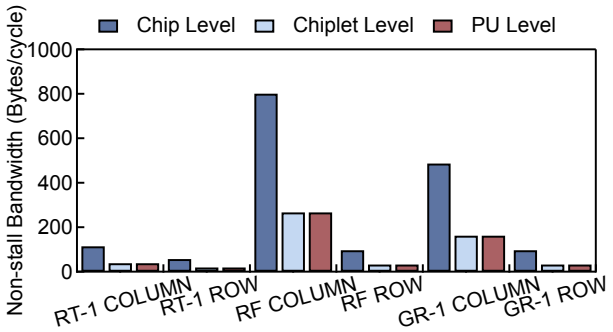
However, compared to the utilization of chiplets and PUs, the SA utilization is both in a relative low level. We witness an average utilization of 39.6% in encoder stage and a 35.4% in decoder stage, indicating that each systolic array is far from fully utilized. The reason is in two folds, first, the size of the matrix computation scheduled on each systolic array is usually smaller than the array size, even if we only use 16×16 and 32×32 MAC array. Second, the data communication rate can not catch up the computation speed. Thus, in many cases, the systolic array is stalled waiting to be fed with data.

Row parallelism is having a high utilization compared to column parallelism. The PU utilization of row parallelism is 4.9% higher compared to column parallelism, and the SA utilization is 10.1% higher. Both cases suggest that in the three networks we evaluate, row parallelism is a better topology as it significantly reduces the volum of data in communication. The results also validate that in row parallelism, the stall cycle is lower compared to the column parallelism.

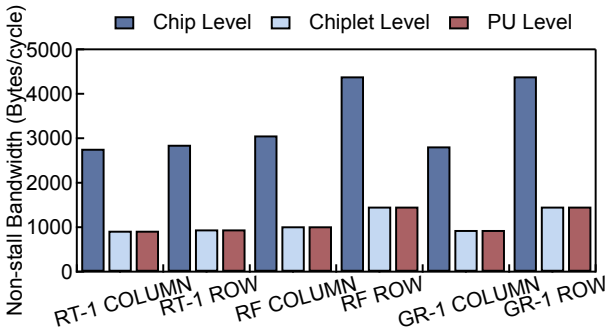
D. Non-stall Bandwidth

Stalling cycles take a significant amount of time. Our findings indicate that the primary reason for this is insufficient bandwidth. To demonstrate the bandwidth requirements for achieving non-stalling inference, we conducted an experiment. We show the results in Fig. 8 and separate the bandwidth requirements for input and output.

We can see that the minimum bandwidth for NoC to read and write LB is the same as the minimum bandwidth for LB to read and write systolic array, and the minimum read and write bandwidth for NoP to GB is three times that of NoC to LB. Meanwhile, the minimum write bandwidth required for RT-1 in row parallelism is 49.8% of the bandwidth in column parallelism. Similarly, the data for RF is 12.1% and GR-1 is 19.8%. In column parallelism, the minimum read bandwidth required for RT-1 is 96.9% when using row parallelism, RF is 69.8%, and GR-1 is 64.1%.

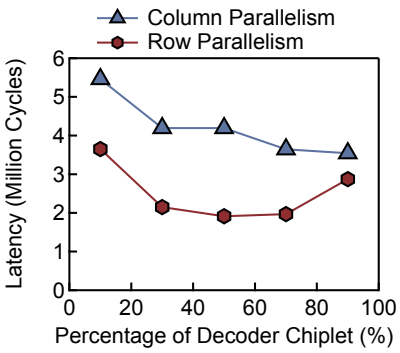


(a) Non-stall bandwidth requirements for data loading.

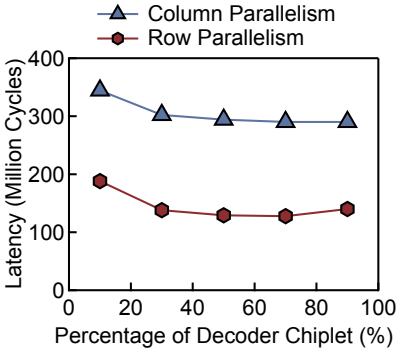


(b) Non-stall bandwidth requirements for data offloading.

