Carat: Unlocking Value-Level Parallelism for Multiplier-Free GEMMs

Zhewen Pan
zhewen.pan@wisc.edu
University of Wisconsin–Madison
Madison, WI, USA

Joshua San Miguel
jsanmiguel@wisc.edu
University of Wisconsin–Madison
Madison, WI, USA

Di Wu*
di.wu@ucf.edu
University of Central Florida
Orlando, FL, USA

Abstract

In recent years, hardware architectures optimized for general matrix multiplication (GEMM) have been well studied to deliver better performance and efficiency for deep neural networks. With trends towards batched, low-precision data, e.g., FP8 format in this work, we observe that there is growing untapped potential for value reuse. We propose a novel computing paradigm, value-level parallelism, whereby unique products are computed only once, and different inputs subscribe to (select) their products via temporal coding. Our architecture, Carat, employs value-level parallelism and transforms multiplication into accumulation, performing GEMMs with efficient multiplier-free hardware. Experiments show that, on average, Carat improves iso-area throughput and energy efficiency by 1.02× and 1.06× over a systolic array and 3.2× and 4.3× when scaled up to multiple nodes.

CCS Concepts: • **Computer systems organization** → **Neural networks**; **Data flow architectures**; • **Hardware** → **Emerging architectures**; **Arithmetic and datapath circuits**; **Application specific integrated circuits**.

Keywords: value-level parallelism, value reuse, temporal computing, low-precision, batch processing, multiplier-free

ACM Reference Format:
Zhewen Pan, Joshua San Miguel, and Di Wu. 2024. Carat: Unlocking Value-Level Parallelism for Multiplier-Free GEMMs. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 18 pages.
https://doi.org/10.1145/3620665.3640364

*Contributed to this project while at UW–Madison.

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0385-0/24/04.
https://doi.org/10.1145/3620665.3640364

1 Introduction

With recent advances in deep learning, deep neural networks (DNNs) have become ubiquitous in a vast array of application domains. To improve performance and efficiency, existing DNN accelerator architectures usually focus on the most abundant operation: general matrix multiply (GEMM). There exist two popular data-centric optimizations to boost the performance and efficiency for GEMM. First, data reuse minimizes the data movement between compute units and memory hierarchies, thus improving memory efficiency [9, 33, 71]. In particular, batch inference is leveraged to enhance the reuse of weights in DNNs [13, 20, 35]. Second, low data precision reduces the data size, so that both the compute unit size and the memory footprint are smaller, improving both the compute and memory efficiency. Multiple low-precision data formats, e.g., INT8 [33] and BF16 [34], have been well studied to replace the standard IEEE FP32 format, with negligible DNN accuracy drop. Very recently, more aggressive data formats, e.g., INT4 [18, 31, 38, 75] and FP8 [39, 48, 53, 65], have been proposed for DNN inference. Among these, FP8 hardware has already been commercialized for deep learning [3, 32, 54], with the relevant results given in Table 1. The measurement results demonstrate the effectiveness of FP8, which is significantly better than the well-studied INT8 in large language models.

In real-world cases, traditional DNN inference, e.g., computer vision and natural language processing, has worked towards increased batch sizes, 64~512 [10, 12, 21, 36, 63, 64], without violating system latency requirements. Adopting low data precision further reduces the latency and increases the batch size under a certain latency budget [22]. There is a recent trend for applications to go beyond this, including financial technologies [2, 26, 41], medical diagnosis [4, 5, 62], judicial systems [1, 8, 37], etc. These applications usually emphasize more on higher throughput with larger batch sizes than traditional applications. One such example is credit scoring: 149 million Chase credit cards need monthly-scheduled credit scoring [7, 57], translating to approximately 5 million latency-insensitive inferences per day. Even in latency-sensitive high-frequency trading (HFT), higher throughput is desirable as it allows more queries to be processed, yielding better trading decisions and, therefore, more profits. Existing HFT systems have already fed on both batched and

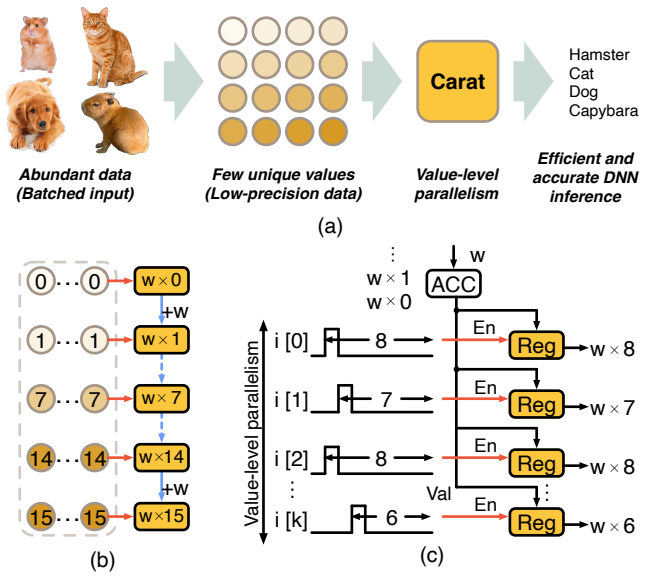


Figure 1. Illustration of value-level parallelism. (a) Value-level parallelism synergizes two emerging trends: data are both more abundant and lower precision. (b) **Concept**: A vector input multiplies a scalar weight w. Multiplication is transformed to accumulation. All unique products are calculated only once, and product values are reused, shown in yellow rectangles for all 4-bit inputs $\in [0, 15]$. Each circled input element subscribes to (selects) its product, i.e., each unique product is reused by multiple inputs with the same value. (c) **Architecture**: The weight is accumulated (ACC) over time, and each vector input subscribes to its product (Val) by selecting the weight accumulation result via a temporal signal, whose spike timing depends on the input value.

low-precision data to achieve simultaneously high throughput and efficiency under latency constraints [76]. Thus, our work leverages *abundant batched data* and *low data precision*, taking advantage of their increased popularity and utility towards future DNN inference systems.

Insight. Though existing GEMM optimizations already demonstrate high performance, we believe there is still untapped potential. Our insight is that *not all performed computations are mandatory*. Let us consider vector-scalar multiplication, which can be stacked towards a complete GEMM, as an example. Assuming an INT4 input vector of size 1024, multiplying this vector with a scalar weight in conventional hardware needs 1024 multiplications. However, this is fundamentally inefficient since the final product vector only has at most 16 unique values; effectively, the multiplications are 64× (i.e., 1024/16) more than needed. For batch processing (up to 256 for DNN inference [13, 20, 35] and several Ks for training [42, 45, 56]) where multiple input vectors are multiplied with the same weight simultaneously, such a waste of compute is even more severe.

Table 1. DNN evaluation with FP8 multiplication and BF16 accumulation, reported by Arm, Intel and NVIDIA in [48]. The adopted FP8 for DNN inference has a 4-bit exponent and a 3-bit mantissa (E4M3). IC/LT/NLP denote image classification, language translation and natural language processing, respectively. For all metrics but perplexity, higher values are better. Data marked with * are from [72, 73].

Model	Size	Task	Metric	Metric measurement		
				BF16	FP8	INT8
VGG16	138M	IC	Accuracy	71.27	71.11	70.75*
Resnet50	26M			76.71	76.76	75.82*
GNMT	255M	LT	BLEU	24.83	24.65	24.53*
Transformer	165M			26.87	26.83	21.23*
BERT	110M	NLP	F1	88.19	88.09	76.89
Transformer-XL	0.46B	NLP	Perplexity	22.98	22.99	–
GPT	175B			6.65	6.68	–
GPT3	6.7B			8.51	8.41	10.29

Table 2. Comparison of approaches to exploit parallelism.

Parallelism level	Source of opportunity	Example
Instruction	Independent instructions	CPUs, GPUs, etc.
Thread	Independent threads	
Memory	Concurrent memory accesses	Caches, etc.
Data	Vectorized data	SIMD, etc.
Value	Few unique values for abundant data	Carat

Value-Level Parallelism. In this work, we propose *value-level parallelism* that enables GEMM computations without any multiplier in hardware.[1] We leverage a form of *value reuse* as in Figure 1, i.e., *unique products are computed only once and reused by the entire input vector*. This consists of three steps, as shown in Figure 1 (b). First, we transform the multiplication with the scalar (here, weight) to the accumulation of the scalar. The accumulation will traverse all unique outputs in order, e.g., in this case, from $w\times0$ to $w\times15$ (INT4). Second, we cluster the elements of the input vector according to their value, e.g., all input 0s are clustered together, to prepare for value reuse. Third, all elements with an identical value effectively *subscribe* to their corresponding accumulation result (partial product), e.g., all inputs with a value 3 simply wait until the accumulation has reached $w\times3$. Figure 1 (c) gives an example architecture, where different inputs subscribe to their products in parallel. Compared to traditional forms of parallelism (Table 2), value-level parallelism is timely given the trends towards larger datasets with lower precisions. Transforming multiplications to accumulations eliminates the need for an array of multipliers in hardware and is particularly efficient when the number of unique values is low.

Carat Architecture. Leveraging value-level parallelism, we propose Carat, a *multiplier-free* architecture for GEMMs. *The recent trends in DNN inference towards both (1) abundant*

[1]Prior DNN accelerators require either single-cycle bit-parallel or multi-cycle bit-serial multipliers to perform multiplications [9, 44].

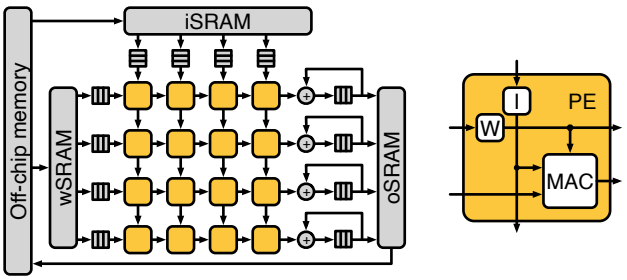


Figure 2. An example systolic array with weight stationary dataflow [33]. Each processing element (PE) contains a multiply-accumulate (MAC) unit and required pipeline buffers. Weights (W) are stationary in a PE, while inputs (I) and outputs flow in and out from the top and right of the PE.

batched data and (2) lower data precision unlock potential for higher value reuse. First, batch processing creates more opportunities for multiple independent inputs to interact with the same weight, i.e., vector-scalar multiplication. Second, low data precision can reduce the number of unique data values. We consider commercialized FP8 in this work, which introduces negligible accuracy loss according to Table 1. Note that value reuse is fundamentally distinct from data reuse; the former pertains to techniques that leverage values common to multiple data elements, while the latter pertains to techniques (e.g., caching) that leverage data elements reused by multiple operations. Our Carat architecture leverages both of these orthogonal concepts for improved performance and efficiency.

