DP4C: A SoC Architecture for NN-driven Network Functions with the Intelligent Plane

Dong Wen, Tao Li, Wenwen Fu, Chenglong Li, Zhuochen Fan, Hui Yang, Chao Zhuo, Lun Li, Zhiting Xiong, Junnan Li

***Abstract*—Neural-network-driven (NN-driven) network functions and their implementation on the data plane are emerging topics due to demonstrated accuracy and high performance. Meanwhile, we argue that deploying NN-driven network functions should satisfy two design goals: the generality to support various NN models, and the flexibility to operate various network functions. Unfortunately, existing work cannot satisfy both goals simultaneously.**

In this paper, we introduce the concept of the Intelligent Plane for NN-driven network functions, and propose DP4C, a cross-plane SoC architecture that integrates the intelligent, control, and data planes within a single chip. DP4C comprises the programmable NN inference engine that iteratively executes inference to ensure model generality in the intelligent plane, a multi-core RISC-V CPU that parses inference results into diverse network functions through its architectural flexibility in the control plane, and the switch fabric in the data plane. To further eliminate the performance bottleneck, we propose (i) the direct register access mechanism coupled with custom instructions to reduce the overhead of cross-plane data migration; and (ii) the multi-core pipelining with adaptive batch-processing for multi-core CPU. DP4C SoC is fabricated using 130 nm technology and has already been deployed in industrial IoT environments. We also design three distinct test cases to evaluate DP4C, fully demonstrating the model generality and operational flexibility.

***Index Terms*—NN-driven network functions, System-on-chip, Neural network inference engine, RISC-V**

I. Introduction

Compared with traditional CPU or GPU platforms, data plane is a network-native platform that has garnered considerable interest. Notably, NN implementations on programmable data planes [1]–[4] are becoming a hot topic due to their demonstrated accuracy and line-rate performance. Currently, NN-driven network functions have attracted the attention of both the academic and industrial community. For example, Broadcom latest Trident 4 switch series implements NN to support traffic analysis on the data plane [5].

Dong Wen, Tao Li, Wenwen Fu, Hui Yang, Chao Zhuo, Lun Li, Zhiting Xiong are with of the School of Computer, National University of Defense Technology, Changsha, China. (Email: wendong19@nudt.edu.cn, taoli@nudt.edu.cn, fuwenwen94@nudt.edu.cn, huiyang@nudt.edu.cn, llws_hzfc@163.com, xjtu_lilun@163.com, xiongzhiting86@163.com)

Chenglong Li is with the Defense Innovation Institute, Academy of Military Sciences, Beijing, China. (Email: chenglongli17@163.com)

Zhuochen Fan is with the Department of Strategic and Advanced Interdisciplinary Research, Pengcheng Laboratory, Shenzhen, China. (Email: fanzhch@pcl.ac.cn)

Junnan Li with the Sixty-third Institution, National University of Defense Technology, Nanjing, China. He is the corresponding author of the paper. (Email: lijunnan@nudt.edu.cn)

✓: Satisfied ■: Partly satisfied ✗: Not satisfied

Work	Solution	Model Generality	Operational Flexibility
FENXI [10]	Software (TPU)	✓	✗
BoS [1]	Software (P4)	✗	■
Leo [2]	Software (P4)	✗	■
DINC [11]	Software (P4)	✗	✗
Henna [12]	Software (P4)	✗	✗
Quark [13]	Software (P4)	✗	■
Taurus [3]	Hardware (Inline)	■	■
N3IC [4]	Hardware (Inline)	✗	✗
Ours	**Hardware (Intelligent Plane)**	✓	✓

TABLE I: Comparison between prior arts and ours.

A. Design Goals and Related Work

To deploy NN-driven network functions on the data plane, we argue that there are two critical design goals to satisfy:

- **Model generality**: The system should support generality for diverse NN models. Diverse NN architectures are employed and tailored to specific tasks. Convolutional neural networks (CNNs) [6] and recurrent neural networks (RNNs) [7] are widely used in traffic classification, while flow prediction leverages feed-forward neural networks (FFNs). Simultaneously, lightweight multi-layer perceptrons (MLPs) [8], [9] suffice for malware detection in Internet of Things (IoT) scenarios. Therefore, a viable solution should accommodate model generality for varying NN architectures.
- **Operational flexibility**: The system must offer flexibility in operating various network functions. Network functions impose fundamentally different operational requirements. For example, traffic classification may need periodic statistics reporting to the management plane, while intrusion detection demands instant ACL updates and packet drops. Therefore, operational flexibility is another necessity for NN-driven network functions.

Meanwhile, existing work fails to address these dual requirements, as revealed by our analysis in Table I. Previous work on deploying NN-driven network functions can be categorized into software or hardware solutions, as demonstrated in Table I. Software solutions aim to modify NN models for Reconfigurable Match-action Table (RMT) platforms such as P4 switches [1], [8], [13]. For instance, BoS [1] compresses NN parameters into one-bit format [14] (Binary NNs), and

replaces complex inference computation with table look-up. Although BoS achieves line-rate throughput, such a software solution suffers accuracy loss and only investigates specific types of NNs, lacking the generality to support other model types. Also, the limited programmability of the RMT architecture causes unsatisfactory operational flexibility [15], [16]. In existing hardware solutions [3], [4], pipelined accelerators are developed and integrated within the forwarding pipeline while maintaining line-rate throughput. However, such pipelined designs sacrifice model generality because accelerators cannot support NN models that exceed pipeline stages, nor can they block the forwarding pipeline by loop [17]. Moreover, these accelerators are integrated within the RMT hardware, lacking the required operational flexibility as well.

In conclusion, current research primarily focuses on modifying NNs or existing data planes, employing simplistic approaches to execute inference. However, such methods cannot meet the model generality and operational flexibility requirements. Therefore, it demands a new approach to fully support NN-driven network functions on the data plane.

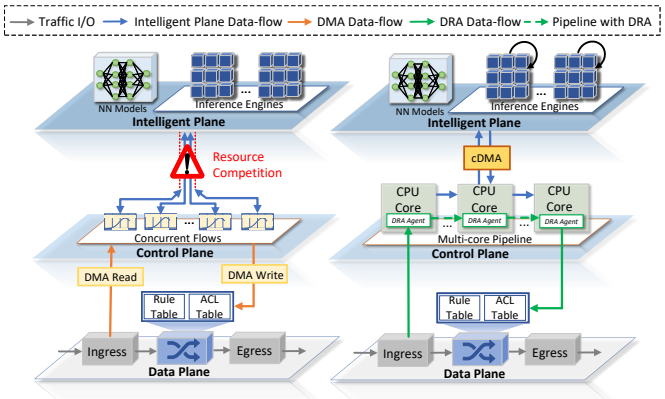
B. Motivation and Challenges

Different from prior approaches, we propose a novel cross-plane design integrating the **Intelligent Plane** in the system-on-chip (SoC) to achieve the design goals. The Intelligent Plane, defined as an independent functional plane decoupled from control and data planes, can flexibly and seamlessly execute NN inference without relying on specific optimizations like binary NNs (BNNs) that significantly compromise the accuracy.

However, the implementation of the intelligent plane brings new practical challenges, as depicted in Figure 1.a. (i) The cross-plane data migration across three functional planes introduces high latency. The complex data migration involves the data plane receiving and forwarding raw traffic to the control plane, the control plane pre-processing and extracting traffic features, inference being performed in the intelligent plane, and finally, parsing the inference results and executing network operations in the control plane (e.g., updating rule tables in the data plane). The conventional DMA mechanism consumes substantial latency, becoming the bottleneck of the system. (ii) Without a unified controlling mechanism, a limited number of inference engines struggle with high-concurrency network flows, causing performance degradation due to resource competition. (iii) The architecture faces a significant trade-off between flexibility, generality and performance. Pipelined RMT offers high performance with limited flexibility, while run-to-completion (RTC) architectures are proven to be flexible but lack performance.

C. Our Solution

This paper proposes DP4C, a novel cross-plane SoC architecture that ***integrates the intelligent, control, and data planes in a single chip***. As illustrated in Figure 1.b., different from the conventional work on the data plane, DP4C SoC introduces and implements the concept of the intelligent plane with three key components. First, it incorporates a novel intelligent plane with programmable RTC NN inference engines that



(a) The workflow in a vanilla design with intelligent plane. It faces the challenges of resource competition and low-efficiency DMA transferring.

(b) Our DP4C reorganizes the workflow of cross-plane SoC with low-latency DRA transferring and competition-free ABPipe method.

Fig. 1: Integration of the intelligent plane brings new challenges in reorganizing data-flow. Our DP4C presents DRA and ABPipe methods to address the challenges.

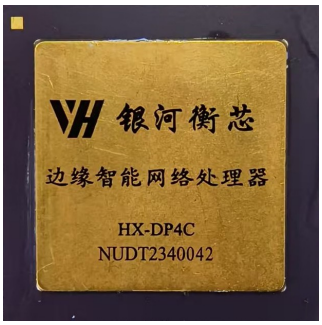
iteratively execute inference to ensure model generality, and a configurable DMA (cDMA) module optimized for inference-specific data migration. Second, the control plane integrates a multi-core RISC-V CPU that exploits architectural flexibility to dynamically execute network operations through runtime translation of inference results. Third, an essential switch fabric is designed in the data plane for traffic forwarding.