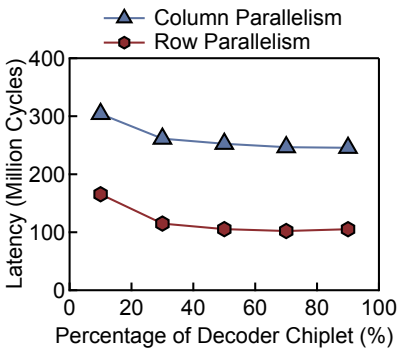
Fig. 8: Non-stall bandwidth requirements for input data loading and output data offloading.



(a) Inference latency change of RT-1 when the number of decoder chiplets change.



(b) Inference latency change of RoboFlamingo when the number of decoder chiplets changes.



(c) Inference latency change of GR-1 when the number of decoder chiplets changes.

Fig. 9: Latency changes when the percentage of decoder chiplet changes. X-axis means the percentage of decoder chiplets compared to total computing chiplets.

As we expected, at both read and write stage, the bandwidth requirements at chip layer is the highest. When gathering the output data, the non-stop bandwidth requirement reaches up to over 4000 Bytes per cycle, which is far beyond the upper bond of current technology. An effective way of lowering the burden on chip layer interconnection is to capture and utilize data locality more inside each chiplet.

E. Decoder Chiplet Scale

HEX-SIM provides simulation for heterogeneous multi-chiplet NPUs, which is a design trend, as robotic applications are using different sensor types. In our current evaluation, the encoder and decoder exhibit different chiplet designs. A natural design decision would be given the fixed chip area, increasing the scale of which type of chiplets would benefit the overall performance.

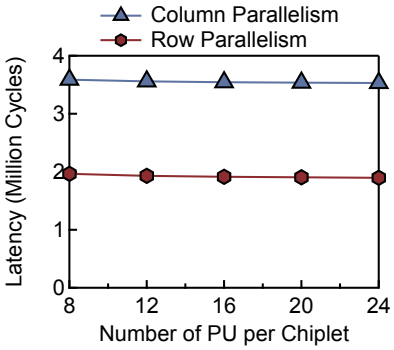
We gradually increase the number of decoder chiplets and re-run our simulation. We show the results in Fig. 9, where the x-axis represents the percentage of decoder chiplets used in total computing chiplets. Results show that the performance increase by 30.7% when the percentage of decoder chiplets increases from 10.0% to 70.0%. The performance improvement converges when further increasing resources on the decoder. For RT-1, further, increasing the percentage to 90.0% can even hurt the performance.

The result suggests that we use more chiplets for the decoder than for the encoder. The main reason for this is that the neural network in the decoder part has a larger number of parameters than the encoder part. The decoder parameters of RT-1, RF, and GR-1 account for 61.3%, 83.4%, and 94.7% respectively, so that the performance increases as the decoder ratio increases. Meanwhile, we also find that performance growth is slow when it accounts for 70.0% -90.0% of the total. This is because parallelism cannot be linearly increased based on the number of chiplets.

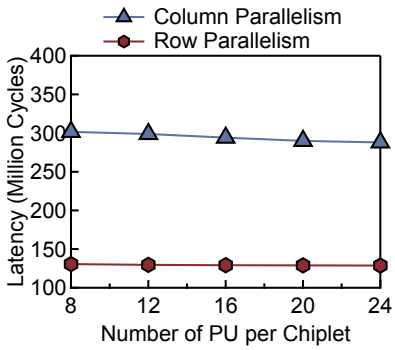
We find that in the case of RT-1, when the number of decoder chiplets exceeds 70% of the total chiplets, the runtime actually increases. This is because RT-1 is a relatively early model, with parameters in encoder and decoder to be the same. Thus, a fixed encoder decoder distribution may not be the optimal solution if runtime reconfiguration is possible.

F. PU Number and Global Buffer Size

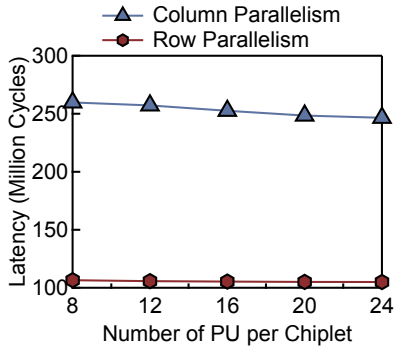
The number of PUs and size of the global buffer determines the computing capability of one single chiplet, yet stacking more computing and storage also increases the size of the chip. The stacked resources are wasted if not fully utilized. To evaluate the impact, we sweep the number of PUs and the size of global buffer.