Contributions. Our contributions are as follows:

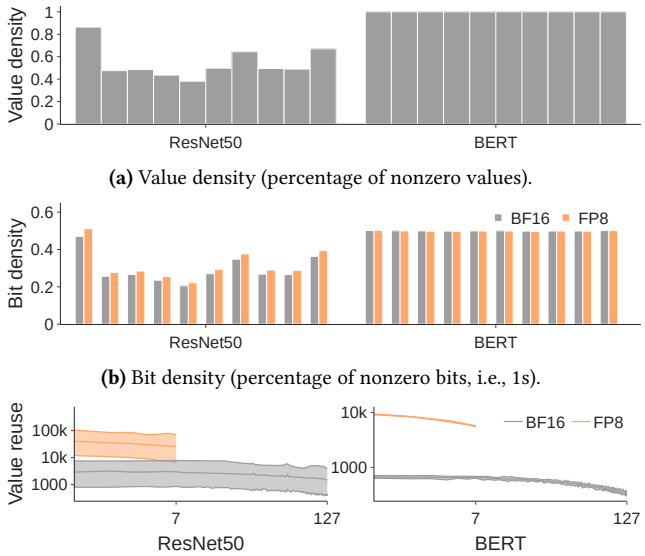
- We shed light on a growing opportunity of *redundant computations* in GEMM processing that stems from batched data and low data precision in DNNs.
- We are the first to propose *multiplier-free* computation for GEMMs.
- We present and evaluate Carat with value reuse that achieves a novel form of *value-level parallelism*.

This paper is organized as follows. Section 2 articulates the motivation. Then Section 3 and Section 4 describe the concept of value reuse via temporal coding and the details of Carat architecture. The following Section 5 and Section 6 evaluate the implementation. Finally, Section 7 and Section 8 discuss and conclude this work.

2 Opportunities for Value Reuse

Existing GEMM hardware has leveraged data-centric techniques for improved efficiency. One such technique is spatial dataflow, with an example in Figure 2. This architecture is designed to maximize data reuse in dense GEMMs [33, 60]. To accelerate GEMMs further, sparse architectures are proposed to skip unnecessary computation upon zero values [9].

We show the exploitable opportunity for sparse acceleration in Figure 3 (a). We profile the outputs of the first



(a) Value density (percentage of nonzero values).

(b) Bit density (percentage of nonzero bits, i.e., 1s).

(c) Value reuse. The x-axis ticks are unique mantissa values, which are maximally 127 and 7 for BF16 and FP8. The y axis is in log scale.

Figure 3. Opportunities for GEMM acceleration. (a) Vision models with ReLU activation [50] have high value sparsity, which is the opposite of value density (y-axis). In contrast, language models without ReLU have almost zero value sparsity. (b) Due to higher value sparsity, vision models also exhibit higher bit sparsity, opposite of bit density, than language models. (c) The opportunity for value reuse is the number of inputs for each unique mantissa value. More details are explained in Section 3 and Section 4.5. From top to bottom, three curves for each color are the maximum, average and minimum reuse opportunities in interested layers.

10 GEMM layers (using random inputs from dataset) from two pretrained DNNs, including ResNet50 [27] on ImageNet dataset [15] for computer vision, and BERT [16] on multiple datasets for natural language processing. We observe that language models do not exhibit rich value sparsity as in vision models; thus prior sparse acceleration techniques for vision models can be considerably less effective on language models. Moreover, as shown in Figure 3 (b), opportunities for bit sparsity (i.e., fine-grained value sparsity) are also limited in language models [44].

Our work is motivated by deep learning trends towards batched, low-precision data, as exemplified in Section 1. We show the exploitable opportunity for value reuse in Figure 3 (c) and observe that: (1) lower data precision exposes more opportunities, which will become even richer with a larger batch size; (2) both vision and language models exhibit significant opportunities for value reuse.

3 Value-Level Parallelism

In this work, we explore value-level parallelism, a computing paradigm that leverages value reuse and temporal coding

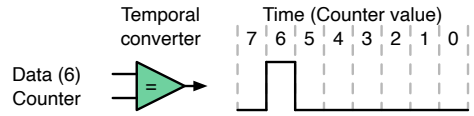


Figure 4. A temporal signal, which is generated by comparing source binary data with a deterministic counter output at each cycle, i.e., temporal converter. In this example, as the 3-bit binary data is 6, the temporal signal spans across $2^3 = 8$ cycles; and a spike (logic-1) only occurs at the 6-th cycle, when the data value equals the counter value.

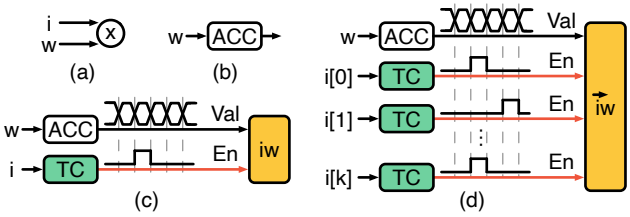


Figure 5. Value-level parallelism for vector-scalar multiplication. TC denotes a temporal converter in Figure 4. ACC denotes an accumulator. Yellow blocks are output product registers, with Val and En referring to the write value (a partial product) and write enable (a temporal signal). (a) Multiplication of an input i and a weight w. (b) Accumulation of the weight w to obtain all partial products. (c) The temporal converter generates a temporal signal for the input i as in Figure 4. At the i-th cycle, a spike occurs, and the weight is accumulated i times, i.e., the partial product is $i \cdot w$. Therefore, $i \cdot w$ is written/selected as the result. (d) Given an input vector $\vec{i}$, the temporal signal for each input $i[k]$ can independently select the product between $i[k]$ and the weight w, achieving value-level parallelism.

(Figure 4) for GEMM computations. Figure 5 shows how to perform integer vector-scalar multiplication via value reuse. It consists of three steps. First, from Figure 5 (a) to (b), we transform the multiplication into an accumulation over time. At each cycle, the weight accumulation result is a partial product for a specific input value. Next, from Figure 5 (b) to (c), we use the temporal signal for the input to subscribe to (select) its partial product. Temporal coding is a popular data encoding scheme in low power computing paradigms [11, 69]. It encodes the information as time-to-first-spike [17, 23, 46, 51, 66–70]. Figure 4 gives an example temporal-coded datum, whose value equals the timing when a spike occurs. Finally, from Figure 5 (c) to (d), different elements from the input vector will perform the previous subscription (selection) step independently to obtain their products, with an example circuit shown in Figure 1 (c). As multiple parallel inputs can reuse one computed output, regardless of the input values, we coin this paradigm as value-level parallelism.

Given the size (height here) of the input vector, H, and the the number of bits to be temporalized, M, the opportunity for value reuse (i.e., the number of inputs that reuse the same product) is $\frac{H}{2^M}$, where 2^M is the maximum rounds of weight accumulation, i.e., the number of unique input values or output products. Longer vectors (larger H) and lower data precision (smaller M) expose higher opportunities for value reuse linearly and exponentially.

4 Carat Architecture

We introduce our GEMM architecture with value reuse, named Carat, to unlock the rich value-level parallelism in batch DNN inference with low-precision data. This implementation adopts FP8 and BF16 format for multiplication and accumulation. In this section, we start with a high-level Carat architecture overview. Then we describe microarchitecture designs in Section 4.2–4.6, followed by a walkthrough example in Section 4.7. Finally, we show how Carat can scale up to a multi-node Network-on-Chip (NoC)-based system in Section 4.8.

4.1 Overview

As shown in Figure 6 (a), Carat has a PE array organization. Its memory hierarchy is similar to prior systolic array-based DNN accelerators [33], which have off-chip memory, e.g., HBM, on-chip SRAM for input (i) and output (o) feature maps, and weights (w). Likewise, Carat double buffers all FIFOs and SRAMs to hide on-chip and off-chip access latency. The Carat compute array consists of two parts, i.e., TCs in green and PEs in yellow. TCs generate temporal signals as in Figure 5, using the number sequence from the counter at the top. Then each column of PEs is responsible for vector-scalar multiplication via value reuse on FP8 data, i.e., each column of PEs is an instance of Figure 5 (d) to explore value-level parallelism. Note that each column reuses the partial products (obtained at the top via accumulation) via both pipelining and broadcast to optimally balance efficiency and performance. Multiple PE columns work on the same input vector but different scalar weights, and calculates a vector-vector outer product. On the right-hand side of the array, a column of BF16 adders accumulate the outer products for the GEMM outputs.

4.2 Special Value Handling

We list the details of FP8 format in Table 3. Carat implements three techniques and supports all special values (subnormal, NaN and Zero) at the inputs, weights and outputs. Though this FP8 format does not encode Infinity, the support for NaN and Zero in Carat applies to Infinity in other FP8 formats [48]. First, if the input is subnormal, we perform subnormal adjustments before generating temporal signals. This adjustment facilitates accumulation forwarding for better performance and efficiency, with more details given in Section 4.4. Second, if the input is either NaN or Zero, we leverage product masking to pass the relevant flags to the output. Their impact

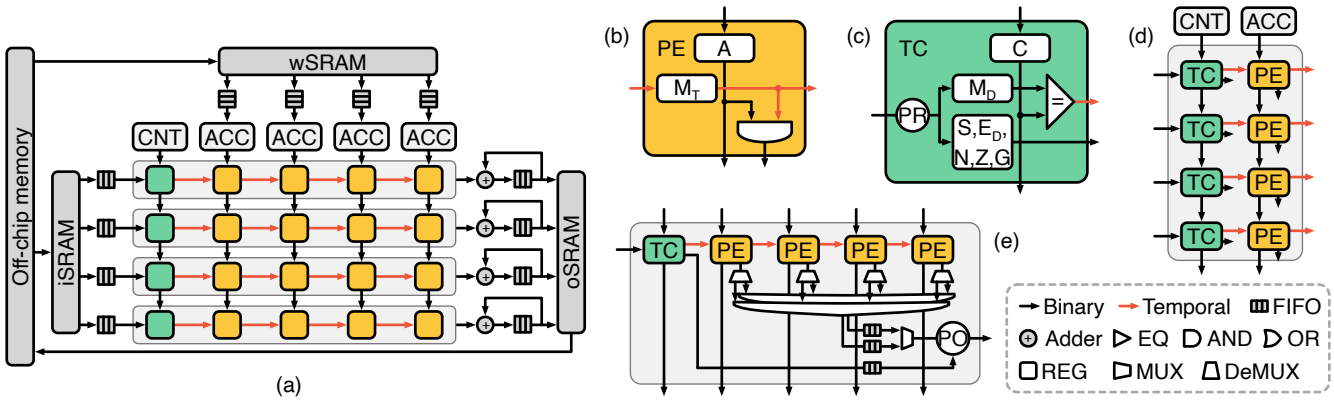


Figure 6. Carat architecture. (a) **Overview**. CNT and ACC denote counter and accumulator. Gray components also exist in systolic arrays (Figure 2). Green and yellow mark temporal converters (TCs) and processing elements (PEs). (b) **PE**. A is the weight accumulation result (partial product) from the row above. M_T is the temporal signal generated from the mantissa M_D in the TC. (c) **TC**. PR means pre-processing FP8 data. S, E, and M stand for sign, exponent and mantissa of the input from the left, respectively; E_D and M_D are the adjusted exponent and mantissa, depending on whether the input is a subnormal number, indicated by a flag D. N and Z are 1-bit flags indicating whether the input is NaN and Zero, respectively. G is the OR gate selection flag for PE outputs. C is the counter number from the row above. All rectangle registers (REG in (b) and (c)) pipeline data to either the right or bottom, either directly or after simple operations, e.g., AND or equality check (EQ). (d) **PE column** for value reuse. (e) **PE row** for output accumulation. PO means post-processing products for correct accumulation.