Specifically, two key innovations are presented in DP4C:

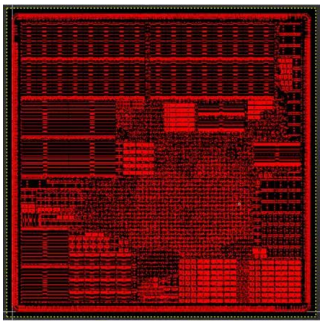
(i) *Direct Register Access (DRA) for RISC-V CPU.* DRA reduces the data migration overhead through the observation that specific traffic segments (e.g., 20-60 byte TCP headers) carry significant information on protocol state and application semantics despite their small data size. Therefore, our DRA [18] is migrated as a shortcut to the traditional memory hierarchy by directly transferring the informative data from the data plane to the CPU registers in the control plane, optimizing the continuous transfer of short messages via a register-to-register (reg-to-reg) method. Furthermore, DRA incorporates custom instruction set extensions co-optimized with the compiler and CPU micro-architecture, further improving the hardware efficiency.

(ii) *The multi-core pipeline with adaptive batch-processing (ABPipe).* A NN-driven network functions can be inherently partitioned into four pipeline stages: (1) traffic pre-processing and feature extraction, (2) the data and controlling interactions with the intelligent plane, (3) NN inference executed by the intelligent plane, and (4) performing NN-driven network operations on the data plane. Built on the low-overhead DRA mechanism, ABPipe distributes stages across different cores, significantly eliminating the disorder resource competition. Moreover, we present an adaptive batch-processing method in ABPipe, which automatically adjusts the batch-size of the inference to guarantee latency performance in the worst cases while maintaining the flexibility of the RTC inference engine and CPU cores.

As shown in Figure 2, the current version of DP4C SoC is fabricated using 130 nm technology, aiming to provide NN-driven network service in resource-constrained scenarios



(a) DP4C SoC chip.



(b) The circuit design of DP4C.

Fig. 2: DP4C SoC chip is fabricated using 130 nm technology.

such as IoT and vehicle networks. The integrated RISC-V CPU is capable of running Real-time Operating System (RTOS), providing a convenient interface to manage the hardware and deploy NN-driven network functions. We also design necessary modules like CAN-Ethernet gateway to better support IoT network scenarios. The first silicon of DP4C has completed pilot trials in multiple industrial environments, including industrial IoT gateways and naval equipment networks, fully validating the intelligent plane implementation and accumulating practical experience for future iterations.

To best of our knowledge, we are the first to present and implement the concept of the intelligent plane in the network. We evaluate DP4C SoC and the proposed intelligent plane with three test cases across malware detection, IoT traffic classification, and flow prediction, while the NN models tested include MLP, CNN, and RNN. The evaluation demonstrates the generality, flexibility, and performance of DP4C and the design of the intelligent plane.

II. Background and Motivation

A. NN-driven Network Functions

NN-driven network functions aim to leverage the power of NNs to improve the performance of traditional network functions, such as traffic classification [19], malware detection [2], traffic scheduling [20], [21], and congestion control [22]. Meanwhile, such network functions present two non-negligible characteristics.

Model generality. The architectural diversity of NN models is a common phenomenon in network functions. While foundational NNs like CNNs and RNNs establish successful baselines, researchers increasingly develop hybrid models that combine multiple types of NNs and traffic features [6], [23] to improve performance, making model generality an essential design goal. Furthermore, to adapt to the fast-changing network traffic, telecommunications operators demand that NN models continuously evolve with the latest paradigm. The field now witnesses the adoption of cutting-edge architectures, including Transformer-based models [19], Mamba [24], and even large-language-models-inspired (LLM) approaches [25]. In summary, model generality is a necessary goal for deploying NN-driven network functions on the data plane. However, neither existing software nor hardware solutions can fully support diverse NN models, failing to achieve the goal.

Operational flexibility. The various execution profiles of data plane network functions reveal fundamental operational flexibility. When deploying NN-driven malware detection, the data plane updates the ACL table and drops the malicious packets once NNs predict a flow as malware traffic [2]. While in the traffic classification task, the data plane is configured to consistently monitor traffic and periodically transfer classification information to the management plane [10], [23]. Meanwhile, in the traffic scheduling tasks, incoming flows are sent to different ports by the data plane setting new forwarding rules [20]. Such an inherent difference between network functions makes it critical for the data plane to achieve operational flexibility. Unfortunately, existing NN deployments on the data plane are established on the RMT platform, resulting in unsatisfactory flexibility due to limitations of the architecture.

B. Programmable Data Plane

The mainstream programmable data plane predominantly follows two architectural paradigms: RMT or many-core architecture, each presenting distinct trade-offs between performance and flexibility.

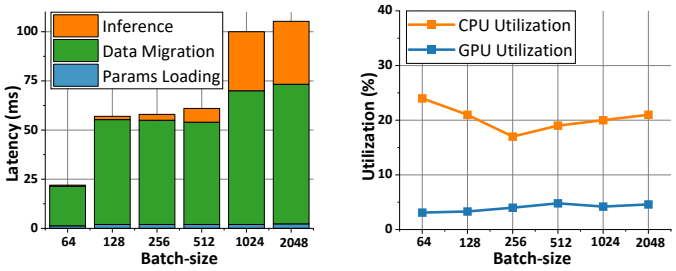
RMT-based programmable data plane. RMT architectures [17], [26] implement packet processing through deterministic multi-stage match-action pipelines. While delivering line-rate throughput for predefined processing steps, this rigid abstraction severely constrains operational flexibility. For example, complex operations like runtime telemetry message generation exceed the capabilities of fixed-stage pipelines [15]. Furthermore, due to the lack of computing capacity, RMT architectures struggle to support NN inference [3], [4].

Many-core-based programmable data plane. Many-core-based data plane integrates multiple embedded CPU cores to achieve the flexible programmability [27]. This architecture employs **thread-level parallelism** (TLP) to process individual flows on different cores, theoretically scaling with traffic concurrency. However, the TLP method causes resource competition for the intelligent plane, which has a limited number of inference engines. In addition, the weak computing capacity of the embedded core is inadequate for NN inference.

C. The Motivation of The Intelligent Plane

The motivation of the intelligent plane originates from two reasons: (i) existing work on the data plane cannot satisfy the requirements of model generality and operational flexibility, as illustrated in Section I-A and Section II-B; (ii) deploying NN-driven network functions on CPU/GPU faces problems of *inefficiency* and *under-utilization*.

Prior research has deployed NN-driven traffic classification on CPUs, TPUs, and GPUs [1], [10], [28]. However, we reproduce the experiments in [28] (a CNN-based traffic classification model) and argue that CPUs/GPUs may not be the ideal platform for such tasks. The model is evaluated on an Intel Xeon 6230R CPU and a NVIDIA 4090 GPU. **Inefficiency:** Figure 3a depicts that data migration constitutes 67.6%-91.4% of the overall latency with varying batch-size, while inference computation occupies less than 30.47%. This stems from the redundant NIC-to-CPU-to-GPU traffic path



(a) The breakdown of end-to-end latency on GPU.

(b) The hardware utilization of CPU/GPU.

Fig. 3: CPUs/GPUs are not the ideal platforms for NN-driven network functions due to the inefficiency and under-utilization.

in conventional GPU server architecture, resulting in low efficiency. **Under-utilization:** The input data and NN models utilized in network functions are much smaller than those in fields of computer visions or large language models, therefore, NN-driven network functions cannot fully leverage intensive computing resources on the CPU/GPU platform. As shown in Figure 3b, the evaluated model consumes less than 5% of the computing capacity on the GPU, and around 20% on the CPU. Moreover, inefficient data migration introduces frequent PCIe interrupts, further compromising hardware utilization. The inefficiency and under-utilization drawbacks make expensive CPU/GPU an uneconomical choice for NN-driven network functions.

In this paper, we propose the concept of the **Intelligent Plane**, an independent functional plane that supports general, flexible, and efficient inference for NN-driven network functions. The introduction of intelligent plane enables a cross-plane design where the intelligent, control, and data planes are integrated within a single SoC chip, eliminating the costly CPU/GPU offload. Furthermore, with the integration of the CPU on the control plane and the RTC inference engine on the intelligent plane, our design offers better model generality and operational flexibility compared to existing work.

III. DP4C Architecture

A. Overview

DP4C presents a novel cross-plane SoC architecture tailored for NN-driven network functions, as illustrated in Figure 4. This architecture integrates three functional planes within a single SoC: the intelligent plane, the control plane, and the data plane. The intelligent plane (detailed in Section III-B) consists of systolic-array-based inference engines programmable via matrix-multiplication (MM) instructions to support diverse NN models. Each inference engine is interconnected with the control plane through configurable DMA (cDMA), an inference-specific data migration module that eliminates the cross-plane performance bottleneck. A multi-core RISC-V CPU is integrated in the control plane with shared memory across cores and the direct register access (DRA) technique. Through reg-to-reg data migration, DRA provides a low-latency and high-efficiency interconnection between the data plane and the control plane (illustrated in Section III-C). Additionally, Section III-D introduces a multi-core pipeline using adaptive

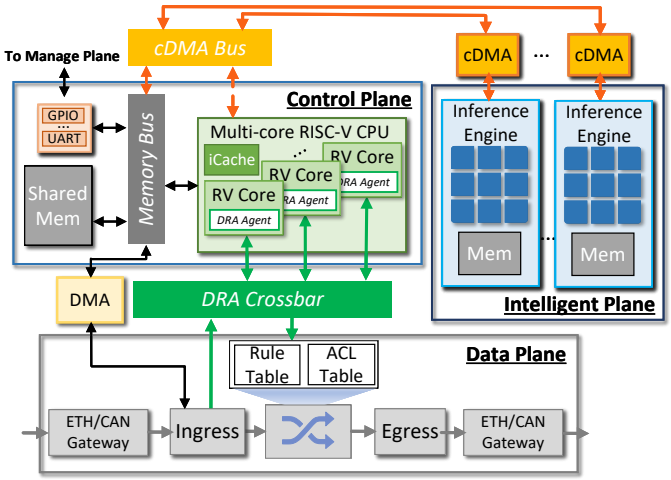


Fig. 4: The architecture of DP4C SoC. The intelligent, control, and data planes are implemented in a single SoC chip.

batch-processing (ABPipe) to mitigate resource competition and enhance performance. In the data plane (Section III-E), we design a network switch that incorporates rule tables and CAN-Ethernet hardware gateways, better serving IoT network applications.