(a) Inference latency of RT-1 when the number of PUs change.

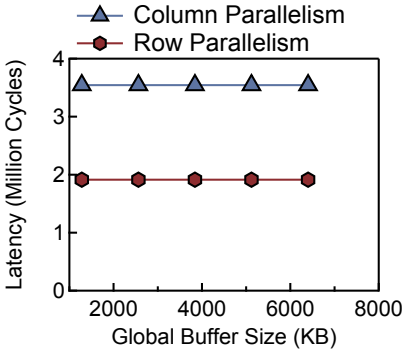


(b) Inference latency of RoboFlamingo when the number of PUs change.

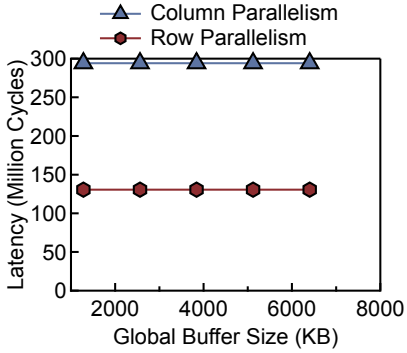


(c) Inference latency of RT-1 when the number of PUs change.

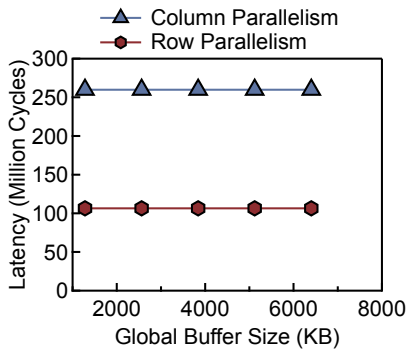
Fig. 10: Latency changes when the number of PUs changes. The X-axis means the number of PUs in one chiplet.



(a) Inference latency of RT-1 when global buffer size changes.



(b) Inference latency of RoboFlamingo when global buffer size changes.



(c) Inference latency of RT-1 when global buffer size changes.

Fig. 11: Latency changes when the size of the global buffer changes. X-axis means the size of the global buffer.

Fig. 10 shows the performance variation when increasing the number of PUs in one chiplet. We scale both encoder and decoder chiplets. When the number of PU in the chiplet increases by 4 each time, its average performance improves by 1.0%, 0.9%, 0.7%, and 0.4%. This is very obvious, the increase in PU has little effect on performance improvement, and the magnitude of performance improvement is gradually decreasing. The reason for this phenomenon is that the read and write bandwidth of GB is limited, and it cannot provide enough data for PU calculation. Moreover, the increase in PU cannot linearly improve parallelism, resulting in a smaller performance improvement.

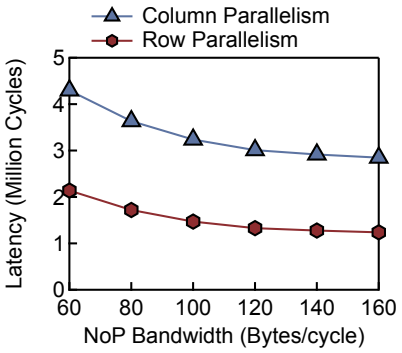
Fig. 11 shows the result on increasing the size of GB. We enlarge the GB size from 1280 KB to 6400 KB. Surprisingly, although the size of GB is increasing, performance has not been improved. This is because the GB read and write time is composed of non-stall time and stall time. The non-stall time for reading is obtained by dividing the transmitted data by the bandwidth, where the transmitted data and bandwidth do not change with the size of the GB. Then, the stall time is the difference between the division value of data and write bandwidth and the division value of data and read bandwidth. If the write bandwidth is greater than the read bandwidth, the stall time is 0. But in either case, the stall time is independent of the size of the GB. Thus, when the bandwidth is low, increasing the size of GB does not help.