Table 3. The details of the studied FP8 E4M3 format [48]. The format is given in sign(1-bit).exponent(4-bit).mantissa(3-bit) in binary format with a subscript of 2, where the sign is represented by S. Note that normal numbers have a full mantissa of {1, M}, while subnormal numbers have a full mantissa of {0, M}.

Exponent bias		7
Infinity		N/A
Zero		$S.0000.000_2$
NaN		$S.1111.111_2$
Subnormal	**Max**	$S.0000.111_2 = 0.875 \times 2^{-6}$
	Min	$S.0000.001_2 = 2^{-9}$
Normal	**Max**	$S.1111.110_2 = 1.75 \times 2^8 = 448$
	Min	$S.0001.000_2 = 2^{-6}$

on the final products is reflected before output accumulation, as described in Section 4.4. Third, for the weight, we use exponent expansion to avoid overflow during weight accumulation at the top, while other special values can be handled by the accumulator itself. We elaborate on this in Section 4.5.

4.3 Processing Element

Figure 6 (b) shows a Carat PE, the simplest, yet most important component. Distinguishing from conventional compute-oriented PEs that perform MAC operations, the functionality of Carat PEs is to (1) pipeline the temporal signal rightward and partial products downward, and (2) subscribe to (select) the correct PE output. The spike in temporal signal M_T selects the partial product to the output port; on the other hand, the absence of spikes will set the output to 0.

Throughout all 8 cycles for one temporal signal, the M_T register and AND gate switch twice due to two edges of the spike, and the A register constantly switches. Similar to TCs, the temporal signal again reduces the power and energy consumption by minimizing the switching activity in all PEs.

4.4 Temporal Converter

Figure 6 (c) is a Carat TC. The TC functionality is three-fold, including pipelining the deterministic number sequence, pre-processing the input FP8 data, and generating the temporal signal. The C register pipelines the number sequence, which is sourced from the counter at the top in Figure 6 (a). This organization eliminates the need to generate the sequence at each row, thus saving area. The number sequence will be later used to generate the temporal signal. Then, upon the arrival of an input from the FIFO on the left, the TC identifies (1) whether the input is a special value, and (2) which output OR gate should the PE output go to. Data are processed according to the type of special values.

Subnormal adjustment. If the input is a normal/subnormal number, the exponent (E) and mantissa (M) are adjusted to E_D and M_D via Equation 1. For normal numbers, $\{1, M_D\}$ is set to the full mantissa {1, M}, i.e., no adjustment; for subnormal numbers with a full mantissa of {0, M}, the leading 1 of {0, M} is shifted leftmost, creating $\{1, M_D\}$, where offset $\in [1, 2, 3]$. This adjustment creates a unified data format for both cases,

removing the need for separate subnormal handling.

$$E_D,\ M_D = \begin{cases} E,\ M & \text{, if normal;} \\ E - \text{offset},\ M \ll \text{offset} & \text{, if subnormal.} \end{cases} \quad (1)$$

Accumulation forwarding. Naively, the temporal signal can be generated using all 4 bits of the adjusted full mantissa $\{1, M_D\}$. This requires a total of $2^4 = 16$ cycles; the weight accumulation would start at the 0-th cycle ($0\times$weight) and end at the 15-th cycle ($15\times$ weight). However, due to the leading 1 in $\{1, M_D\}$, the earliest temporal spike would occur at the 8-th cycle (for $M_D = 000_2$) and the latest at the 15-th cycle (for $M_D = 111_2$). This would mean that the partial products from the 0-th to 7-th cycles would never be subscribed to (selected), wasting energy. Instead, our implementation starts the accumulation from the 8-th partial product ($8\times$ weight), so we only require at most 8 rounds of accumulation. Accordingly, we compare the 3-bit M_D with the number sequence to generate the temporal signal and select the correct partial product within 8 cycles. To obtain the 8-th partial product ($8\times$ weight) at the beginning, we simply add 3 to the weight exponent with a small fixed-point adder. In total, 8 such adders are needed, one for each PE column in Figure 6 (a), incurring minimal hardware overhead.

Product masking. When handling NaN or Zero inputs, traditional floating point multipliers directly mask the product to NaN or Zero, and Carat also follows this strategy. TC identifies NaN and Zero and stores N and Z flags. These flags are passed to the post-processing block (PO) as in Figure 6 (e) and used to mask the subscribed product accordingly before it is accumulated into the FIFO.

In addition to handling special values, the TC also uses a gate (G) flag to indicate which ping-pong buffer in Figure 6 (e) this input should use. We describe details in Section 4.6.

During all 8 cycles in generating one temporal signal, all registers remain constant, except that the C register is pipelining accumulated partial products; the equality check logic only produces one spike (logic-1) when the adjusted mantissa M_D equals the counter number C. These two features reduce the power and energy consumption in all TCs by minimizing the switching activity.

4.5 Processing Element Column

Figure 6 (d) shows one column of PEs and TCs, which is an implementation of value reuse for vector-scalar multiplication as in Figure 5 (d). The distinction in between is that: the example circuit of Figure 5 (d) generates the temporal signals of all vector inputs simultaneously, and all inputs see identical accumulated partial products at all cycles via broadcast (Figure 1 (c)); on the contrary, Figure 6 (d) synchronously pipelines the temporal signals and partial products downward. This design choice improves the scalability of Carat by avoiding broadcast. However, due to the simplicity of TCs and PEs, we can locally share the same set of C and A registers among multiple TCs and PEs in a column, e.g., 8 in this work, without impacting the frequency significantly. Then each group of 8 TCs and PEs is still pipelined. The FP8 accumulator at the top accumulates the weights, and pipelines the partial products to all PEs below in the column. The counter at the top generates a deterministic number sequence and pipelines it to all TCs in the rows below. The column of TCs then processes the FP8 inputs that come from the left. First, it temporalizes the mantissa bits to a temporal signal in red using the deterministic number sequence. The temporal input in red subscribes to (selects) the correct partial products at each row. Second, it extracts extra information for output accumulation, marked by the black arrow below the red one. The output product of a PE flows through the port at the bottom right corner of the PE for output accumulation. Based on this single PE column for vector-scalar multiplication, we can further pipeline the temporal inputs to more columns, e.g., appending more PE columns on the right-hand side of Figure 6 (d). Therefore, multiple columns calculates a vector-vector outer product. This multi-column organization hides the latency of the temporal signal. With 8 PE columns, Carat output one set of products per cycle, leading to no throughput loss, but improving the energy due to reduced switching activity.

Exponent expansion. As Carat accumulates FP8 weights for value reuse, when the weight accumulation is too much or the weight value is too large, it is possible that partial products overflow for FP8 data. To eliminate accumulation overflow, FP8 accumulators at the top and the relevant pipeline registers need extended exponent bits to ensure the correctness. More specifically, for FP8 data with a $3 + 1 = 4$-bit full mantissa (plus 1 due to the leading 1 in the adjusted full mantissa), the weight accumulation increases the partial product by up to $2^4 - 1 = 15\times$, thus requiring 4 extra exponent bits in the accumulators. With overflow addressed, Carat yields no accuracy degradation from Table 1. Note that for weights that are NaN or Zero, the accumulator needs to mask the output partial products accordingly.

4.6 Processing Element Row

Figure 6 (e) shows one PE row, which consists of 1 TC, 8 PEs, and 2 output OR gates. It routes out and post-processes the PE outputs for correct output accumulation. The number of PEs matches the cycle count of the temporal signal, to enable fully pipelined execution, i.e., a second input can immediately start the temporalization right after a first temporal signal finishes the generation. As the TC and PEs pipeline the temporal spike to the right, PEs always produce their outputs for one input from left to right in order, i.e., for one input, only one PE among all is outputting a valid output at a time. Therefore, we simply use an OR gate to get the valid output, as all other outputs are 0s (disabled by the temporal spike).

Due to the fully pipelined execution, there could be two spikes that co-exist in a PE row, with an example in Figure 7 (k), where green and red spikes in the first row belong to different inputs. To guarantee correctness, we use two OR gates, with each assigned to one input. The output for an input will go through its assigned OR gate into an assigned ping-pong buffer, indicated by the gate flag G from TC.

Afterwards, we post-process the output of the ping-pong buffer before starting accumulation. The post-processing needs the corresponding information from the TC, including the sign S, adjusted exponent (E_D), NaN (N), Zero (Z) and gate (G) flags. We store the information in a FIFO, with each entry assigned to one input. More specifically, asserted N and Z flags will mask the product to NaN and Zero; signs and exponents of the input and product are XORed and added as the sign and exponent of the final product, respectively.

4.7 Walkthrough Example

Figure 7 gives a cycle-by-cycle walkthrough example to show 1) how the temporal signals subscribe to (select) partial products in (a)-(l) and 2) how to obtain full GEMM results in (m)-(p). In this example, different PE rows do not share C and A registers. As for each input, the sign, adjusted exponent, NaN, Zero and gate flags remain constant; we ignore their logic and focus on the registers with state transitions. Note that in this example, we assume weights and inputs are located in the top and left SRAMs; in real cases, their positions can be switched to maximize the utilization and efficiency.

How does a PE column work? At cycle 0, a new input comes in from the left and updates the adjusted mantissa register M_D to red 7, and the corresponding counter number register C has an initial value of 0. Then at cycle 7, the counter number C increases to 7, equal to M_D, and the TC generates a temporal spike in red. During cycle 1~3, more PE rows take in new inputs, and generate their temporal spikes, whose timing only depends on the input value. For one PE column, the weight accumulation needs 8 cycles, e.g., in the first column, the partial product traverses from 8× the weight at cycle 1 to 15× at cycle 8. When the PE column pipelines the partial products downward, the temporal spike subscribes to (selects) the partial product of each input. For example, red 7 selects 15× at cycle 8, red 1 selects 9× at cycle 4, and red 0s select 8× at cycle 2 and 4. Value reuse allows different inputs to reuse the same partial products independently and in parallel, enabling value-level parallelism.

How does a PE row work? In one PE row, a spike flows from left to right and selects a partial product in each column with identical rounds of accumulation. For example, the red input in the second row has a 0 M_D, with a temporal spike at cycle 1; then at each cycle between cycle 2 and cycle 9, this spike arrives at the next PE on the right-hand side, selects 8× the weight as the output and routes it out through an assigned OR gate. Additionally, we show how Carat simultaneously processes a second set of inputs in green (all zeros for simplicity) between cycle 8 and cycle 15. We take the first row as an example. At cycle 8, the TC accepts a new input and immediately asserts a temporal spike in green. At cycle 9, the first and second PEs both have a spike for a different input; and each PE individually selects its partial product and routes it out to its assigned ping-pong buffer through the assigned OR gate. Afterwards, the ping-pong buffer sends the buffered results to the output accumulator and FIFO to obtain outer products and ultimately GEMM results.