B. Model-General Intelligent Plane

The intelligent plane serves as an independent functional plane that supports NN inference with both generality and high efficiency. To achieve these design goals, we implement a systolic-array-based accelerator coupled with configurable DMA (cDMA), a new module designed for cross-plane inference-specific data migration.

1) Inference Engine:

Computation pipeline. The computation pipeline architecture, depicted in Figure 5, is established on the systolic array [29], an MM-throughput-optimal architecture that is validated in Google TPU and prominent NN accelerators [30]. A $n \times n$ systolic array can perform MM between a (l, n) feature matrix X and a (n, n) parameter matrix W. When the systolic array produces the temporal MM results of sub-matrix blocks, the accumulator directly receives and accumulates partial products with prior accumulation results until all blocked MM iterations are completed. This seamless array-accumulator cooperation supports general NN inference through variable-dimension blocked MM. The pipeline is complemented by activation and pooling modules, enabling the intelligent plane to independently execute all inference operations.

Memory and control. Each inference engine incorporates a private memory to store NN parameters, temporal data, and instructions. Such memory also functions as the interconnection interface to the control plane, receiving traffic features as the input of NNs and outputing inference results. To optimize the constrained on-chip resource budget, our design intentionally unifies instruction and data cache. When the current instruction queue is empty, the hardware triggers the instruction fetcher (iFetch) to retrieve instruction blocks from memory. These instructions are sent to the decoder, which generates hardware control signals. For instance, the decoder calculates operand

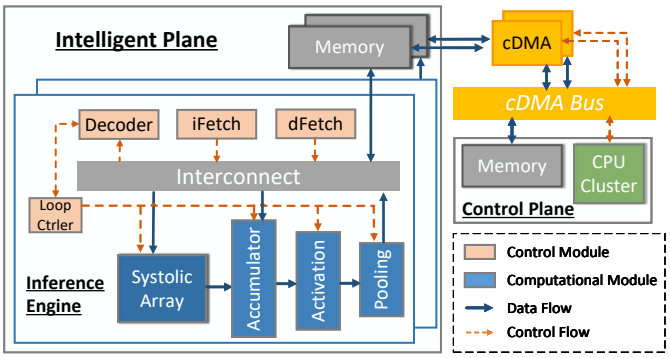


Fig. 5: The architecture of the intelligent plane and inference engine.

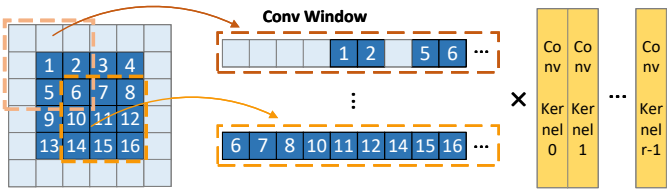


Fig. 6: `Img2col` operation. The light-blue squares represent the zero-padded elements, the blue squares are raw input.

addresses of the data fetcher (dFetch) to access data required for computation. Simultaneously, we implement a hardware loop mechanism governed by a dedicated loop controller (Loop Ctrler). By configuring the dimensional parameter l, users can dynamically adjust MM dimensions for the feature matrix X, enabling general support for NN inference across variable inputs and parameters.

2) Inference-Specific Configurable DMA:

Image-to-column operation. CPU/GPU platforms typically offload data migration tasks like image-to-column (`img2col`) to host processors. As illustrated in Figure 6, `img2col` converts convolutional operations into MM via transforming zero-padded feature maps into rows of the feature matrix by convolution windows that have variable dimensions (e.g., 3×3) and stride lengths. The feature matrix subsequently multiplies with the convolutional kernel matrix, completing inference computation. Although enabling efficient MM computation, `img2col` induces irregular memory access patterns and conditional zero-padding operations that require extensive branch prediction, load/store instructions, and address computations on CPUs, inducing substantial processing overhead.

cDMA offloading. We propose the inference-specific configurable DMA (cDMA) module to offload zero padding, address calculation, and data migration in `img2col`, fully facilitating an efficient cross-plane interconnection between the intelligent plane and the control plane. As formalized in Algorithm 1, the CPU in the control plane configures several parameters for cDMA, including the base address of the feature map X, the target address of the feature matrix M, and specifications for convolution and zero-padding. Consequently, cDMA dynamically counts the convolution window along the vertical and horizontal directions to establish the outer loop bounds (Lines 1-4). During the inner loop (Lines 5-7), elements within the same convolution window are indexed by

Algorithm 1: cDMA-offloaded `Img2col` data rearrangement and migration.

Input : Feature map X and size (h, w, c). Conv kernel size (k_h, k_w, k_c), stride (s_h, s_w). Zero-padding (z_h, z_w). h, w, c denote height, width, and channel dimensions.

Output: Feature matrix M = `img2col`(zero_pad(X))

1 $conv\_wind\_h \leftarrow \lfloor (h + 2z_h - k_h)/s_h \rfloor + 1$;
2 $conv\_wind\_w \leftarrow \lfloor (w + 2z_w - k_w)/s_w \rfloor + 1$;
3 **for** $i \leftarrow 0$ **to** $conv\_wind\_h - 1$ **do**
4 **for** $j \leftarrow 0$ **to** $conv\_wind\_w - 1$ **do**
5 **for** $k \leftarrow 0$ **to** $c - 1$ **do**
6 **for** $p \leftarrow 0$ **to** $k_h - 1$ **do**
7 **for** $q \leftarrow 0$ **to** $k_w - 1$ **do**
8 $x \leftarrow is_h + ps_h$;
9 $y \leftarrow js_w + qs_w$;
10 **if** $x \le z_h$ ***or*** $x \ge z_w + w$ ***or*** $y \le z_h$ ***or*** $y \ge z_h + h$ **then**
11 cDMA (0) $\rightarrow M[c(ij + j) + ps_w + q]$;
12 **else**
13 cDMA ($X[x + yh + kc]$) $\rightarrow M[c(ij + j) + ps_w + q]$;

x and y to identify their location on the feature map (Lines 8-9). To efficiently handle zero-padding, cDMA checks whether the current element belongs to the padding area through the (x, y) index (Line 10). For elements within zero-padding regions, cDMA directly writes zeros to the target address; otherwise, cDMA calculates the address of the required data in feature map X and transfers them into feature matrix M (Line 13). This cDMA-offloading process concurrently executes data rearrangement and migration, eliminating CPU involvement in these memory-intensive operations.

C. Flexible Control Plane and Direct Register Access

The control plane executes flexible, programmable post-inference network operations while also handling traffic pre-processing and cross-plane coordination among data, intelligent, and data planes. We implement the control plane using a multi-core RISC-V CPU for its flexibility and programmability. Moreover, the DRA technique is presented to enhance the efficiency of cross-plane data migration and interconnections by optimizing the continuous transfer of short messages. Additionally, the management plane communication is enabled through integrated GPIO/UART port configuration.

1) Multi-core CPU:

DP4C integrates CV32E40P [31], an open-source RISC-V core supporting 32-bit integer operations, multiplication/division, and compressed instruction sets. Our SoC incorporates three CV32E40P cores, each employing a sequential single-issue four-stage pipeline (instruction fetch, decode, execute, write-back) as depicted in Figure 7. These cores support multiple ISA extensions, including hardware loops and DSP-enhanced ALU instructions. Based on the extensible feature of CV32E40P, we further present and integrate DRA technology into the CPU core with customized instructions.

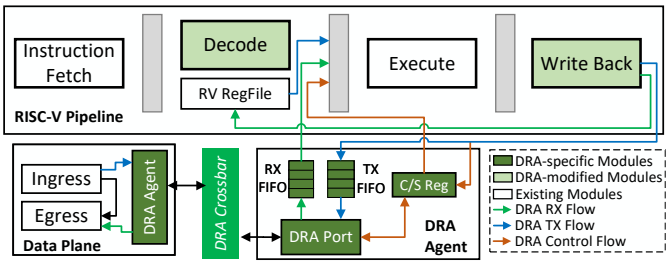
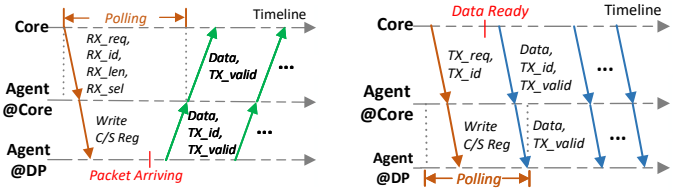


Fig. 7: The DRA architecture, its integration into the CPU pipeline, and its interface to the data plane. DRA requires only minor modifications to the decode and write-back stages of the existing RISC-V pipeline.



(a) CPU receives data from the data plane.

(b) CPU transfers data to the data plane.

Fig. 8: The DRA interaction between the CPU and data plane.

2) DRA Design and Integration:
Motivated by the observation that a small portion of traffic segments carries significant protocol and application information, we propose DRA and its CPU adaptation to optimize cross-plane data migration via reg-to-reg transfer of short messages.