G. Effect of NoP and NoC Bandwidth

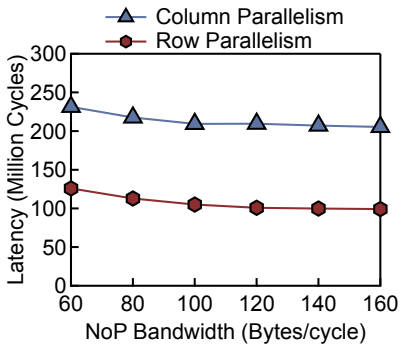
In the above subsections, we have observed that increasing the number of processing units (PUs) and the size of global buffers (GBs) does not have a significant impact on overall performance. We here further analyze the relationship between larger bandwidth with the performance. Notice that given the current technology, the NoP and NoC bandwidth we use in this evaluation may not be satisfied in real products, with some of the results being hypothetical. Therefore, these findings are intended to provide insights for chip designers rather than representing actual performance.

Fig. 12 shows the performance variation with increased NoP bandwidth. We tested the bandwidth of NoP from the 60 Bytes per CLK to 160 Bytes per CLK. The average performance improvements in each interval were 12.4%, 8.7%, 5.4%, 2.0%, and 1.5%, respectively. Performance does indeed increase with the increase of NoP bandwidth, yet the performance improvement is extremely limited when the NoP bandwidth exceeds 120 Bytes per CLK.

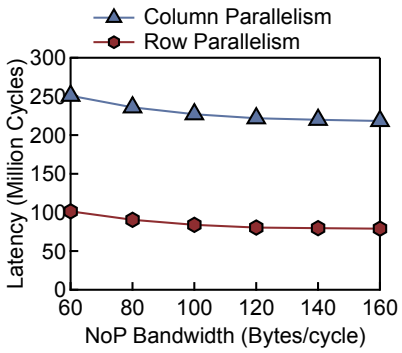
As shown in Fig. 13, similar to NoP bandwidth, increasing NoC bandwidth can also improve performance. We tested



(a) Inference latency of RT-1 when NoP bandwidth changes.

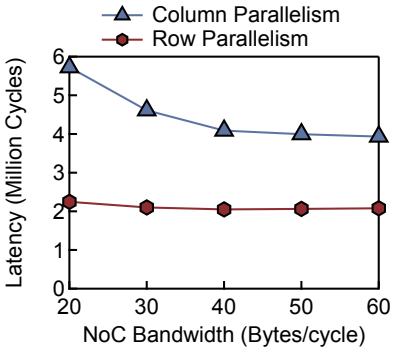


(b) Inference latency of RF when NoP bandwidth changes.

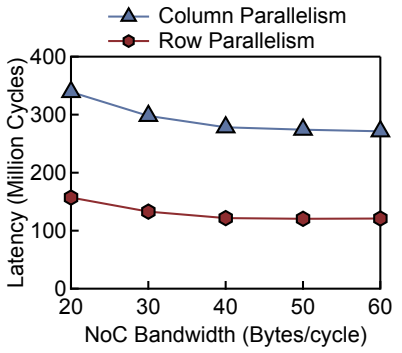


(c) Inference latency of GR-1 when NoP bandwidth changes.

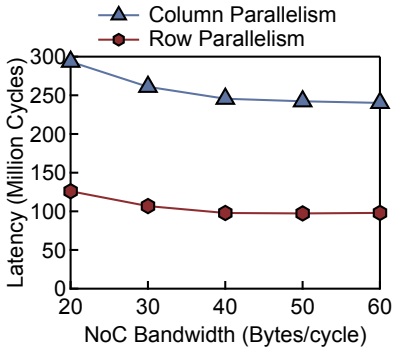
Fig. 12: Latency changes when the bandwidth of NoP changes.



(a) Inference latency of RT-1 when NoC bandwidth changes.



(b) Inference latency of RF when NoC bandwidth changes.



(c) Inference latency of GR-1 when NoC bandwidth changes.

Fig. 13: Latency changes when the bandwidth of NoC changes.

the NoC bandwidth range from 20 Bytes to 60 Bytes per CLK, with average performance improvements of 13.3%, 7.2%, 1.0%, and 0.3% for each interval, respectively. The improvements converge NoC bandwidth reaches 40 Bytes per CLK.