How do we obtain GEMM results? Outer products from different input sets are added element-wise to produce full GEMM results, with an example shown in Figure 7 (m)-(p). By cycle 15, all subscribed products for the first row of the red input set are now in the ping-pong buffer and ready to be accumulated into the corresponding output FIFO. At cycle 16, the first row accumulates and pushes the first product (which was subscribed by the leftmost column) into its corresponding FIFO (top FIFO). This accumulation is done in a circular FIFO fashion with a feedback loop. Then in the following cycle, the next product of the first row is also pushed into this FIFO, while the first product of the second row is pushed into its FIFO. This process continues across all rows. By cycle 23, the top FIFO has fully computed the outer product for the first row of the red input set. At cycle 24, the green set of products begins to accumulate into the FIFOs, starting from the top. In the example, 8I is added to 15A, which is the sum of the leftmost entries of the green and red first-row outer products, respectively. This process continues to accumulate the outer products, eventually producing the GEMM result.

4.8 Multi-Node Carat

The above sections introduce our proposed Carat as a single compute node. Figure 8 shows how Carat can be scaled up to larger multi-note systems using a 2-D mesh Network-on-Chip (NoC) with a shared off-chip memory [24, 60]. The NoC supports both multi-cast and uni-cast traffic to distribute and reuse the inputs and weights across time and space, and to perform reduction among the partial sums generated by each node, thus reducing the off-chip memory accesses. We evenly tile the GEMM computation across all nodes [60].

5 Experimental Setup

5.1 DNN

In this work, we focus on the GEMM operations in DNNs, and use up to 256 batch size, which is within the best prior efforts for DNN inference, 64~512, as described in Section 1 [10, 12, 21, 36, 63]. We evaluate the MLPerf benchmark [47], which contains multiple industrial DNN workloads (both vision and language models) with dedicated datasets, including ResNet50 for image classification on ImageNet dataset [27], UNet for image segmentation on biomedical cell datasets [59],

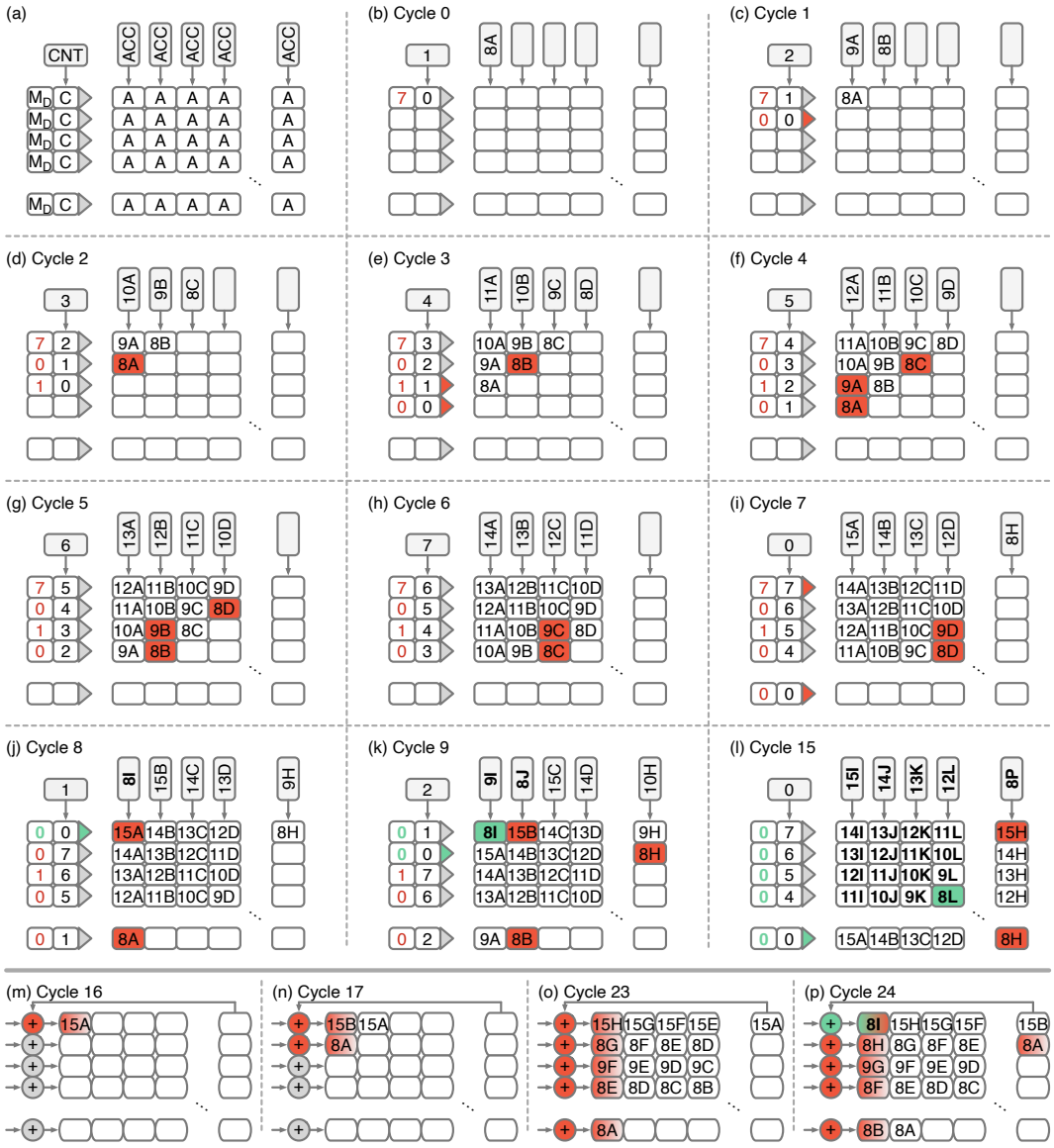


Figure 7. Cycle-level Carat walkthrough example on an 8 × 8 PE array. (a) shows the key components selected from Figure 6. The left and right clusters denote the TC column and PE array. CNT and ACC denote counter and accumulator. The C and M_D are registers for the counter number and adjusted mantissa, that together generate the temporal signal with the equality check logic in triangle. The A denotes the pipeline register for partial products generated by the accumulator at the top. (b)-(l) draw the transition of cycle-level register states in the array, among which (b)-(k) are continuous in cycle. At every cycle, the array takes in a new input and updates the M_D register. There are two sets of inputs marked with red and green, with each set sharing the same partial products. We distinguish two corresponding sets of partial products by whether they are bold or not. Each column works on a different weight, with the two sets of weights marked with A-H and I-P, respectively. For each set, we fill the TC (represented by equality check logic in triangle) and the PE (represented by the A register) with the color of this set, upon the occurrence of a temporal spike. A colored TC means a temporal spike is generated, i.e., the counter number equals the adjusted mantissa; a colored PE mean the current partial product is selected as the output. At every cycle, the TCs and PEs pipeline the counter number and partial products downward, and the temporal signals to the right. The counter number begins with 0 and resets to 0 after 7, while accumulated partial products start from 8× to 15× the weights. We mark the multiples of weights for conciseness. (m)-(p) show the output FIFOs for accumulating outer products element-wise to compute a GEMM result. Subscribed products from each row are accumulated into their position in their corresponding FIFO. Products are accumulated into the FIFO in a circular fashion via a feedback loop. At cycle 24 (p), 8I is added to 15A, which is the sum of the leftmost entries of the green and red first-row outer products, respectively.

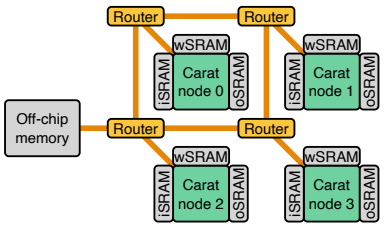


Figure 8. Multi-node Carat architecture. iSRAM, wSRAM and oSRAM denote on-chip input, weight and output SRAMs. The NoC in yellow has a 2-D mesh topology. Each node is a Carat connected to the router, and only one off-chip memory exists at a corner. Each node only works on a subset of all GEMM computation.

SSD for object detection on VOC dataset [43], RNNT for speech-to-text on synthetic voice dataset [28], BERT for natural language processing on GLUE dataset [16], and DLRM for recommendation on Terabyte dataset [52]. Among these DNNs, most layers in ResNet50, UNet and SSD are matrix convolution, thus compute bounded; RNNT, BERT and DLRM contain mostly matrix multiplication, thus memory bounded. We skip the FP8 accuracy evaluation [48], which has been validated in Table 1.

5.2 Hardware

Considering the decreased opportunity of sparse GEMM architectures in language models (Section 2), our evaluation focuses on dense GEMM hardware.

Choice of baselines. We consider three baselines in this work. The first is a conventional binary computing systolic array (bSA) [33], which maximizes data reuse via a weight stationary dataflow (caching weights to be reused by more inputs). The architecture of bSA is given in Figure 2. bSA is to emphasize the comparison between well-studied data reuse and orthogonal value reuse. The second baseline is a GEMM architecture that Reuses computation based on Input Similarity [58], RIS for short. RIS organizes MAC units into a 1-D vector array, with MAC units working on a shared input and different weights. For consecutive inputs in a batch, RIS calculates the full GEMM results for the first input, and only computes the delta of GEMM results based on the difference between consecutive, 4-bit quantized inputs. More similarity between inputs in a batch leads to more savings in computation and memory accesses. The original RIS couples its 128-MAC vector array with 40 MB on-chip memory to minimize off-chip accesses. However, such a large memory-to-compute ratio is no longer preferred in more recent DNN accelerators [33, 60]. To ensure a fair comparison, we configure RIS with an identical memory hierarchy to Carat and bSA, except that no FIFO exists. The reason is that RIS cannot use FIFOs to hide dynamic memory accesses. This baseline is to compare Carat with prior computation reuse schemes. The third baseline is a GEMM architecture based on temporal coding, uSystolic (uSA) [70]. We compare uSA to Carat and show different ways to leverage temporal coding. uSA only works on fixed-point data and spends multiple cycles for each multiplication, while our Carat supports floating-point data and hides the temporal coding latency via pipeling. uSA has a weight stationary systolic array architecture as bSA in Figure 2, but directly interacts with the off-chip memory, i.e., no SRAMs.

Table 4. Comparison of single-node Carat and baselines. The off-chip bandwidth is 128 GB/s. i, w, and o refer to input, weight, and output, respectively, with each having a SRAM size of 128 KB. A configuration *a*-*b* means sweeping all powers of 2 from *a* to *b*.