DRA architecture. As shown in Figure 7, the kernel module of the DRA method is the DRA agent, which comprises the receive FIFO (RX FIFO), transfer FIFO (TX FIFO), control&state register (C/S Reg), and DRA port. Incoming data are first received by the DRA port and then pushed into RX FIFO, which has the same data width as the general-purpose registers in the RISC-V register file (RV RegFile). Customized DRA instructions enable the CPU pipeline to fetch these data from RX FIFO at the decode stage, and write back to the RV RegFile at the final stage, completing the low-overhead reg-to-reg data receiving from the data plane. Conversely, the CPU core can transfer data from RV RegFile to DRA TX FIFO via customized instructions. These data are further sent to the destination, accomplishing the transfer procedure. The CPU pipeline directly configures DRA parameters to access the C/S registers. Critically, different CPU cores can be interconnected via the DRA agents and a crossbar, facilitating the low-overhead pipeline in ABPipe (Section III-D).

DRA integration. The DRA architecture requires only minimal modifications to the existing CPU pipeline. Specifically, we integrate DRA decoding logic at the decode stage to recognize custom instructions and enable direct RX FIFO accesses. Concurrently, we augment the write back stage with direct TX FIFO write capability. These integrations deliver full DRA read/write functionality while preserving the fundamental pipeline architecture without significant alterations.

DRA interaction mechanism. In DRA method, C/S Reg is utilized to control data migration through following flags:

- *RX_req*: 1 bit, request to receive data.
- *RX_id*: 3 bits, the index of *RX_req* destination modules.
- *RX_len*: 6 bits, the length of the requested data.
- *RX_sel*: 8 bits, the data selection of a packet.
- *TX_req*: 1 bit, request to transfer data.
- *TX_id*: 3 bits, the destination index of *TX_req*.
- *TX_valid*: 1 bit, transferred data is valid.

This DRA interaction mechanism is exemplified through data plane-to-CPU migration in Figure 8. In the CPU receiving procedure, the CPU configures C/S Reg for DRA and starts polling it. A true value of *RX_req* means the CPU makes a request for receiving data from the module indexed by *RX_id* (e.g., CPU core 0: 000, core 1: 001, and the data plane: 111). *RX_len* determines the message size received by DRA, and *RX_sel* enables targeted extraction of specific packet segments (e.g., headers or top-64B payloads) These flags in C/S Reg are transferred to the DRA crossbar, which forwards the message to the destination module, namely the data plane in Figure 8a. When a new packet arrives in the data plane, the DRA agent continuously transfers selected data segments with the valid signal (*TX_valid*). The crossbar forwards data to the requesting CPU core identified by *TX_id*. Once the CPU core detects *TX_valid* during the polling, it triggers FIFO readout to complete reception. Conversely, transferring operations in Figure 8b begin with the CPU configuring C/S Reg with *TX_req* and *TX_id*. The core then transfers data with continuous *TX_valid* signal until completion.

DRA customized instructions. To preserve compatibility with the RISC-V instruction set which only has 32 general-purpose registers, we extend three DRA customized instructions: *dra_lui*, *dra_recv*, and *dra_send*. As detailed in Figure 9, these instructions leverage existing RISC-V encoding: *dra_recv* and *dra_send* mirror the *addi* format to enable the data exchange between RV RegFile and DRA FIFO, with their 12-bit immediate segments set to zero to enforce pure data movement. *dra_lui*, extends the base *lui* instruction, writes a 23-bit immediate value that carries all controlling flags to the DRA C/S Reg, which configures the DRA operations. Crucially, we implement these via the .word assembly directive per RISC-V specification, achieving binary compatibility without compiler modifications and ensuring seamless integration with the existing software/hardware environment.

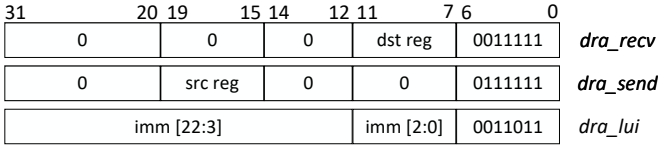


Fig. 9: The DRA-customized instructions. Src_reg and dst_reg determine the source and destination register of the DRA data migration. Unused instruction segments are padding with 0.

D. Multi-core Pipeline with Adaptive Batch-processing

To eliminate resource competition for the intelligent plane, and further improve the overall performance of DP4C, we present a multi-core pipeline technique with an adaptive batch-processing method (ABPipe). As depicted in Figure 10, ABPipe divides and distributes NN-driven network functions across the intelligent plane and different CPU cores, while

DRA and cDMA are utilized for low-overhead control and data paths. Moreover, we propose the adaptive batch-processing, which flexibly adjusts the batch-size to guarantee performance in the worst cases.

1) ABPipe Architecture:

Pipeline overview. NN-driven network functions are divided into four distinct stages and mapped to different components, as depicted in Figure 1 and Figure 10.

- **Pre-processing stage**: CPU core 0 extracts traffic features such as protocol after receiving raw traffic via DRA. The processed features are stored in shared memory, with their address and size being sent to Core 1 through DRA.
- **Interactions stage**: Based on the control and data information received by DRA, core 1 transfers extracted features to the intelligent plane by cDMA.
- **Inference stage**: When cDMA data migration is completed, inference engines execute user-defined NN inference. The inference results are transferred to the shared memory of the control plane by the cDMA module.
- **Post-inference operation stage**: Core 2 parses inference results into practical actions, for example, the ACL table is updated based on the NN results for malware detection.

Competition-free pipeline. As illustrated in Figure 1.a, conventional TLP architectures exhibit significant resource competition at the intelligent plane where concurrent traffic flows chaotically preempt inference engine resources. ABPipe eliminates this competition through disciplined pipeline staging: there is only one input stage (pre-processing), one output stage (post-inference operation), and one interaction stage. Core 1 controls all intelligent plane interactions, resolving contention by serializing access rather than allowing disordered competition among concurrent flows.

DRA-enhanced low-overhead pipeline. Frequent controlling and data exchange introduce critical bottlenecks in conventional multi-core pipelines [32]. ABPipe resolves this limitation through DRA communication. Figure 4 demonstrates how DRA agents and a crossbar interconnect enable low-latency control and data transfers between cores. Shared memory accessible to all CPU cores eliminates redundant data migration overhead, while data addresses transferred by DRA permit different cores to directly access the required data.

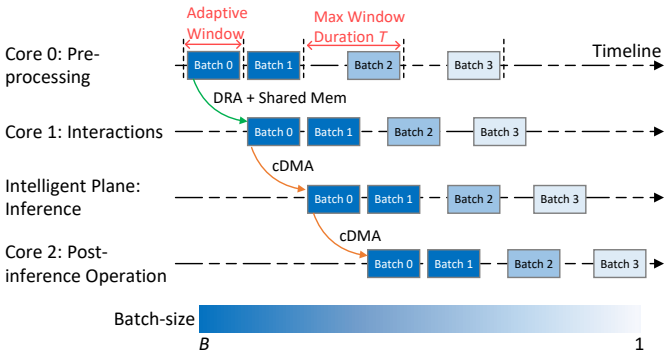


Fig. 10: The workflow of ABPipe. Blocks of varying colours correspond to adaptive batch-size, with darker colours representing larger batch-size reaching a maximum of *B*.

2) Adaptive Batch Processing:

Variable network workloads and traffic speeds render fixed batch-size inadequate for the quality of the service (QoS) requirements of NN-driven network functions [33]. Smaller batch-size reduces latency but degrades hardware efficiency, while larger batch-size in lightly-loaded networks improves accelerator utilization at the expense of long batch-waiting time. Therefore, the core objective of the adaptive batch-processing is to select the proper batch-size that guarantees performance while improving the hardware efficiency under different traffic speed and network workloads.

Adaptive method. Adaptive batch-processing aggregates discrete flows during a waiting period before issuing batches for downstream processing. We define this aggregation interval as the adaptive window in Figure 10. To guarantee the latency in the worst cases, e.g., a lightly-loaded network environment that requires a longer batch-waiting time to fill a batch, we define T as the maximum duration of the adaptive window. On the other hand, when traffic speed is fast in the network, incoming flows may fill a batch within T. In this case, we define B as the optimal batch-size, which can keep a balance between inference latency and hardware efficiency. When accumulated traffic reaches B within the current window, flows are directly sent to the next stage at once (Line 2-3 in Algorithm 2), such as batch 0 and batch 1 in Figure 10. Conversely, when network traffic arrives at a lower speed and the window duration reaches the threshold of T without filing B, the partial batch progresses immediately without waiting (Lines 4-5), as batch 2 and batch 3 in Figure 10. The adaptive batch-processing algorithm runs on core 0, which hosts the input stage of ABPipe. Parameters T and B are configurable at runtime, users can adjust these parameters due to different QoS by modifying the program on core 0.

Algorithm 2: Adaptive batch processing.

Input : Max window duration T, optimal batch-size B, current received flows f, current batch-waiting time t.

Output: Issued data batch Issue(f).

```
1 while working==True do
2     if t ≤ T and f == B then
3         Issue(f);
4     else if T==t then
5         Issue(f);
6     else
7         BatchWait();
```

E. Data Plane

Figure 4 demonstrates the design of the data plane, which follows the common store-and-forward mode with multiple look-up tables that implement ACL and rule-based forwarding functions. To fully support the deployment in IoT scenarios, DP4C data plane incorporates a specific CAN-Ethernet gateway, enabling efficient connectivity with industrial equipment such as robot arms, camera interfaces, and various sensors. While DRA is designed for reg-to-reg short message transfer, we also retain the DMA path between the data plane and the control plane as a backup channel for large-size data migration.

Components	Implementation Details
Intelligent Plane	1× inference engine with one 32×32 systolic array, 512 KB private memory.
Multi-core CPU	3× CV32E40P cores, 1 MB shared memory, 32-b×32 regfile in each core.
DRA Agent	32-b×64 TX FIFO, 32-b×64 RX FIFO.
Data Plane	4 Gbps Ethernet switching, 4× 1000 Mbps PHY ports, 5× CAN interfaces, 64KB packet cache, 2K-size flow table.