Both trends indicate that higher bandwidth NoP and NoC will provide sufficient data flow for the systolic array to perform calculations, and thus resulting in improved performance, particularly in scenarios with high data communication volume like column parallelism. The performance improvement gradually converges as the bandwidth of LB to SA does not scale with the bandwidth of NoP or NoC, the bottleneck thus transfer from data communication between chiplets and PUs to data communication inside PUs. However, we can still get the conclusions that improving bandwidth is the most effective approach of reducing runtime latency for a multi-chiplet NPU.

H. Real Hardware Evaluation

As shown in Tbl. IV, we simulate SIMBA, the first multi-chiplet NPU chip. We use all the parameters reported from SIMBA's original paper, which achieves a theoretical peak performance at 128 TOPS.

It is worth noting that the RF we selected has a larger number of model parameters in the encoder stage than GR1, but the actual inference performance of RF is slightly lower than that of GR1. This is because larger data volumes require higher bandwidth. Therefore, the actual inference performance of RF on SIMBA is lower than that of GR1.

TABLE IV: Real hardware evaluation, indicating the inference performance of different neural networks on Simba.

Network	Latency(ms)	Actural Performance(TOPS)
RT1	1.84	31.61
RF	183.60	56.85
GR1	157.48	58.12

V. Related Work

Multi-modality large language models for robotic applications. Using large models to control robots show great potentials in extending the usability of robots in complex tasks. Most state-of-the-art methods are applying multi-modal large language models in their solution. These large language models usually have multiple modalities such as vision and human language instructions and are consuming significant high inference latency [48–50]. Accelerating these multi-modality large language models is thus critical for real-time and low-power robotic applications [51].

Simulating performance of NPUs. Since NPUs have been widely used in data center and edge devices, simulating the

performance of different NPUs are proposed by the community. Tools like SCALE-SIM [28] first incorporate different data flows and provide cycle-accurate simulation for a single-core NPU and mNPUsim [38] further extend the simulating methodology to multi-core. SIAM [52] belongs to multi-chiplet simulation. However, SIAM only simulates in-memory computing (IMC) architectures. Our work provides the first multi-chiplet simulator for non-IMC architecture, which is still the major design methodology in most real-world products. To our knowledge, HEX-SIM provides the first cycle-accurate non-IMC architecture simulation tool for a heterogeneous multi-chiplet NPU.

Multi-chiplet chip design. Designing large monolithic chips to enhance chip performance has been a traditional approach to improving chip performance. However, larger die sizes of these chips can negatively impact yield and drive up costs. To address this, multi-chiplet chip design has been widely adopted in commercial products, such as AMD ZEN CPUs [53] and Radeon GPUs [54]. We have observed the same design methodology being utilized in NPU designs, such as Simba and Gemini [24, 55]. This emerging trend further motivates our work, with a particular focus on developing tools and simulations in this field.

VI. DISCUSSION

Chiplet has become an effective way of integrating heterogeneous chips into one computing system. In this work, we integrate four different chiplets to form an NPU to support the inference of multi-modal large language models that are widely adopted in the robotic domain.

We also support the flexibility for the users to integrate their own chiplets. To do so, users need to program a template file for the users to design their customized chiplet. The template file contains all needed parameters such as connectivity models, the number of PUs, the size of the global buffer, NoC bandwidth, and the hardware architecture of PUs. The type of PU needs to be within the current library we support, with detailed parameters such as the size of the systolic array, dataflow, the size of the local buffer, etc. If the users want to incorporate a chiplet out of our library, such as an in-memory computing unit, they can also be plugged in through an additional cycle-accurate simulator of the PU itself.

VII. CONCLUSION

Motivated by the increase of parameters and modality of neural networks, this work presents a cycle-accurate simulator for multi-chiplet NPUs. Using HEX-SIM, users can effectively test the performance and optimize their NPU designs. Through thorough evaluation using HEX-SIM on three state-of-the-arts multi-modality large language models, we find that the chiplets used for the decoder and interconnection bandwidth contribute most to performance improvements, while the size of global buffers and number of PUs are bounded by the above two parameters. The code is available at https://github.com/jimrelief/HEX-SIM.

VIII. ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62025404, 62104229).

REFERENCES

[1] A. A. Shvets, A. Rakhlin, A. A. Kalinin, and V. I. Iglovikov, "Automatic instrument segmentation in robot-assisted surgery using deep learning," in *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 2018, pp. 624–628.

[2] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, 2017.