Configuration	Carat	bSA	RIS	uSA
i/w/o SRAM (KB)	128			–
Array height (H)	32-512	4-16	1	16-64
Array width (W)	8	H	16-256	H
MUL word	FP8			INT8
ACC word	BF16			INT24

Configuration of compute arrays. We summarize the single-node hardware configurations of the Carat and baseline designs in Table 4. The off-chip memory of all designs is HBM with 128 GB/s bandwidth. The on-chip SRAMs and FIFOs are set to deliver sufficient bandwidth and are always double buffered to hide the compute latency, if applicable. We select the square systolic arrays for their best performance and efficiency in baseline designs [33, 70]. We sweep the array shape and ensure that Carat, bSA and RIS designs have either a similar on-chip area (iso-area) or equal floating-point operations per second (iso-FLOPS). In iso-area settings, Carat has half the number of floating-point units in bSA and RIS. Note that this setup is approximately iso-area, as Carat employs such a drastically different architecture that exact iso-area comparison with bSA and RIS is impractical, due to discrete power-of-2 array shape configurations. Nonetheless, we try our best to ensure the fairest comparison. The shapes of uSA are selected to have similar on-chip area to their counterparts, as it exclusively supports fixed-point operations. bSA and RIS share the same word settings as Carat to maintain the same level of accuracy (Table 1), while uSA needs INT8 multiplication and INT24 accumulation [60, 70]. For each multiplication, uSA needs 256 cycles, unlike the 1 cycle in bSA. Though the temporal coding needs 8 cycles in Carat, Carat hides the latency and exhibits only 1-cycle multiplication latency.

Network-on-Chip. For the multi-node Carat, we organize single Carat nodes into a 2-D mesh and connect them with NoC. We assume X-Y routing [30, 60] to avoid deadlock. We vary the NoC shape, e.g., 4×4 and 8×8, to show the performance and efficiency scaling. The multi-node comparison is

Table 5. Comparison of multi-node Carat and systolic array. Systolic array takes the Google edge TPU settings [25]. All designs adopts FP8 multiplication and BF16 accumulation.

Configuration (per node)	Carat Mesh		bSA
	4×4 NoC	8×8 NoC	
i/w/o SRAM (KB)	32	128	2048
Array height (H)	512	128	64
Array width (W)	8		H

configured as in Table 5. For both NoC settings, we ensure that their total FLOPS and total on-chip SRAM size equal to that of the bSA with an array size of 64×64. We keep the memory configuration for this 64×64 bSA as in Goolge Edge TPU, which has a total of 6 MB SRAM for all tensors [25], and we split the storage evenly among three variables, i.e., 2048 KB each. The original RIS adopts a ring-based NoC, which is however not scalable as we will see later, and we exclude RIS in the multi-node evaluation.

5.3 Evaluation Methodology

In this work, we are interested in both the performance and efficiency comparison of different designs.

Performance modeling. We build a cycle-level performance simulator for all evaluated designs. In this simulator, we consider multiple factors to ensure high simulation accuracy. First, we tile all DNN layers, i.e., schedule data, and obtain the compute utilization due to inefficient tiling. The space of data schedule is constrained by the array shape and SRAM size [30]. To obtain an optimized data schedule for each design, we search the space under the constraints. During the search, we maximize the utilization of both the compute array and SRAM, and prioritize the former. Second, we model the SRAM access contention incurred by data schedule. Memory contention happens when the granularity of SRAM accesses is not multiple of the SRAM block size. We model this by aligning the SRAM accesses to the block size to estimate the resource under-utilization. Third, we account for the compute stalls due to insufficient off-chip memory bandwidth. Note that we assume sufficient SRAM bandwidth not to stall compute by default. Fourth, we simulate the impact of NoC. After we tile the GEMM onto NoC, we further estimate the NoC latency. We adopt X-Y routing to avoid deadlock and we use the routing algorithm to derive the worst-case link bandwidth, which is used in latency calculation. Note that RIS designs need to map the output channel dimension of each GEMM to the vector MAC array, so that two consecutive inputs can be compared and the computation is skipped in case of two identical inputs [58]. RIS is originally designed to process continuous video frames, where temporal pixel similarity varies between 50% and 90%. In this work, given the workload diversity, even 50% pixel similarity is impractical. However, we conservatively assume 50% similarity for evaluation. To validate this simulator, we sample multiple GEMM layers from the evaluated MLPerf benchmark, and ensure the simulated results reflect the architecture behavior faithfully on those sampled layers.

Table 6. Switching activity of Carat in cost modeling. 8 cycles lead to 12.5% switching activity for most of the logic except the subscription logic in PE (spike register and AND gate), labeled as "Rest", whose switching activity is doubled to 25% as it responds to both temporal signal edges. All others have full switching activity.

FIFO			PE		TC		OR
i	w	o	C reg	Rest	A reg	Rest	gate
12.5%	12.5%	100%	100%	25%	100%	12.5%	12.5%

Cost modeling. Leveraging the above simulator, we estimate the cost in a pre-silicon event-based manner. This methodology is common in prior works, including but not limited to Aladdin [61], Accelergy [74], MAESTRO [40], and one of our baselines, RIS [58]. First, we obtain the number of different events through the performance simulator, including (1) on-chip memory/SRAM/FIFO accesses, (2) the number of multiplications and accumulations, and (3) the number of NoC transfers. Then, we retrieve the cost of different events from cost modeling tools. For SRAM, we use CACTI7 [6] to obtain the area, leakage power, and access energy. We extrapolate HBM access energy numbers from [55]. We synthesize individual compute units with Synopsys Design Compiler with SAED 32 nm technology at 400 MHz, and aggregate them together. For 2-D mesh NoC, we follow the design in [77] and assume a static network using crossbar switches with 128 GB/s. We extrapolate the router area and power in our evaluation. Lastly, we aggregate the final cost by multiplying the single event cost with the event count. We count for the switching activity due to temporal coding, e.g., the 8-cycle temporal signal in Carat reduces the switching activity of the relevant logic. We list the impacted switching activity in Table 6. To validate our cost model, we also place-and-route (P&R) the core component of Carat, i.e., the TC and PE array, as shown in Table 7. Notably, the area error rate resides within 2.4%. P&R results show that Carat can run in 370~450 MHz clock frequency; conversely, bSA arrays can only go up to approximately 170 MHz after P&R. In our evaluation, we configure all designs with an identical frequency of 400 MHz. Note that this frequency increases the bSA performance more than twice according to P&R results. Moreover, we place-and-route Carat with a height of 64 and vary the sharing factor (Section 4.5) from 8 to 64. We find that the frequency fluctuates within 440~470 MHz, implying that PEs are off the critical path.

Table 7. Comparison of synthesis and place-and-route results for the TC and PE array, with a default sharing factor of 8 (Section 4.5).

Carat height	Synthesis / P&R area error rate	Frequency (MHz)	
		Synthesis	P&R
32	2.4%	885.0	446.4
64	0.8%	854.7	444.4
128	1.5%	408.2	440.5
256	0.9%	404.9	395.3
512	-0.7%	401.6	373.1

Table 8. Comparison of Carat and baselines, in terms of throughput, on-chip (OC) area and on-chip/full-system (FS) energy and power efficiency. Single-node (SN) and multi-node ($n \times n$ NoC) settings are from Table 4 and Table 5. For each design, the number in the "()" is the array height. The results are for a batch size of 256. Carat (64) and (128) are iso-area and iso-FLOPS comparisons, respectively.

Design		Throu. (Gflop/s)	OC area. (mm^2)	Energy eff. (Gflop/s/J)		Power eff. (Gflop/s/W)	
				OC	FS	OC	FS
SN	Carat (64)	49.5	1.8	47.6	21.2	372.0	167.9
	Carat (128)	98.1	2.2	139.2	67.3	571.0	276.1
	bSA (8)	48.5	1.7	44.8	20.2	372.0	167.9
	RIS (64)	72.8	1.7	30.8	21.9	170.3	121.2
	uSA (32)	2.9	1.5	0.2	0.1	22.4	11.0
SN	bSA (64)	1973.8	39.1	3035.6	2537.5	618.6	517.1
4×4	Carat (512)	4971.7	79.5	8895.7	7914.7	719.7	640.3
8×8	Carat (128)	6310.2	34.4	12900.4	11468.6	822.3	731.0

6 Evaluation

This section evaluates Carat's performance (e.g., throughput and utilization) and cost (e.g., area, energy and power efficiency) for both a single node and multiple nodes.[2] Table 8 summarizes the evaluation results in this work.

6.1 Single-Node Carat

In general, we find that single-node Carat can perform comparably and in some cases outperform other implementations in throughput and efficiency with marginal overheads.

Why does Carat outperform? For iso-area comparison, Carat (64) with half FLOPS reaches slightly better throughput and efficiency than bSA, as value reuse transforms multiplication into accumulation, reducing mandatory computations. Note that iso-area Carat has lower throughput than RIS, as RIS skips 50% of computations. However, RIS requires more memory accesses, i.e., four memory accesses per input, lowering the efficiency. If higher throughput is favored at the cost of area overhead, we can opt for an iso-FLOPS Carat (128) implementation, which would have a similar throughout-area trade-off compared to increasing the FLOPS in bSA. Though uSA adopts hardware-friendly fixed-point data, Carat has much higher performance and efficiency by reducing multiplication cycles from 256 to 1.

Takeaway. Value-level parallelism at a single node offers comparable and in some cases better throughput and efficiency compared to prior GEMM optimizations, e.g., data reuse, computation reuse and temporal coding.

6.2 Single-Node Sensitivity Study

This section shows single-node sensitivity studies on the batch size and array shape to better understand Carat.

6.2.1 Batch Size.

Throughput guarantee. For most models and batch sizes in Figure 9 (a), iso-FLOPS Carat has a throughput that is no worse than those of bSA and uSA, ensuring high throughput for general use cases. In small batch sizes, Carat can be on par with bSA for vision models, whose GEMM operations are mostly matrix convolution, where value reuse opportunities are already abundant. However, language models, dominated by matrix multiplication, require large batch size for higher throughput. When the batch size exceeds the height of Carat array, e.g., 128 and 256 here, Carat outperforms in throughput consistently across all models. Iso-area Carat almost halves the throughput of iso-FLOPS Carat at large batch sizes, but still maintains slightly better throughput than bSA.