TABLE II: The hardware implementation details.

IV. Implementation and Test Bed

A. Hardware Implementation

As the first version and the trial product of the SoC family, DP4C is taped out using 130 nm technology. Figure 2 shows the picture of DP4C SoC chip product and its circuit design. Constrained by the chip manufacturing technique, DP4C occupies a size of $99.2mm^2$ and consumes power of 1.3 Watt while running at 150 MHz. Designed for the IoT scenarios, DP4C has 4 Gbps line-rate forwarding throughput that supports four 1000M PHY ports and five CAN-Ethernet gateways. The systolic array implemented in the inference engine has a size of 32×32, offering 307.2 GOPs of computing power. The capacity of data forwarding and NN inference are able to satisfy the requirements of IoT applications [34]. We use Verilog HDL for development, and the detailed hardware implementation configurations are listed in Table II.

B. Software Implementation, Compilation, and Execution

Implementation. We integrate Free RTOS [35] on the DP4C SoC, providing the basic interaction and debug functions for the DP4C. Concurrently, DP4C integrates a global time register that triggers timing interrupts, enabling the multi-core CPU to perform periodic task switching.

Compilation. The compilation of DP4C involves two distinct procedures: compiling programs for CPU cores in the control plane and compiling NN models for inference engines in the intelligent plane. The compiler for CPU cores is established on the standard riscv-gnu-tool-chain [36] for 32-bit architecture and the IMC instruction set, enabling DP4C to support Linux systems. Following the common practice [37], the tool-chain for inference engines begins with analyzing the computation graph exported from the framework, such as PyTorch, decomposing the NN inference into blocked MM, activation, and other computations. A Python script for compilation is developed to transform these operations into instructions for the inference engine. Simultaneously, this script will check whether the NN models can fit within the DP4C on-chip memory capacity, or raise an error prompting users to adjust NN models. Finally, a bin file that comprises computing instructions and parameters is generated by the Python script for a valid NN model.

Execution. Constrained by on-chip memory, we adapt an offline-compiling and online-bootload paradigm that supports two program execution methods on DP4C: (i) Bin files generated by the DP4C tool-chain can be pre-loaded into on-board Flash storage, and the RTOS running on the DP4C can execute the program through accessing Flash. (ii) Following prior data plane designs [17], [26], DP4C can be programmed via control packets, specific network messages that carry program bin files. ***The detailed tool-chain, user-guide, and demo workflow for DP4C are open-sourced at [38].***

C. Test Bed

We build a test bed with a motherboard installing DP4C, a TOYO Corporation SYNESIS traffic replayer, a monitor host, and a testing host, as shown in Figure 11. The traffic replayer can replay the traffic traces from the datasets utilized in Section V with variable speed. The monitor host functions as the management plane, which receives messages from DP4C or reads the rule tables to calculate the inference accuracy of the tested models deployed on the SoC.

Fig. 11: The test bed for evaluating DP4C.

V. Setup, Test Cases, and Evaluated Models

A. Experimental Setup

We evaluate the DP4C SoC under three typical test cases of NN-driven network functions using the following configurations, as detailed in Table IV. These cases employ distinct input features, output operations, and NN models to evaluate the flexibility and generality of DP4C.

NNs training and evaluation metrics. We train all NNs on Intel Xeon Golden 6230R and NVIDIA 4090 GPU, supported by Ubuntu 22.04 and PyTorch 1.12. Model performance in Table III is evaluated using accuracy (Acc), precision (PR), recall (RC), and F1-score (F1). We also reproduce the NN models utilized in baseline approaches with the same training environment and evaluation metrics.

ABPipe configurations. We leverage DRA mechanism to organize the low-overhead multi-core pipeline. In adaptive batch-processing, we let $B = 72$, the maximum window duration in ABPipe is set to $T = 800$ microseconds (us). This configuration complies with the 1000-us real-time in-network inference requirement suggested in prior work [39].

B. Test Case 1: Malware Detection

Malware detection constitutes a critical network function, identifying whether incoming flows contain malware traffic. For this task, we train and test a 5-layer 1D-CNN model using the IDS-2017 dataset [40] (32533 training flows; 13943 testing flows). Following standard pre-processing practices [19], we strip Ethernet headers, IP addresses, and ports from the first packet in a flow, providing the top 64 bytes of the remaining payload as model input. In the post-inference procedure, the control-plane CPU core identifies malware flows, updates the ACL table in the data plane, and blocks malware traffic.

Work	NN model	#Layer	Size	Acc	PR	RC	F1
Test Case 1: Malware Detection (IDS-2017)							
BoS [1]	Binary RNN	4	6.8KB	87.85%	93.15%	81.80%	87.48%
Taurus [3]	RNN	4	0.24KB	88.03%	87.71%	90.12%	88.92%
N3IC [4]	Binary MLP	3	0.8KB	82.04%	84.17%	85.05%	84.61%
Ours (@DP4C)	CNN	5	461.0KB	**99.65%**	**96.73%**	**99.44%**	**99.62%**
Test Case 2: IoT Traffic Classification (IOT-2022)							
BoS [1]	Binary RNN	4	6.8KB	74.92%	76.11%	74.75%	75.43%
Taurus [3]	MLP	4	0.25KB	84.25%	87.22%	82.97%	84.53%
N3IC [4]	Binary MLP	3	0.8KB	62.97%	65.50%	51.04%	58.27%
Ours (@DP4C)	RNN	**5**	259.0KB	**94.42%**	**92.57%**	**93.89%**	**92.87%**
Test Case 3: Prediction on Elephant and Mice Flows (USTC-2016)							
BoS [1]	Binary RNN	4	6.8KB	88.49%	88.73%	89.15%	88.94%
Taurus [3]	MLP	4	0.25KB	87.06%	87.01%	86.95%	86.98%
N3IC [4]	Binary MLP	3	0.8KB	68.46%	69.57%	68.07%	68.82%
Ours (@DP4C)	MLP	3	4.4KB	**93.90%**	**93.96%**	**93.97%**	**93.96%**

TABLE III: The comparison of NN accuracy. Both BoS and N3IC leverage binary NNs, where parameters and features are reduced into 1-bit. The matrix multiplication is transformed into bit-wise operations like `and`, significantly harming accuracy.

Test Case	Input	Output	Dataset	Model
Malware detection	64-Byte raw bytes	Updating ACL table	IDS-2017 (30.8GB)	CNN 5 layers
IoT TC	pkt_size, pkt_IAT	Msgs to mgt plane	IOT-2022 (5GB+)	RNN 5 layers
Pred. on EM flow	±duration, ±flow_size, ±avg_pkt_size, ±avg_pkt_IAT	Updating rule table	USTC-2016 (20GB+)	MLP 3 layers

TABLE IV: Implemented test cases. (avg_)pkt_IAT represent the (average) packet arriving interval time. ± means the traffic in two directions of a session.

C. Test Case 2: IoT Traffic Classification

In IoT edge networks connecting diverse devices (e.g., smart screens, sensors, and cameras), traffic classification can identify device working states. The IoT-2022 dataset [41] exemplifies this, capturing traffic from devices in Power, Idle, and Active states, each of which exhibits distinct network characteristics. We train and test a 5-layer RNN model (39262 training flows; 13087 testing flows) using sequences of top-10 packet sizes and inter-arrival times as input features. The post-inference operation on the control plane is to generate and send device status inference results to the management plane with a 0.1 second period.

D. Test Case 3: Prediction on Elephant and Mice Flows

Elephant/mice flow prediction is critical for network optimization, as elephant flows (long duration, large size) and mice flows (short duration, small size) demand distinct QoS requirements. This test case utilizes the USTC-2016 dataset [42], covering four elephant flow types (e.g., video, voice, and file transfer) and four mice flow types (e.g., email and social networks). We split the data into 526428 training flows and 225612 testing flows. A three-layer MLP leverages statistical traffic features from top-10 packets for classification. The control plane updates forwarding rules based on the inference results: mice flows are prioritized to low-latency paths, while elephant flows are directed to high-bandwidth paths.

VI. Evaluation

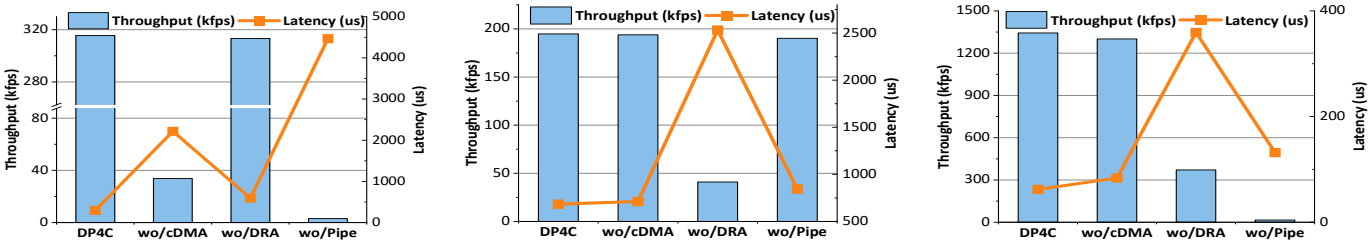
A. End-to-end Evaluation on Test Cases

Accuracy Performance and Model Generality. Due to the model generality of DP4C, we select the best-match NN models to deploy and achieve outstanding accuracy, as listed in Table III. On-chip NNs consistently exceed 93% accuracy and attains over 99.5% accuracy and F1-score in malware detection. The intelligent plane abstraction decouples inference from the data plane, eliminating NN size and type limitations and supporting models spanning CNN, RNN, and MLP architectures, with sizes ranging from 4.4 KB to 461.0 KB. To conclude, the experimental results reflect the critical importance of model generality to accuracy, and prove the model generality of DP4C.