[3] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.*, "The limits and potentials of deep learning for robotics," *The International journal of robotics research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[4] P. Katara, Z. Xian, and K. Fragkiadaki, "Gen2sim: Scaling up robot learning in simulation with generative models," 2023.

[5] L. Caballero, Á. Perafan, M. Rinaldy, and W. Percybrooks, "Predicting the energy consumption of a robot in an exploration task using optimized neural networks," *Electronics*, vol. 10, no. 8, p. 920, 2021.

[6] M. Wei and V. Isler, "Predicting energy consumption of ground robots on uneven terrains," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 594–601, 2021.

[7] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *CoRR*, vol. abs/1711.07128, 2017. [Online]. Available: http://arxiv.org/abs/1711.07128

[8] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *International conference on artificial neural networks*. Springer, 2007, pp. 220–229.

[9] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting." in *Interspeech*, 2015, pp. 1478–1482.

[10] R. C. Schank, "Conceptual dependency: A theory of natural language understanding," *Cognitive psychology*, vol. 3, no. 4, pp. 552–631, 1972.

[11] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[12] M. Canonico, L. De Russis *et al.*, "A comparison and critique of natural language understanding tools," in *Cloud Computing 2018*, 2018, pp. 110–115.

[13] Z. Song, Q. Chen, Z. Huang, Y. Hua, and S. Yan, "Contextualizing object detection and classification," in *CVPR 2011*. IEEE, 2011, pp. 1585–1592.

[14] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the*

IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10 781–10 790.

[15] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE, 2020, pp. 237–242.

[16] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. Ryoo, G. Salazar, P. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich, "Rt-1: Robotics transformer for real-world control at scale," in *arXiv preprint arXiv:2212.06817*, 2022.

[17] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, "Palm-e: An embodied multimodal language model," 2023.

[18] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for google's training chips: Tpuv2 and tpuv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, 2021.

[19] Z. Boulkenafet, J. Komulainen, L. Li, X. Feng, and A. Hadid, "Oulu-npu: A mobile face presentation attack database with real-world variations," in *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*. IEEE, 2017, pp. 612–618.

[20] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 1–14.

[21] C.-T. Chu, S. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Ng, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems*, vol. 19, 2006.

[22] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," in *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 2005, pp. 408–419.

[23] P. Gepner and M. F. Kowalik, "Multi-core processors: New way to achieve high system performance," in *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*. IEEE, 2006, pp. 9–13.

[24] Y. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. R. Pinckney, and P. Raina, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *International Symposium on Microarchitecture*, 2019.

[25] Z. Zhuang, B. Yu, K.-Y. Chao, and T.-Y. Ho, "Multi-package co-design for chiplet integration," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[26] H. Zhi, X. Xu, W. Han, Z. Gao, X. Wang, M. Palesi, A. K. Singh, and L. Huang, "A methodology for simulating multi-chiplet systems using open-source simulators," in *Proceedings of the Eight Annual ACM International Conference on Nanoscale Computing and Communication*, 2021, pp. 1–6.

[27] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: preparing for a new exponential," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 67–72.

[28] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 58–68.

[29] X. Li, M. Liu, H. Zhang, C. Yu, J. Xu, H. Wu, C. Cheang, Y. Jing, W. Zhang, H. Liu, H. Li, and T. Kong, "Vision-language foundation models as effective robot imitators," *arXiv preprint arXiv:2311.01378*, 2023.

[30] G. Lauterbach, "The path to successful wafer-scale integration: The cerebras story," *IEEE Micro*, vol. 41, no. 6, pp. 52–57, 2021.

[31] R. Yousry, E. Chen, Y.-M. Ying, M. Abdullatif, M. Elbadry, A. ElShater, T.-B. Liu, J. Lee, D. Ramachandran, K. Wang *et al.*, "11.1 a 1.7 pj/b 112gb/s xsr transceiver for intra-package communication in 7nm finfet technology," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 180–182.

[32] L. T. Su, S. Naffziger, and M. Papermaster, "Multi-chip technologies to unleash computing performance gains over the next decade," in *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2017, pp. 1–1.

[33] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[34] H. Min, J. Kwon, and B. Egger, "Flexer: Out-of-order scheduling for multi-npus," in *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, 2023, pp. 212–223.