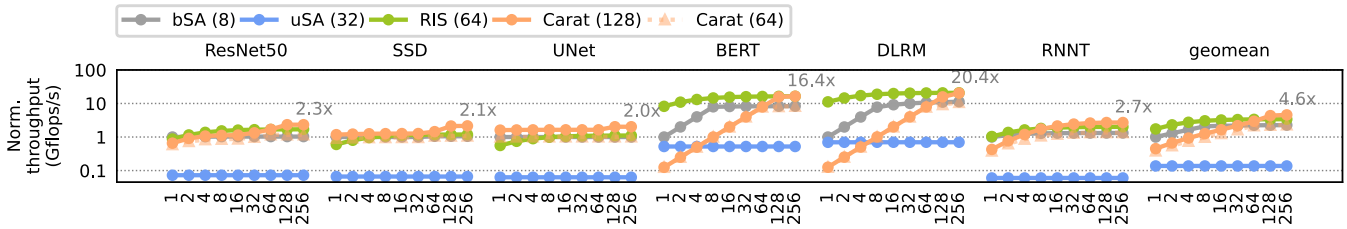
Large batch benefits utilization. In Figure 9 (b), we observe that Carat gradually increases its utilization as the batch size grows, ultimately reaching or exceeding that of bSA. The reason is that larger batch sizes expose more value reuse opportunities, which can be exploited in Carat. One example is that when the batch size is 128, we group all 128 inputs in a batch into one input vector for value reuse, and iso-FLOPS Carat with 128 rows will always have 100% utilization; iso-area Carat saturates even earlier at 64 batch size. bSA also exhibits an ascending trend in utilization in language models upon larger batch sizes, due to more data reuse opportunities. However, bSA saturates at a batch size of 8, as its array shape is 8×8. As a result, Carat benefits much more from large batch. uSA also exhibits high utilization, but does not translate to high throughput due to the long multiplication latency.

Takeaway. Carat's performance benefits stem from value reuse, and larger batch sizes expose richer opportunities for value-level parallelism.

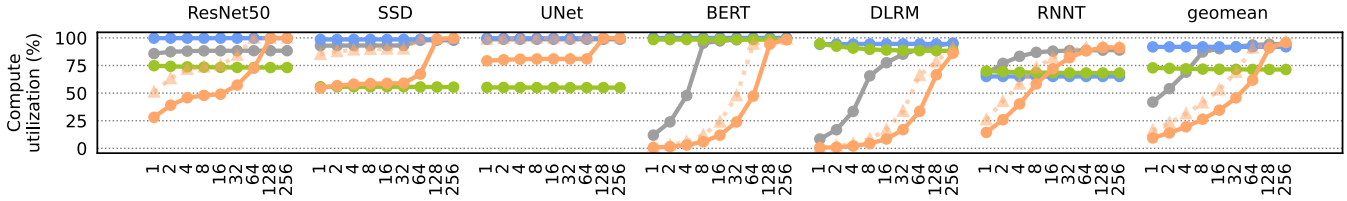
6.2.2 Array Shape.

Shape scaling. In Figure 10 (a), the throughput of all designs scales up with the array height, and Carat shows the best throughput among all. However, the utilization degrades for all when the array becomes larger. In Figure 10 (b), we observe a large utilization plunge for Carat, when the array height 512 is larger than the batch size 256. Before this

[2]We follow the convention in [70] and define throughput over energy and power as energy and power efficiency, where energy efficiency is a variant of energy-delay product.



(a) Throughput in log scale. All points and numbers are normalized to bSA throughput at a batch size of 1 for each model.



(b) Utilization. Attainable throughput over peak throughput.

Figure 9. Batch size study per model for MLPerf benchmark. Numbers in "()" denote the array height, as in Table 4. The x axis is the batch size, we vary the batch size from 1 to 256. ResNet50, SSD and UNet are vision models with matrix convolution; BERT, DLRM, RNNT are language models with matrix multiplication.

Table 9. Trade-off in iso-area and iso-FLOPS Carat. Δ is the ratio of changes in Carat over others. OC is short for on-chip. For area and throughput (thro.), higher is better; for energy (ener.), lower is better.

Baseline	Iso-area				Iso-FLOPS			
	AreaΔ		Thro.Δ	Ener.Δ	AreaΔ		Thro.Δ	Ener.Δ
	Array	OC			Array	OC		
bSA (8)	18.5%	4.2%	2.2%	-3.7%	137.9%	30.6%	102.2%	-34.9%
bSA (16)	28.7%	17.6%	5.0%	14.5%	189.1%	94.0%	65.4%	4.4%
RIS (64)	46.3%	6.6%	-31.9%	-55.9%	193.8%	33.6%	34.7%	-70.2%
RIS (256)	56.9%	12.3%	111.3%	-80.3%	252.5%	85.2%	233.1%	-82.0%

point, the utilization drop is almost linear, as in uSA and bSA. However, RIS exhibits even worse scalability, i.e., the utilization drops earlier at an array shape of 128. The reason is that RIS maps the output channel to a vector array, and the fixed output channel does not increase with the batch size, leaving MAC units underutilized. In Figure 10 (c), we observe that due to different architectures, Carat area can not exactly match that of bSA and RIS, which are both built on conventional MAC units. In Figure 10 (d), the energy advantage of Carat over bSA vanishes at a large shape, e.g., 512 here. Though RIS requires less area, its excessive memory accesses blow up the energy. uSA always consumes the most energy due to the long latency. We further show iso-area and iso-FLOPS comparisons in Figure 11. In both settings, Carat effectively trades off area for performance, efficiency, or both. We show such trade-offs in Table 9. In general, both the area overhead and gain in iso-FLOPS Carat are higher. Note that allowing a similar overhead in bSA (doubling the PE count and having 4× FLOPS that of iso-area Carat) will show similar gains.

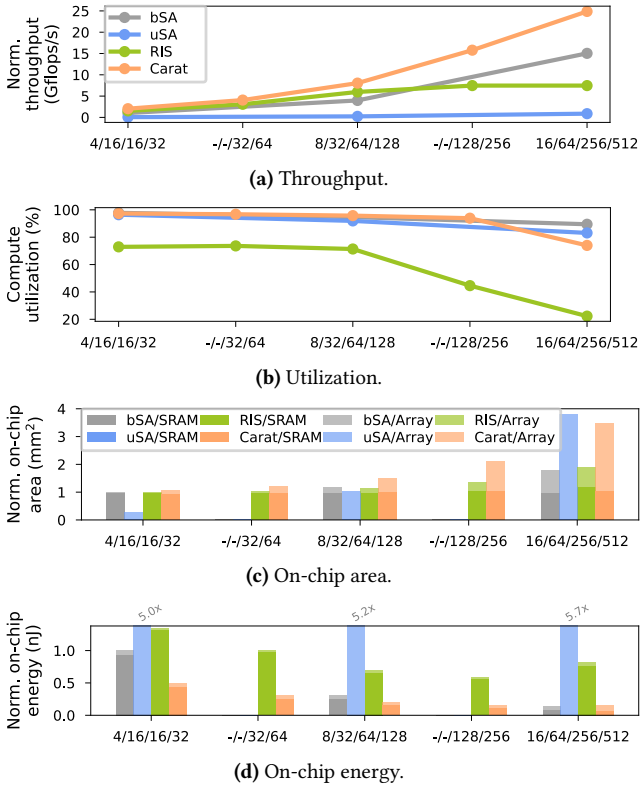
Breakdown comparison. In Figure 12, the area and energy breakdown of the compute arrays is shown. For bSA, RIS and uSA, PEs dominate the hardware, which contain costly MAC units. On the contrary, in both iso-area (64) and iso-FLOPS (128) Carat, PEs only acount for a small fraction (7%) of the total area and energy, as Carat PEs are responsible for subscription, instead of computing. The Carat accumulators occupy half of the total area, and the remaining components are auxiliary logic to implement value-level parallelism. Then in Figure 13, we show the PE-level breakdown. We observe that conventional MAC units in bSA and RIS dominate the PE, while uSA and Carat PEs with temporal computing designs have more diverse compositions.

Takeaway. Carat's throughput and efficiency advantages diminish upon larger shapes. Thus one should be mindful of the proper shape to benefit most from value-level parallelism.

6.3 Multi-Node Carat

Via value reuse, single-node Carat can perform comparably or outperform baselines in terms of performance and efficiency. Carat can achieve further performance and efficiency improvements with orthogonal data reuse via NoC. Figure 14 summarizes the comparison between bSA and multi-node Carat, with two NoC configurations given in Table 5.

This evaluation is meant to show how to scale Carat to larger systems, e.g., in the cloud. We keep bSA as a single-node implementation and scale it up, compared to a multi-node (scaled-out) Carat. Note that though bSA can also be scaled out in this way, the results of multi-node bSA vs. multi-node Carat would be very similar to the single-node comparison between them. Here we omit the comparison against RIS and uSA designs. RIS is skipped due to its severe



(a) Throughput.

(b) Utilization.

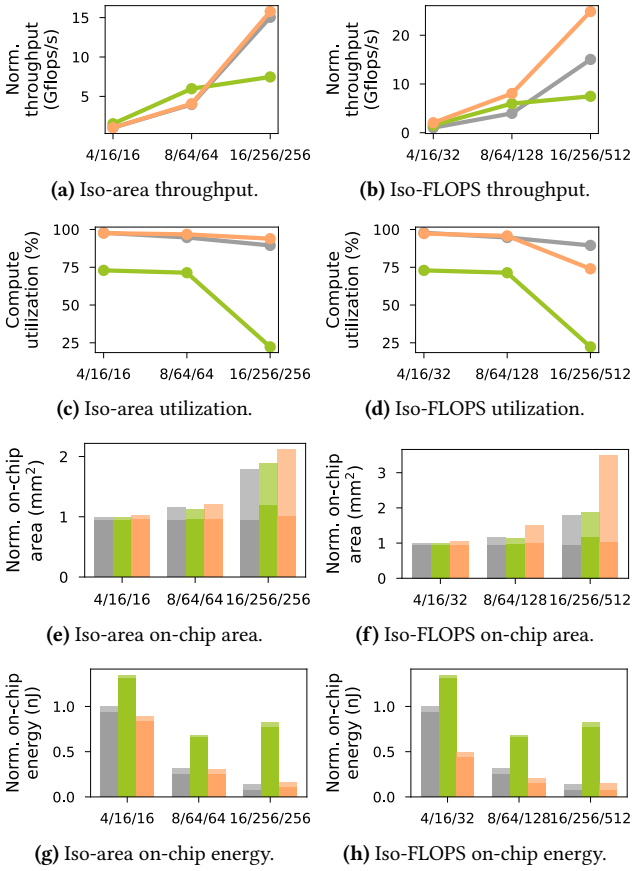
(c) On-chip area.

(d) On-chip energy.

Figure 10. Array shape study for MLPerf benchmark. The x axis ticks, $a/b/c/d$, are the array height values, with a for bSA, b for uSA, c for RIS and d for Carat; "-" means no configuration for this tick, as we only allow square bSA and uSA. All data points are normalized to the bSA results with an array height of 4. The batch size is set to 256. In (c) and (d), area and energy are broken down into SRAM and array. The latter contains everything on-chip but SRAM.

underutilization at large batch sizes in Figure 10 (a) and (b), which is further exacerbated when scaled out with a ring-based NoC [58]. As RIS splits all output channels across its vector MAC PEs, ring-based RIS will have a vector size up to 4096, which is far beyond the number of output channels in most GEMM layers, leading to heavy underutilization. uSA is also omitted due to its low throughput in our setup and its lack of potential to be adopted in higher throughput settings.

Throughput improvement. Figure 14 (a) shows the throughput improvement of multi-node Carat over bSA. With an array height of 64, the underutilization of bSA is more severe than that in Figure 10 (b). By scaling up to more nodes, Carat speeds up the computation, thus improving throughput. However, this scaling is not linear. Maintaining the same total number of floating-point units, 4×4 and 8×8 NoCs have a Carat node of height 512 and 128, respectively. However, an array height of 512 makes Carat more susceptible to compute underutilization, as shown in Figure 10 (b).