Throughput and Latency Performance. As established in Section IV-C, the traces from datasets are replayed at 4 Gbps, the full line rate of DP4C. Figure 12 depicts throughput and latency in three test cases. DP4C sustains a 4 Gbps line rate while delivering 194.8 to 371.3 kilo-flows per second (kfps) throughput. This performance satisfies NN-driven function requirements for IoT environments. Simultaneously, DP4C achieves 62.1-683.6 us end-to-end latency, which meets real-time in-network inference requirements in prior studies [39]. The throughput and latency variations across test cases stem from differences in their traffic features, NN models, and post-inference operations. We detail and analyze these performance differences in Section VII-E.

B. Comparison

1) Baseline Approaches on the Data Plane:
We select BoS [1], a software-based approach for deploying NN-driven network functions on P4 device, as a baseline, alongside two hardware solutions: Taurus [3] and N3IC [4]. The implementation details of the baselines are as follows:



(a) Test case 1 performance. (b) Test case 2 performance. (c) Test case 3 performance.

Fig. 12: The throughput and latency performance of three test case.

- **BoS** represents the state-of-the-art software approach for deploying NN-driven network functions on Tofino switches. This method implements binary RNNs (BRNNs) that compress both weights and activations to 1-bit representations, transforming matrix multiplication into bit-wise operations such as XOR.
- **Taurus** integrates a pipelined NNs accelerator into the P4 architecture. To maintain line-rate forwarding, Taurus supports only tiny NN models that fit in the inference pipeline. The design is implemented and evaluated using 15 nm ASIC technology with 1.0 GHz running frequency.
- **N3IC** features a pipelined NN accelerator integrated within NIC devices, supporting binary MLP (BMLP) model inference. The system was implemented and evaluated on an FPGA-based testbed operating at 250 MHz.

We compare DP4C with baselines across four criteria: model generality, accuracy, operational flexibility, and normalized performance using three test cases.

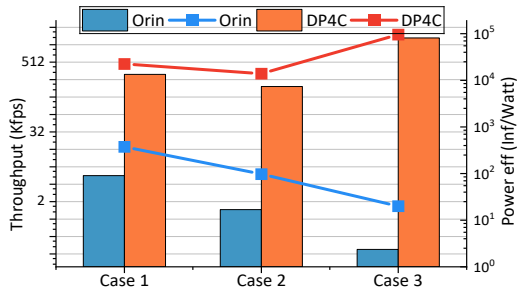
2) Comparison on Model Generality and Accuracy: DP4C outperforms all baselines in accuracy as detailed in Table III, achieving the highest accuracy across all test cases. Due to the model generality of RTC inference engine, DP4C supports best-match NN models for distinct tasks by leveraging CNN, RNN, and MLP for the three test cases, respectively, delivering 5.41%–31.45% higher accuracy.

Baselines exhibit fundamentally limited model generality that compromises accuracy. BoS supports only BRNN models, Taurus accommodates NN models constrained to 0.25KB, and N3IC executes exclusively 0.8KB-size BMLP models, with small model capacities and binarization techniques further harming accuracy. This constrained generality prevents baselines from leveraging optimal models for different tasks, while DP4C can support different NN models up to 1920.8× larger.

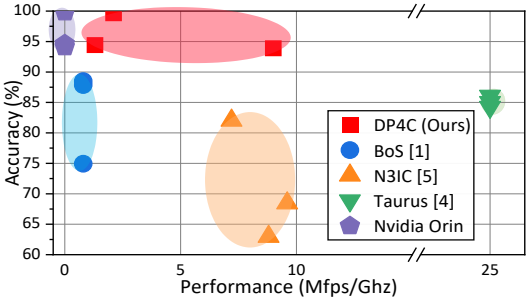
3) Comparison on Operational Flexibility: We evaluate flexibility through three post-inference operations: updating ACL table, generating traffic classification results for the management plane, and updating forwarding rule table. Using CPU programmability, DP4C can execute all operations, achieving its operational flexibility design goal. Meanwhile, baselines cannot fully satisfy operational flexibility requirements as compared in Table V. BoS fails to generate classification messages for the management plane due to P4 switch limitations. Similarly, the NIC-integrated design in N3IC restricts it to ACL table updates. Taurus executes operations at per-packet granularity without configuring the global state, failing to achieve flexibility.

Op.	Updating ACL	Msgs to mgt plane	Updating rules
BoS	✔	✗	✔
N3IC	✔	✗	✗
Taurus	✗	✗	✗
DP4C	✔	✔	✔

TABLE V: Comparison on operational flexibility. ✔ means the method supports the operation, while ✗ means cannot support.



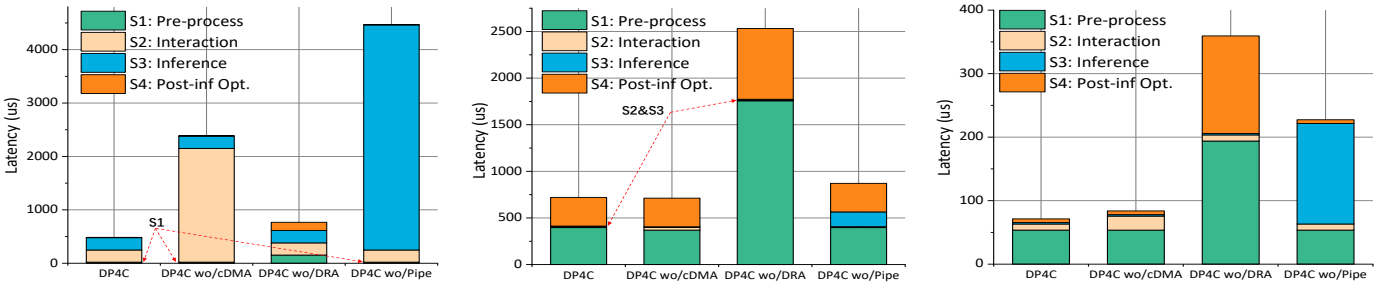
(a) Performance and power efficiency comparison to advanced ESoC.



(b) Normalized comparison to baselines.

Fig. 13: Comprehensive comparison results.

4) Comparison on Edge SoC and Power Efficiency: To fully compare the end-to-end performance and power efficiency between DP4C and advanced Edge SoC (ESoC) for NN workloads, we introduce NVIDIA Orin, a mainstream ESoC featuring 1.3 Ghz running frequency, 275 TOPs computing capacity, and 15 Watts (W) power consumption. The Orin platform runs Ubuntu 20.04 with PyTorch 1.13.1, CUDA 11.6, and cuDNN 8. Across three test cases, Orin achieves end-to-end throughput below 6 Kfps, being 34.6× lower than DP4C, as shown in Figure 13a. This performance disparity stems from architectural inefficiency of ESoC: consistent with observations in Section II-C, the majority of end-to-end latency on ESoC is dominated by data migration from the network interface to the computing components. In fact, data migration consumes on average 72.80% of total time, while inference



(a) Test case 1 latency breakdown. (b) Test case 2 latency breakdown. (c) Test case 3 latency breakdown.

Fig. 14: The latency breakdown in three test cases.

accounts for only 7.18% across three test cases. Such architectural inefficiency directly impacts the power efficiency (power eff) of ESoC: with data migration consuming substantial power, Orin ESoC only performs 20.0-375.32 inference-per-watt (Inf/Watt). In contrast, DP4C achieves 13.91K-81.05K Inf/Watt, showcasing better power efficiency. The comparison of two dimensions significantly proves the motivation of this work: by integrating the **Intelligent Plane** into network SoCs, DP4C fundamentally improves the performance and power efficiency for NN-driven network functions.

5) Comparison on Comprehensive Performance: Baselines are implemented on various hardware platforms: BoS runs on a 1.25 GHz Tofino switch, Taurus is evaluated using 15 nm technology, N3IC employs a 250 Mhz FPGA platform, and Nvidia Orin integrates the 1.3 Ghz SoC. We employ normalized millions-of-flows-per-second-per-GHz (Mfps/GHz) as a fair metric for performance, with accuracy being incorporated for a comprehensive comparison.

Figure 13b reflects that Taurus has the highest throughput but exhibits lower accuracy due to its pipelined architecture that is inherently limited to support tiny NNs, leading to high throughput and compromised accuracy. N3IC demonstrates higher throughput efficiency than DP4C in certain scenarios using through BMLP models that reduce computing costs. However, such a design significantly sacrifices accuracy. Similarly, BoS adopts BRNNs on the P4 switch, compromising accuracy for deployability on the programmable data plane. However, constrained by rigid RMT architecture of P4, BoS has lower inference efficiency than DP4C. Edge SoC Orin achieves comparable accuracy to DP4C due to its flexible support of NN models, but exhibits the lowest throughput efficiency caused by redundant data migration. We acknowledge that the RTC architecture adopted by the DP4C inference engine may yield inferior throughput performance compared to other hardware solutions. However, the comprehensive comparison in Figure 13b shows that DP4C strikes a balance between accuracy and throughput, achieving the trade-off between generality, flexibility, and throughput.

VII. Ablation Study and Deep Dive

Ablation studies are conducted on the proposed cDMA, DRA, and ABPipe. Figure 12 and Figure 14 demonstrate their impact on throughput and latency performance in detail.