[35] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "Openpiton: An open source manycore research framework," *SIGPLAN Not.*, vol. 51, no. 4, p. 217–232, mar 2016. [Online]. Available: https://doi.org/10.1145/2954679.2872414

[36] M. Olyaiy, C. Ng, A. S. Fedorova, and M. Lis, "Sunstone: A scalable and versatile scheduler for mapping tensor

algebra on spatial accelerators," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 259–271.

[37] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[38] S. Hwang, S. Lee, J. Kim, H. Kim, and J. Huh, "mNPUsim: Evaluating the Effect of Sharing Resources in Multi-core NPUs," in *2023 IEEE International Symposium on Workload Charcterization (IISWC)*, 2023.

[39] L. Peng, W. Shi, J. Zhang, and S. Irving, "Exploiting model-level parallelism in recurrent neural network accelerators," in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2019, pp. 241–248.

[40] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[41] M. Shoeybi, M. Patwary, R. Puri, P. Legresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using gpu model parallelism," 2019.

[42] B. Wang, Q. Xu, Z. Bian, and Y. You, "Tesseract: Parallelize the tensor parallelism efficiently," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[43] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[45] H. Wu, Y. Jing, C. Cheang, G. Chen, J. Xu, X. Li, M. Liu, H. Li, and T. Kong, "Unleashing large-scale video generative pre-training for visual robot manipulation," 2023.

[46] M.-J. Park, J. Lee, K. Cho, J. Park, J. Moon, S.-H. Lee, T.-K. Kim, S. Oh, S. Choi, Y. Choi, H. S. Cho, T. Yun, Y. J. Koo, J.-S. Lee, B.-K. Yoon, Y.-J. Park, S. Oh, C. K. Lee, S.-H. Lee, H.-W. Kim, Y. Ju, S.-K. Lim, K. Y. Lee, S.-H. Lee, W. S. We, S. Kim, S. M. Yang, K. Lee, I.-K. Kim, Y. Jeon, J.-H. Park, J. C. Yun, S. Kim, D.-Y. Lee, S.-H. Oh, J.-H. Shin, Y. Lee, J. Jang, and J. Cho, "A 192-gb 12-high 896-gb/s hbm3 dram with a tsv auto-calibration scheme and machine-learning-based layout optimization," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 256–269, 2023.

[47] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[48] W. Dai, J. Li, D. Li, A. M. H. Tiong, J. Zhao, W. Wang, B. Li, P. N. Fung, and S. Hoi, "Instructblip: Towards general-purpose vision-language models with instruction tuning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[49] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman *et al.*, "Foundation models in robotics: Applications, challenges, and the future," *arXiv preprint arXiv:2312.07843*, 2023.

[50] K. Kawaharazuka, T. Matsushima, A. Gambardella, J. Guo, C. Paxton, and A. Zeng, "Real-world robot applications of foundation models: A review," *arXiv preprint arXiv:2402.05741*, 2024.

[51] K. Asadi, H. Ramshankar, H. Pullagurla, A. Bhandare, S. Shanbhag, P. Mehta, S. Kundu, K. Han, E. Lobaton, and T. Wu, "Vision-based integrated mobile robotic system for real-time applications in construction," *Automation in Construction*, vol. 96, pp. 470–482, 2018.

[52] G. Krishnan, S. K. Mandal, M. Pannala, C. Chakrabarti, J. Seo, Ü. Y. Ogras, and Y. Cao, "SIAM: chiplet-based scalable in-memory acceleration with mesh for deep neural networks," *CoRR*, vol. abs/2108.08903, 2021. [Online]. Available: https://arxiv.org/abs/2108.08903

[53] R. Bhargava and K. Troester, "Amd next generation" zen 4" core and 4 th gen amd epyc™ server cpus," *IEEE Micro*, 2024.

[54] S. Dasgupta, T. Singh, A. Jain, S. Naffziger, D. John, C. Bisht, and P. Jayaraman, "8.4 radeon rx 5700 series: The amd 7nm energy-efficient high-performance gpus," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 150–152.

[55] J. Cai, Z. Wu, S. Peng, Y. Wei, Z. Tan, G. Shi, M. Gao, and K. Ma, "Gemini: Mapping and architecture co-exploration for large-scale dnn chiplet accelerators," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 156–171.