(a) Iso-area throughput.

(b) Iso-FLOPS throughput.

(c) Iso-area utilization.

(d) Iso-FLOPS utilization.

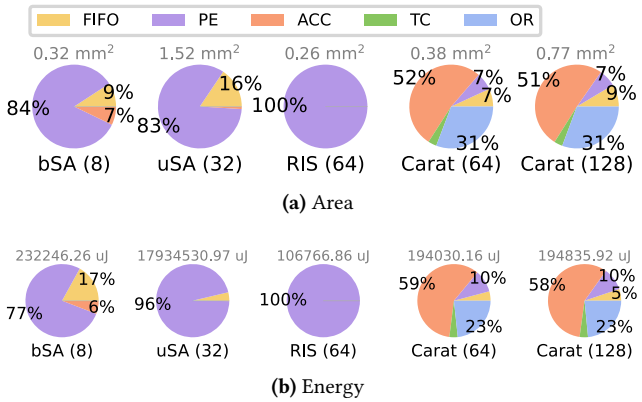
(e) Iso-area on-chip area.

(f) Iso-FLOPS on-chip area.

(g) Iso-area on-chip energy.

(h) Iso-FLOPS on-chip energy.

Figure 11. Iso-area (left) and iso-FLOPS (right) comparison. This figure follows the legends in Figure 10, except that uSA is omitted.

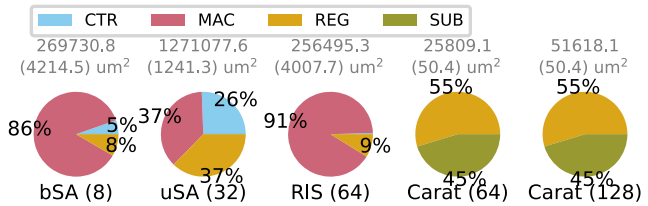


(a) Area

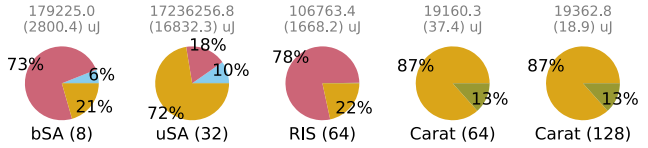
(b) Energy

Figure 12. Array-level breakdown. "()" labels the array height.

Efficiency enhancement. Figure 14 (b) shows off-chip memory access count for different designs. With NoC to support more flexible data schedule (e.g., multi-cast, uni-cast, and reduction), multi-node Carat is able to reduce the off-chip memory accesses almost by half, improving the system

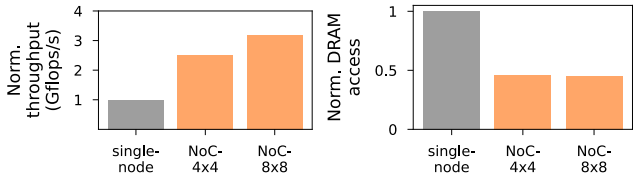


(a) Area.

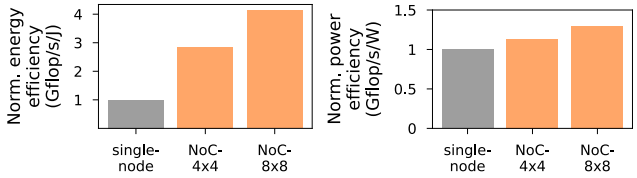


(b) Energy.

Figure 13. PE-level breakdown. "( )" labels the array height. The numbers at the top of each pie are for total and individual PEs. CTR, REG and SUB refer to control, register and subscription logic.



(a) Throughput. **(b)** Off-chip memory access.



(c) On-chip energy efficiency. **(d)** On-chip power efficiency.

Figure 14. Multi-node Carat study for MLPerf benchmark. Hardware configurations are listed in Table 5. Results are normalized to that of single-node bSA. The batch size is 256.

efficiency as in Table 8. Furthermore, in Figure 14 (c) and (d), we observe an increase in both energy and power efficiency. The power efficiency improvement is marginal, and the energy improvement almost follows that of throughput.

Takeaway. Leveraging orthogonal data reuse on top of value reuse, multi-node Carat can simultaneously improve the throughput and efficiency. Designers shall choose a proper single-node Carat shape to obtain the best gain from both data reuse and value-level parallelism.

7 Discussion

7.1 Limitations

This work focuses on the demonstration of value-level parallelism on dense GEMMs. The proposed Carat architecture consider the acceleration opportunities from data sparsity to be limited, especially in the recently popular language models, as shown in Figure 3 (a) and (b). However, a lot of recent works on DNN acceleration explore the possibility of deliberate structured sparsity, i.e., manually introducing sparse data at a target granularity. Structured sparsity acceleration in Carat is feasible but not explored in this work.

7.2 Related Work

Computation reuse. Computation reuse has been well studied in both non-DNN and DNN workloads. Fuchs et al. accelerated datacenter workloads by introducing one additional NVM layer to store the computed results [19]. Carat, however, is a plug-and-play GEMM architecture, without sophisticated system changes. In the context of DNNs, Riera et al. reused computation based on the input similarity (RIS), but at the cost of accuracy drop and poor scalability [58]. Kartik et al. profiled the weight distribution to identify the cases where one input is multiplying with different weights of the same value, and reuse the product in such cases [29]. Different from these works, value reuse is input agnostic and works for arbitrary data distribution.

Bit-level computing. Temporal coding is a bit-level computing paradigm. Other bit-level computing to accelerate GEMMs includes bit-serial, unary, and neuromorphic computing [14, 44, 49, 68–70]. While these works feed on fixed-point data, Carat allows temporal coding to work on floating-point data with special value handling.

8 Conclusion

In this work, we identify a waste of compute in DNN inference on batched, low-precision data, due to calculating the same products repeatedly. To address this problem, we propose a novel computing paradigm, value-level parallelism, via value reuse and temporal coding. Value reuse computes unique products only once and different inputs subscribe to their products via temporal coding. With this, we present a multiplier-free Carat architecture with FP8 data to leverage value-level parallelism. Our experiments with large batch DNN inference show that Carat achieves comparable or better performance and efficiency over well-designed baselines.

9 Acknowledgements

We thank all reviewers for their valuable feedback. Also, special thanks to Mohammad Sazadur Rahman, Niranjan Shetty and our shepherd for their generous support when revising the paper, and to all anonymous animals who contributed to the idea development and Figure 1. This work is supported by the Wisconsin Alumni Research Foundation, University of Central Florida and NSF under award No. CNS-2045985.

A Artifact Appendix

A.1 Abstract

The scope of artifact evaluation covers the major results in Section 6, i.e., Figure 9, Figure 10, Figure 11, Figure 12, and Figure 14. The provided artifact can be run on a x86_64 machine with docker installation. We have tested the workflow on ubuntu 20.04. To run the artifact, extract the zip file and follow the instruction in `README.md` or Section A.4 and A.5. To see the results generated from running the artifact, see Section A.6 for detail.

A.2 Artifact check-list (meta-information)

- **Model:** Cycle-level performance model, event-based cost model
- **Data set:** MLPerf suite (included in the artifact)
- **Run-time environment:** Docker
- **Hardware:** x86_64 machine
- **Metrics:** Throughput, utilization, area, energy/power efficiency
- **Output:** Figure 9, Figure 10, Figure 11, Figure 12, Figure 14
- **Experiments:** Batch size, array shape, iso-x comparison, area breakdown, multi-node scaling studies
- **How much disk space required (approximately)?:** 1.5GB
- **How much time is needed to prepare workflow (approximately)?:** 1 hour
- **How much time is needed to complete experiments (approximately)?:** 1-4 hours
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT License
- **Workflow framework used?:** in-house simulation framework
- **Archived (provide DOI)?:** 10.5281/zenodo.10553038

A.3 Description

A.3.1 How to access. First, obtain the zip file from https://zenodo.org/records/10553038. Then, extract the zip file and follow the instruction in `README.md` or Section A.4 and A.5.

A.3.2 Hardware dependencies. A x86_64 machine is required for building the docker image to run experiments.

A.3.3 Software dependencies. Docker [3] is required to build the image and run the container.

A.3.4 Data sets. Our evaluation is on the MLPerf benchmark suite. The user does not need to access the benchmark as the extracted layer shapes are already included in our simulator framework.

A.3.5 Models. The models used in our simulation framework include a cycle-level performance model and an event-based cost model.

[3] Available at https://docs.docker.com/engine/install/ubuntu/.

A.4 Installation

After the zip file is extracted and docker is installed, one may follow the following steps, also available in `README.md`, to run the artifact.

1. Build the docker image. The build process installs all the software dependencies and automatically runs all the experiments. Note that depending on the machine configuration, this step may take a few hours. For 12th Gen Intel i9-12900 3.4GHz CPU with 64GB RAM, it takes one hour to build the image.
   `docker build -t carat .`
2. Create a container called `asplos24` from the built image by
   `docker create --name asplos24 carat`
3. Retrieve all the generated figures from the container by
   `docker cp asplos24:/artifact/result_plot .`
4. Finally, to terminate the docker container, run
   `docker stop asplos24`

A.5 Experiment workflow

We use scripts to automatically run the workflow for results production. To produce a figure, the evaluation scripts first generate all the hardware configurations based on a set of provided templates. Then, the simulation is run using the specified hardware configuration and benchmark. More specifically, both the performance model and cost model are run. Note that the automated workflow launches these simulations in parallel. Finally, the scripts parse the generated result log and aggregate results across runs from all the configurations to generate figures.

A.6 Evaluation and expected results

After running the steps in Section A.4, the generated figures can be found locally in `result_plot` directory. Each figure is labeled as `figX-Y.pdf` that corresponds to what is included in Section 6.

A.7 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

References

[1] Shakeel Ahmad, Muhammad Zubair Asghar, Fahad Mazaed Alotaibi, and Yasser D Al-Otaibi. A Hybrid CNN+BILSTM Deep Learning-Based DSS for Efficient Prediction of Judicial Case Decisions. *Expert Systems with Applications*, 209:118318, 2022.

[2] Ryo Akita, Akira Yoshihara, Takashi Matsubara, and Kuniaki Uehara. Deep Learning for Stock Prediction Using Numerical and Textual Information. In *International Conference on Computer and Information Science*, 2016.

[3] Arm. Arm supports FP8: A new 8-bit floating-point interchange format for Neural Network processing. Online, Sep 2022.

[4] Mir Mohammad Azad, Apoorva Ganapathy, Siddhartha Vadlamudi, and Harish Paruchuri. Medical Diagnosis Using Deep Learning Techniques: A Research Survey. *Annals of the Romanian Society for Cell Biology*, 25(6):5591–5600, 2021.

[5] Mihalj Bakator and Dragica Radosav. Deep Learning and Medical Diagnosis: A Review of Literature. *Multimodal Technologies and Interaction*, 2(3):47, 2018.