A. cDMA vs. DMA

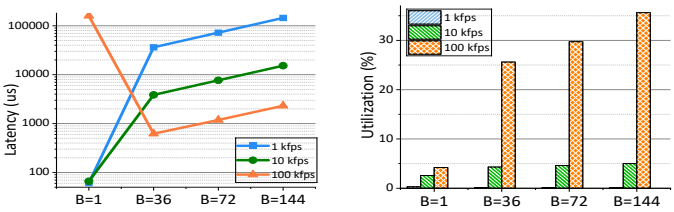
To evaluate the impact of cDMA, we disable this inference-specific data migration mechanism (labeled wo/cDMA in Figures 12 and 14) and revert to conventional DMA. Figure 12 shows that cDMA removal increases end-to-end latency by 2.02–15.13× and reduces throughput by 0.03%–99.02%, with the most severe degradation observed in test case 1. These results confirm the effectiveness of cDMA. The limited improvement in test cases 2 and 3 stems from their input characteristics: Unlike test case 1, which processes B×64 matrices via `img2col`, test cases 2 and 3 use short feature vectors, resulting in lower interaction overhead regardless of the migration mechanism. In test case 1 (Figure 14a), data migration consumes 47% of the baseline latency, while operations wo/cDMA occupy 91.1% of latency, demonstrating the efficiency of cDMA for interaction-constrained workloads.

B. DRA vs. DMA

Similarly, we disable DRA and revert to DMA-based data migration. Results (labeled wo/DRA in Figures 12 and 14) demonstrate that the throughput decreases by 78.95% and 72.36%, while the latency increases by 3.70–5.79× in test cases 2 and 3, respectively. We analyze the DRA impacts on pre-processing and post-inference stages separately.

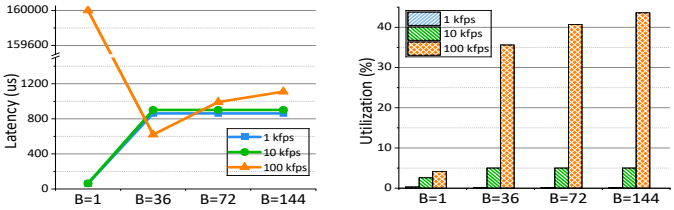
Impact on pre-processing stage. Both test cases 2 and 3 adopt traffic features that demand frequent cross-plane data migration. For example, the avg_pkt_size adopted by test case 3 requires the transfer of 10 packets per flow to calculate accurate values, increasing the overhead of cross-plane data migration. The breakdown results in Figure 14a and Figure 14b also confirm high migration overhead: these two test cases are pre-processing-constrained cases, where traffic pre-processing exceeds 54% of total latency. Without DRA technique, the pre-processing stage consumes 2.1-2.9× more time due to inefficient CPU register transfers. Test case 1, however, avoids this constrained by using simple features (first packet's top 64 bytes) that require minimal migration, which explains the negligible effect of DRA in the experiment.

Impact on post-inference operation stage. Post-inference operations exhibit distinct characteristics across test cases. Test case 1 requires only ACL updates for the malware flow. While test case 2 generates classification messages every 0.1 second, and test case 3 demands per-flow forwarding rule updates, making post-inference stage particularly time-consuming in these two cases. As Figures 14b and 14c confirm, disabling DRA increases post-inference latency by 4.43×-3.66×, making this stage as the secondary performance bottleneck after pre-processing. In summary, our ablation results demonstrate that DRA significantly optimizes cross-plane data migration during both pre-processing and post-inference operations.



(a) Latency. (b) Hardware utilization.

Fig. 15: The latency and hardware utilization under different batch-size B and traffic speed ***without*** the adaptive method.



(a) Latency. (b) Hardware utilization.

Fig. 16: The latency and hardware utilization under different batch-size B and traffic speed ***with*** the adaptive method.

C. ABPipe Ablation

We present ABPipe to eliminate resource competition and improve inference performance. In the ablation study, we disable ABPipe and revert to the first-come-first-served mode (wo/Pipe in Figure 12 and Figure 14). The results show that ABPipe has a noteworthy influence on overall performance, improving throughput by 105.16× and 84.48× in test cases 1 and 3, and reducing latency by 4.44× in test case 2. Test case characteristics significantly influence the impact of ABPipe. The complex CNN model in test case 1 introduces an inference-constrained workload, where NN inference occupies over 50.5% of time. By eliminating resource contention and enabling batch processing, ABPipe leads to substantial gains: 105.16× throughput increase with 18.25× inference latency reduction. While in test case 2, the throughput improvement of ABPipe is limited due to the pre-processing bottleneck. Test case 3, on the other hand, exhibits the shifting of bottleneck from pre-processing to the inference stage without ABPipe in Figure 14c. The resource competition introduces significant degradation of the inference performance, increasing latency by 68.87× and making inference the primary bottleneck of the system. The throughput performance is reduced to only 1.18%. Experiments and analysis fully demonstrate the effectiveness of ABPipe in accelerating inference and enhancing throughput, particularly in inference-constrained scenarios.

D. Deep Dive on Adaptive Batch-processing

We design the adaptive batch-processing method for dual objectives: (i) meeting latency-sensitive QoS requirements, and (ii) maximizing hardware utilization efficiency under different traffic speeds. We conduct ablation experiments on test case 1 to validate the adaptive batch-processing in ABPipe, as depicted in Figure 15 and Figure 16. The maximum batch-size B is set from 1 to 144, and the traffic speed in the network ranges from 1 kfps to 100 kfps.

When operating without the adaptive method, ABPipe uses fixed batch-size B with first-come-first-serve mode. When $B = 1$, it achieves sub-100-us latency under light network loads (1-10 kfps) by eliminating batch-waiting latency. However, at the 100 kfps traffic, this configuration suffers catastrophic latency degradation to 160k us due to low utilization below 4%. Such hardware efficiency is insufficient for high-speed concurrent flows. Conversely, with a large batch-size of $B \geq 32$, ABPipe primarily incurs batch-waiting penalties, especially when traffic speed is 1 kfps and 10 kfps. Under 100 kfps traffic speed, the latency performance is improved from 144874 us to 2314.5 us due to the reduced batch-waiting time. Although a larger batch-size improves the hardware utilization, these configurations still violate 1000-us QoS requirements. Thus, without the adaptive batch-processing, neither latency nor hardware efficiency objectives are fully satisfied.

We further validate the effectiveness of the adaptive batch-processing method in Figure 16. The max window duration is configured as $T = 800$ us, which means ABPipe issues the current batch immediately once the window duration reaches the threshold. This configuration achieves latency improvements of 861.8–901.us at 1–10 kfps when $B \geq 32$, resulting in a 168.11× reduction in worst-case latency. Under 100 kfps network loads, latency remains ≤1120us while hardware efficiency reaches 35.6–43.6%. Specifically, the $B = 72$ case achieves a balance between latency and utilization under different traffic speeds, therefore we choose the setting in other evaluations. These results demonstrate how T constrains batch-waiting time to reduce inference latency, while the adaptive batch-processing enhances hardware utilization.

E. Discussion: Where is the Bottleneck?

After evaluating DP4C on three test cases, we wonder *which part is the bottleneck in NN-driven network functions?*

Three dedicated designed test cases exhibit entirely different bottleneck characteristics. Test case 1 is inference-constrained with complex NN model, while test case 2 is both pre-processing- and post-inference- constrained where corresponding stages occupy over 97% of end-to-end time. Concurrently, test case 3 is one of the typical pre-propressing-constrained cases. The above observations profoundly reveal the variety of NN-driven network functions: various input features, NN models, and post-inference operations could change the bottleneck in practical deployment, the most focused inference stage may not be the practical bottleneck of the system. This also proves the necessity of model generality and operational flexibility proposed in the design goals. Existing solutions focusing solely on inference optimizations prove inadequate for addressing system-level problems. In contrast, our techniques demonstrate comprehensive optimization effectiveness against identified bottlenecks: cDMA accelerates inference-specific data migration, DRA reduces cross-plane migration overhead, and ABPipe significantly enhances inference throughput. Collectively, these optimizations enable the efficient deployment of end-to-end NN-driven network functions on DP4C.

F. Scalability: For Future Architectures and Models

Although most NNs used in network functions are compact to meet the demands of low-latency and high-throughput traffic, it is still important to consider the potential scalability of DP4C to support more complex NN models in the future.

Architectural support for advanced models. Advanced models such as Transformers, graph neural networks (GNNs), and language models (LMs) also hold promise for network tasks. Regardless of the NN variant, the dominant computation remains GEMM, which can be accelerated by programmable inference engines in DP4C, with non-linear operations like `Softmax` in Transformers and LMs being offloaded to CPUs. The programmability of DP4C enables general support for such advanced NNs. We acknowledge that executing intensive non-linear computation on CPUs is inefficient; however, smaller fabrication nodes can offer higher on-chip transistor density, allowing DP4C to evolve with larger inference engines, higher-performance CPUs, and specific components for `Softmax` in Transformers and LMs.

Chiplet technology for future architectures. As a popular method, chiplet technology can integrate dies of more CPU clusters (control plane), inference engines (intelligent plane), and data plane into a single, larger chip. This integration offers higher computing power and larger memory capacity, better supporting complex NNs. Additionally, with proper load-balancing methods across different chiplets, DP4C could efficiently process higher bandwidth traffic, extending its application from IoT environments to the Internet and beyond.

Hierarchical inference for future models. When NN models exceed on-chip memory capacity, DRAM or HBM becomes necessary. However, the resulting memory access overhead can become a bottleneck, making it difficult to match the low-latency, high-speed traffic. A hierarchical inference approach may address this issue: lightweight NNs reside in on-chip memory to provide low-latency, high-throughput inference for all incoming traffic, while more complex NNs are reserved for high-priority flows, producing higher-accuracy results under relaxed latency requirements. The CPU within DP4C offers the programmability to implement customized hierarchical inference strategies, enabling a balance among performance, accuracy, and the deployment of larger models.

VIII. Conclusion

In this paper, we propose and implement the concept of the **Intelligent Plane** in DP4C SoC architecture that fully supports NN-driven network functions. Experiments demonstrate that with the presented DRA and ABPipe methods, DP4C successfully achieves all design goals.