[6] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *Transactions on Architecture and Code Optimization*, 14(2), Jun 2017.

[7] Chase. How Often is Your Credit Score Updated? Online, Sep 2023.

[8] Baogui Chen, Yu Li, Shu Zhang, Hao Lian, and Tieke He. A Deep Learning Method for Judicial Decision Support. In *International Conference on Software Quality, Reliability and Security Companion*, pages 145–149, 2019.

[9] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *International Symposium on Computer Architecture*, 2016.

[10] Yujeong Choi, Yunseong Kim, and Minsoo Rhu. Lazy Batching: An SLA-aware Batching System for Cloud Machine Learning Inference. In *International Symposium on High-Performance Computer Architecture*, 2021.

[11] Iulia M. Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function. In *International Conference on Acoustics, Speech and Signal Processing*, 2020.

[12] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *Symposium on Networked Systems Design and Implementation*, 2017.

[13] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large Scale Distributed Deep Networks. In *International Conference on Neural Information Processing Systems*, 2012.

[14] Alberto Delmas Lascorz, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, Kevin Siu, and Andreas Moshovos. Bit-Tactical: A Software/Hardware Approach to Exploiting Value and Bit Sparsity in Neural Networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina" Toutanova. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[17] S. Rasoul Faraji and Kia Bazargan. Hybrid Binary-Unary Hardware Accelerator. *Transactions on Computers*, 69(9):1308–1319, 2020.

[18] Andrea Fasoli, Chia-Yu Chen, Mauricio Serrano, Swagath Venkataramani, George Saon, Xiaodong Cui, Brian Kingsbury, and Kailash Gopalakrishnan. Accelerating Inference and Language Model Fusion of Recurrent Neural Network Transducers via End-to-End 4-bit Quantization. *arXiv*, 2022.

[19] Adi Fuchs and David Wentzlaff. Scaling Datacenter Accelerators with Compute-Reuse Architectures. In *International Symposium on Computer Architecture*, 2018.

[20] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[21] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. Low Latency RNN Inference with Cellular Batching. In *EuroSys Conference*, 2018.

[22] Jiong Gong, Haihao Shen, Guoming Zhang, Xiaoli Liu, Shane Li, Ge Jin, Niharika Maheshwari, Evarist Fomenko, and Eden Segal. Highly Efficient 8-Bit Low Precision Inference of Convolutional Neural Networks with IntelCaffe. In *Reproducible Quality-Efficient Systems Tournament on Co-Designing Pareto-Efficient Deep Learning*, 2018.

[23] Patricia Gonzalez-Guerrero, Meriam Gay Bautista, Darren Lyles, and George Michelogiannakis. Temporal and SFQ Pulse-Streams Encoding for Area-Efficient Superconducting Accelerators. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

[24] Google. System Architecture. Online, Nov 2022.

[25] Google. Edge TPU Compiler. Online, Apr 2023.

[26] Björn Rafn Gunnarsson, Seppe Vanden Broucke, Bart Baesens, María Óskarsdóttir, and Wilfried Lemahieu. Deep Learning for Credit Scoring: Do or Don't? *European Journal of Operational Research*, 295(1):292–305, 2021.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[28] Yanzhang He, Tara N. Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, Qiao Liang, Deepti Bhatia, Yuan Shangguan, Bo Li, Golan Pundak, Khe Chai Sim, Tom Bagby, Shuo-yiin Chang, Kanishka Rao, and Alexander Gruenstein. Streaming End-to-end Speech Recognition for Mobile Devices. In *International Conference on Acoustics, Speech and Signal Processing*, 2019.

[29] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W. Fletcher. UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition. In *International Symposium on Computer Architecture*, 2018.

[30] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. CoSA: Scheduling by Constrained Optimization for Spatial Accelerators. In *International Symposium on Computer Architecture*, 2021.

[31] Intel. Neural Network Distiller. Online, Oct 2019.

[32] Intel. Cross-Industry Hardware Specification to Accelerate AI Software Development. Online, Seq 2022.

[33] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of A Tensor Processing Unit. In *International Symposium on Computer Architecture*, 2017.

[34] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj

Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A Study of BFLOAT16 For Deep Learning Training. *arXiv*, 2019.

[35] Yunseong Kim, Yujeong Choi, and Minsoo Rhu. PARIS and ELSA: An Elastic Scheduling Algorithm for Reconfigurable Multi-GPU Inference Servers. *arXiv*, 2022.

[36] Jack Kosaian, Amar Phanishayee, Matthai Philipose, Debadeepta Dey, and Rashmi Vinayak. Boosting The Throughput and Accelerator Utilization of Specialized CNN Inference Beyond Increasing Batch Size. In *International Conference on Machine Learning*, 2021.

[37] Kankawin Kowsrihawat, Peerapon Vateekul, and Prachya Boonkwan. Predicting Judicial Decisions of Criminal Cases from Thai Supreme Court Using Bi-directional GRU with Attention Mechanism. In *Asian Conference on Defense Technology*, pages 50–55, 2018.

[38] Eli Kravchik, Fan Yang, Pavel Kisilev, and Yoni Choukroun. Low-bit Quantization of Neural Networks for Efficient Inference. In *International Conference on Computer Vision Workshops*, 2019.

[39] Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. FP8 Quantization: The Power of the Exponent. In *Advances in Neural Information Processing Systems*, 2022.

[40] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *International Symposium on Microarchitecture*, 2019.

[41] Raymond S. T. Lee. Chaotic Type-2 Transient-Fuzzy Deep Neuro-Oscillatory Network (CT2TFDNN) for Worldwide Financial Prediction. *Transactions on Fuzzy Systems*, 28(4):731–745, 2020.

[42] Sunwoo Lee, Qiao Kang, Sandeep Madireddy, Prasanna Balaprakash, Ankit Agrawal, Alok Choudhary, Richard Archibald, and Wei-keng Liao. Improving Scalability of Parallel CNN Training by Adjusting Mini-Batch Size at Run-Time. In *International Conference on Big Data*, 2019.

[43] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot Multibox Detector. In *European Conference on Computer Vision*, 2016.

[44] Hang Lu, Liang Chang, Chenglong Li, Zixuan Zhu, Shengjian Lu, Yanhuan Liu, and Mingzhe Zhang. Distilling Bit-Level Sparsity Parallelism for General Purpose Deep Learning Acceleration. In *International Symposium on Microarchitecture*, 2021.

[45] Siyuan Ma and Mikhail Belkin. Kernel Machines That Adapt To GPUs for Effective Large Batch Training. In *Machine Learning and Systems*, 2019.

[46] A. Madhavan, T. Sherwood, and D. Strukov. Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms. In *International Symposium on Computer Architecture*, 2014.

[47] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. MLPerf Training Benchmark. In *Machine Learning and Systems*, 2020.

[48] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. FP8 Formats for Deep Learning. *arXiv*, 2022.

[49] Harideep Nair, John Paul Shen, and James E Smith. A Microarchitecture Implementation Framework for Online Learning with Temporal Neural Networks. In *Computer Society Annual Symposium on VLSI*, 2021.

[50] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *International Conference on International Conference on Machine Learning*, 2010.

[51] M. Hassan Najafi, David. J. Lilja, Marc D. Riedel, and Kia Bazargan. Low-Cost Sorting Network Circuits Using Unary Processing. *Transactions on Very Large Scale Integration Systems*, 26(8):1471–1480, 2018.

[52] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *arXiv*, 2019.

[53] Badreddine Noune, Philip Jones, Daniel Justus, Dominic Masters, and Carlo Luschi. 8-bit Numerical Formats for Deep Neural Networks. *arXiv*, 2022.

[54] NVIDIA. NVIDIA, Arm, and Intel Publish FP8 Specification for Standardization as an Interchange Format for AI. Online, Sep 2022.

[55] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W. Keckler, and William J. Dally. Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems. In *International Symposium on Microarchitecture*, 2017.

[56] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. MegDet: A Large Mini-Batch Object Detector. In *Conference on Computer Vision and Pattern Recognition*, 2018.

[57] Becky Pokora. Credit Card Statistics And Trends 2023. Online, Mar 2023.

[58] Marc Riera, Jose-Maria Arnau, and Antonio Gonzalez. Computation Reuse in DNNs by Exploiting Input Similarity. In *International Symposium on Computer Architecture*, 2018.

[59] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015.

[60] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *International Symposium on Microarchitecture*, 2019.

[61] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *International Symposium on Computer Architecture*, 2014.

[62] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep Learning in Medical Image Analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[63] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Symposium on Operating Systems Principles*, 2019.

[64] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. FlexGen: High-Throughput Generative Inference of Large Language Models with A Single GPU. *International Conference on Machine Learning*, 2023.

[65] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-Bit Floating Point (HFP8) Training and Inference for Deep Neural Networks. *Advances in neural information processing systems*, 32, 2019.

[66] Georgios Tzimpragos, Advait Madhavan, Dilip Vasudevan, Dmitri Strukov, and Timothy Sherwood. Boosted Race Trees for Low Energy Classification. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[67] Georgios Tzimpragos, Jennifer Volk, Alex Wynn, James E. Smith, and Timothy Sherwood. Superconducting Computing with Alternating Logic Elements. In *International Symposium on Computer Architecture*,

2021.

[68] Di Wu, Jingjie Li, Zhewen Pan, Younghyun Kim, and Joshua San Miguel. uBrain: A Unary Brain Computer Interface. In *International Symposium on Computer Architecture*, 2022.

[69] Di Wu, Jingjie Li, Ruokai Yin, Hsuan Hsiao, Younghyun Kim, and Joshua San Miguel. uGEMM: Unary Computing Architecture for GEMM Applications. In *International Symposium on Computer Architecture*, 2020.

[70] Di Wu and Joshua San Miguel. uSystolic: Byte-Crawling Unary Systolic Array. In *International Symposium on High-Performance Computer Architecture*, 2022.

[71] Di Wu and Joshua San Miguel. Special Session: When Dataflows Converge: Reconfigurable and Approximate Computing for Emerging Neural Networks. In *International Conference on Computer Design*, 2021.

[72] Hao Wu. Low precision Inference on GPU. Online, Mar 2019.

[73] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. *arXiv*, 2020.

[74] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *International Conference on Computer-Aided Design*, 2019.

[75] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, and Kurt Keutzer. HAWQ-V3: Dyadic Neural Network Quantization. In *International Conference on Machine Learning*, 2021.

[76] Sungyeob Yoo, Hyunsung Kim, Jinseok Kim, Sunghyun Park, Joo-Young Kim, and Jinwook Oh. LightTrader: A Standalone High-Frequency Trading System with Deep Learning Inference Accelerators and Proactive Scheduler. In *International Symposium on High-Performance Computer Architecture*, 2023.

[77] Yaqi Zhang, Alexander Rucker, Matthew Vilim, Raghu Prabhakar, William Hwang, and Kunle Olukotun. Scalable interconnects for reconfigurable spatial architectures. In *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.