IX. Acknowledgment

We appreciate the Editor and Reviewers for their advice on this article. This work is supported by the National Natural Science Foundation of China (NSFC) (No. U24B20151).

References

[1] J. Yan, H. Xu, Z. Liu, Q. Li, K. Xu, M. Xu, and J. Wu, "Brain-on-switch: Towards advanced intelligent network data plane via nn-driven traffic analysis at line-speed," *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 419–440, 2024.

[2] S. U. Jafri, S. Rao, V. Shrivastav, and M. Tawarmalani, "Leo: Online ml-based traffic classification at multi-terabit line rate," *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 1573–1591, 2024.

[3] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: a data plane architecture for per-packet ml," *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1099–1114, 2022.

[4] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 513–533, 2022.

[5] "Broadcom introduces first switch chip with neural networks in industrys," 2024. [Online]. Available: https://investors.broadcom.com/news-releases/news-release-details/broadcom-introduces-industrys-first-switch-chip-neural-network

[6] K. Wang, G. Jian, and L. Xinyan, "Mtc: A multi-task model for encrypted network traffic classification based on transformer and 1d-cnn." *Intelligent Automation & Soft Computing*, 2023.

[7] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*, pp. 1171–1179, 2019.

[8] K. Zhang, S. Nancy, and K. Ahmed, "An intelligent data plane with a quantized ml model for traffic management," *IEEE NOMS 2023-2023 IEEE Network Operations and Management Symposium*, pp. 1–9, 2023.

[9] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," *2016 international conference on wireless networks and mobile communications (WINCOM)*, pp. 258–263, 2016.

[10] M. Gallo, A. Finamore, G. Simon, and D. Rossi, "Fenxi: Deep-learning traffic analytics at the edge," *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 202–213, 2021.

[11] C. Zheng, H. Tang, M. Zang, X. Hong, A. Feng, L. Tassiulas, and N. Zilberman, "Dinc: Toward distributed in-network computing," *Proceedings of the ACM on Networking*, vol. 1, no. CoNEXT3, pp. 1–25, 2023.

[12] A. Akem, M. Gucciardo, and M. Fiore, "Henna: Hierarchical machine learning inference in programmable switches," *Proceedings of the International Workshop on Native Network Intelligence*, pp. 1–7, 2022.

[13] M. Zhang, L. Cui, X. Zhang, F. P. Tso, Z. Zhen, Y. Deng, and Z. Li, "Quark: Implementing convolutional neural networks entirely on programmable data plane," *arXiv preprint arXiv:2501.15100*, 2025.

[14] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[15] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "Panic: A high-performance programmable nic for multi-tenant networks," *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 243–259, 2020.

[16] M. Yang, A. Baban, V. Kugel, J. Libby, S. Mackie, S. S. R. Kananda, C.-H. Wu, and M. Ghobadi, "Using trio: juniper networks' programmable chipset-for emerging in-network applications," *Proceedings of the ACM SIGCOMM 2022 Conference*, pp. 633–648, 2022.

[17] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[18] C. Li, T. Li, J. Li, W. Fu, and B. Wang, "Dra: Ultra-low latency network i/o for tsn embedded end-systems," *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pp. 01–10, 2023.

[19] G. Zhou, X. Guo, Z. Liu, T. Li, Q. Li, and K. Xu, "Trafficformer: an efficient pre-trained model for traffic data," *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 102–102, 2024.

[20] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Wan, L. Liu, Z. Ding *et al.*, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 1345–1358, 2022.

[21] A. Dietmüller, S. Ray, R. Jacob, and L. Vanbever, "A new hope for network model generalization," *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pp. 152–159, 2022.

[22] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin, "Multi-objective congestion control," *Proceedings of the Seventeenth European Conference on Computer Systems*, pp. 218–235, 2022.

[23] I. Akbari, M. A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, and S. Tuffin, "A look behind the curtain: traffic classification in an increasingly encrypted web," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 1, pp. 1–26, 2021.

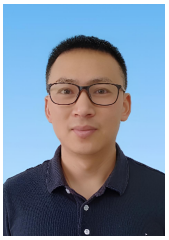
[24] T. Wang, X. Xie, W. Wang, C. Wang, Y. Zhao, and Y. Cui, "Netmamba: Efficient network traffic classification via pre-training unidirectional mamba," *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, pp. 1–11, 2024.

[25] D. Wu, X. Wang, Y. Qiao, Z. Wang, J. Jiang, S. Cui, and F. Wang, "Netllm: Adapting large language models for networking," *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 661–678, 2024.

[26] T. Wang, X. Yang, G. Antichi, A. Sivaraman, and A. Panda, "Isolation mechanisms for high-speed packet-processing pipelines," *19th USENIX*

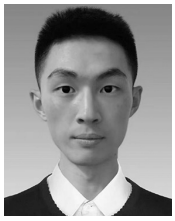
Symposium on Networked Systems Design and Implementation (NSDI 22), pp. 1289–1305, 2022.
[27] S. Di Girolamo, A. Kurth, A. Calotoiu, T. Benz, L. Benini, and T. Hoefler, "A risc-v in-network accelerator for flexible high-performance low-power packet processing," *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 958–971, 2021.
[28] G. Xie, Q. Li, and Y. Jiang, "Self-attentive deep learning method for online traffic classification and its interpretability," *Computer Networks*, vol. 196, p. 108267, 2021.
[29] H. T. Kung and C. E. Leiserson, "Systolic arrays (for vlsi)," *Sparse Matrix Proceedings 1978*, vol. 1, pp. 256–282, 1979.
[30] Y. Li, M. Wen, J. Shen, Z. Chen, Y. Shi, and Z. Shao, "Map-sim: A dnn-specific mapping optimization framework for shared-memory cpu-systolic array architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2025.
[31] "cv32e40p open-source risc-v cpu," 2020. [Online]. Available: https://github.com/openhwgroup/cv32e40p
[32] Q. M. Nguyen and D. Sanchez, "Pipette: Improving core utilization on irregular applications through intra-core pipeline parallelism," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 596–608, 2020.
[33] S. Selvam, A. Nagarajan, and A. Raghunathan, "Efficient batched inference in conditional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.
[34] C. Li, Z. Li, T. Li, C. Li, and B. Wang, "A deterministic embedded end-system tightly coupled with tsn schedule," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3707–3719, 2023.
[35] "Free rtos," 2022. [Online]. Available: https://github.com/FreeRTOS
[36] "Risc-v tool-chain," 2022. [Online]. Available: https://github.com/riscv-collab/riscv-gnu-toolchain
[37] S. Cao, M. Yang, Y. Liu, Y. Li, B. Zhao, X. Chen, and Z. Jiang, "A heterogeneous cnn compilation framework for risc-v cpu and npu integration based on onnx-mlir," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. (TCAD)*, 2025.
[38] "The open-source homepage of dp4c," 2025. [Online]. Available: https://github.com/JunnanLi/FL-M32
[39] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, Z. Shen, Y. Xi *et al.*, "Flow event telemetry on programmable data plane," *Proceedings of the ACM SIGCOMM 2020 Conference*, pp. 76–89, 2020.
[40] "Cic-ids-2017 datasets," 2017. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html
[41] "Cic-iot-2022 datasets," 2022. [Online]. Available: https://www.unb.ca/cic/datasets/iotdataset-2022.html
[42] "Ustc-tfc datasets," 2017. [Online]. Available: https://www.unb.ca/cic/datasets/vpn.html

Dong Wen received the bachelor's degree in computer science from Northeastern University, Shenyang, China, in 2019. He is currently pursuing the Ph.D. degree under the supervision of Prof. T. Li and Prof. Z. Sun.



Tao Li is the associate professor of the College of Computer Science and Technology, National University of Defense Technology (NUDT). He received Ph.D's degree in computer science from NUDT, in 2010. His research focuses on high-performance network chips and systems. He has led and participated in more than ten national major projects. He has been in charge of the development of five specialized network chips.

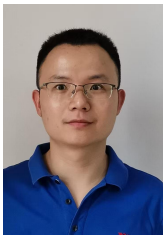
Wenwen Fu received Ph.D. degree in Computer Science and Technology from National University of Defense Technology, Changsha, China, in 2022. His main research interest is Time-Sensitive Networking.



chenglong Li received Ph.D. degree in Computer Science and Technology from National University of Defense Technology, Changsha, Hunan, China, in 2024. His research interests include time-sensitive networking and RISC-V.

Zhuochen Fan received his Ph.D. degree in computer science from Peking University in 2023. He is currently an Associate Researcher with Pengcheng Laboratory. Before joining Pengcheng Laboratory, he was a Boya Post-Doctoral Research 0Fellow at Peking University. His research interests include approximation algorithms in security, computer networks, and databases.

Hui Yang received Ph.D. degree in Electronic Science and Technology from National University of Defense Technology, Changsha, Hunan, China, in 2016. Her main research interests include RDMA and computer architecture.



Chao Zhuo received bachelor's degree in Computer Science and Technology from Changsha University of Science and Technology, Changsha, Hunan, China, in 2012. His main research interest is time-sensitive network.

Lun Li received bachelor's degree in Electronic Science and Technology from Xi'an Jiaotong University, Xi'an, Shaanxi, China, in 2020. His main research interests include RDMA NIC and drivers.

Zhiting Xiong received master's degree in Hunan University, Changsha, Hunan, China, in 2012. His main research interest is time-sensitive network.

Junnan Li is an Assistant Researcher with the Sixty-Third research institute, National University of Defense Technology, Nanjing, China, with a research focus on high-performance network processor and systems. He received Ph.D's degree in computer science from National University of Defense Technology in 2020. He has participated in national major projects, including the 863 Program, Key Research and Development Plan. He has published over ten research papers, and has authored one monographs.

.