Future Generation Computer Systems 172 (2025) 107864

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

X-DINC: Toward Cross-Layer ApproXimation for the Distributed and In-Network ACceleration of Multi-Kernel Applications

Zahra Ebrahimi [a,b,*,1], Maryam Eslami [b,1], Xun Xiao [c], Akash Kumar [b]

[a] Center for Advancing Electronics Dresden (cfaed), Technische Universität Dresden, Dresden, Germany
[b] Chair for Embedded Systems, Ruhr-Universität Bochum, Bochum, Germany
[c] Munich Research Center, Huawei Technologies, Munich, Germany

ARTICLE INFO

Keywords:
In-network computing (INC)
Distributed computing
Cross-layer approximation
Blind source separation
Independent component analysis
Field programmable gate array (FPGA)
Programmable networks
P4
Energy-efficiency
sustainability

ABSTRACT

With the rapid evolution of programmable network devices and the urge for energy-efficient and sustainable computing, network infrastructures are mutating toward a computing pipeline, providing In-Network Computing (INC) capability. Despite the initial success in offloading single/small kernels to the network devices, deploying multi-kernel applications remains challenging due to limited memory, computing resources, and lack of support for Floating Point (FP) and complex operations. To tackle these challenges, we present a cross-layer approximation and distribution methodology (X-DINC) that exploits the error resilience of applications. X-DINC utilizes a chain of techniques to facilitate kernel deployment and distribution across heterogeneous devices in INC environments. First, we identify approximation and optimization opportunities in data acquisition and computation phases of multi-kernel applications. Second, we simplify complex arithmetic operations to cope with the *computation* limitations of the programmable network switches. Third, we perform application-level sensitivity analysis to measure the trade-off between performance gain and Quality of Results (QoR) loss when approximating individual kernels via various techniques. Finally, a greedy heuristic swiftly generates Pareto/near-Pareto mixed-precision configurations that maximize the performance gain while maintaining the user-defined QoR. X-DINC is prototyped on a Virtex-7 Field Programmable Gate Array (FPGA) and evaluated using the Blind Source Separation (BSS) application on industrial audio dataset. Results show that X-DINC performs separation up to 35% faster with up to 88% lower Area-Delay Product (ADP) compared to an *Accurate-Centralized* approach, when distributed across 2 to 7 network nodes, while maintaining audio quality within an acceptable range of 15–20 dB.

1. Introduction

In the era of 5G and forthcoming 6G, a staggering amount of data needs to be processed by computation-intensive and latency-sensitive applications, e.g., video conferencing, speech recognition, and e-healthcare. However, real-time processing at such a high data rate faces substantial challenges, for which cloud computing will no longer be a promising and sustainable solution due to the following three main aspects. (1) *Response Time*: data transmission itself takes a significant portion of the total execution time in many cloud-based applications (e.g., more than 70% in some Facebook MapReduce jobs [1]). Clearly, the projected traffic explosion in the upcoming 6G can exacerbate the response time in such latency-sensitive applications. (2) *Processing Speed*: the *general-purpose* processors in cloud facilities (at most 50 Gbps in Amazon) operate at a much slower speed than programmable switches (12–50 Tbps) [2]. This processing speed might be inadequate for offloading prompt critical decision-making actions in the cloud. (3) *Energy-Efficiency and Sustainability*: a huge amount of energy, accounting for up to 50% in many cases [3], is consumed solely for transmitting data from endpoints to the cloud, significantly contributing to increased carbon emissions and environmental footprint. In fact, the electricity costs, solely to *execute* the applications in the cloud has already exceeded the cost of purchasing hardware for the whole data center [4]. These issues have raised the quest for processing data on-the-fly while being transmitted within the network, dubbed as INC [5].

Although INC is still in its infancy, early research works [2,6,7] have shown remarkable advantages by offloading single-kernel applications to the network. In particular, by bringing the processing to the proximity of data, INC has shown to reduce the *network traffic* up to 90% for specific tasks [5]. Moreover, application's *response time* and/or *energy* have been also reduced thanks to INC, compared to the centralized

* Corresponding author at: Center for Advancing Electronics Dresden (cfaed), Technische Universität Dresden, Dresden, Germany.
E-mail addresses: zahra.ebrahimi_mamaghani@tu-dresden.de, zahra.ebrahimi@rub.de (Z. Ebrahimi).
[1] Zahra Ebrahimi and Maryam Eslami contributed equally to this article.

https://doi.org/10.1016/j.future.2025.107864
Received 28 November 2024; Received in revised form 24 February 2025; Accepted 15 April 2025
Available online 7 May 2025

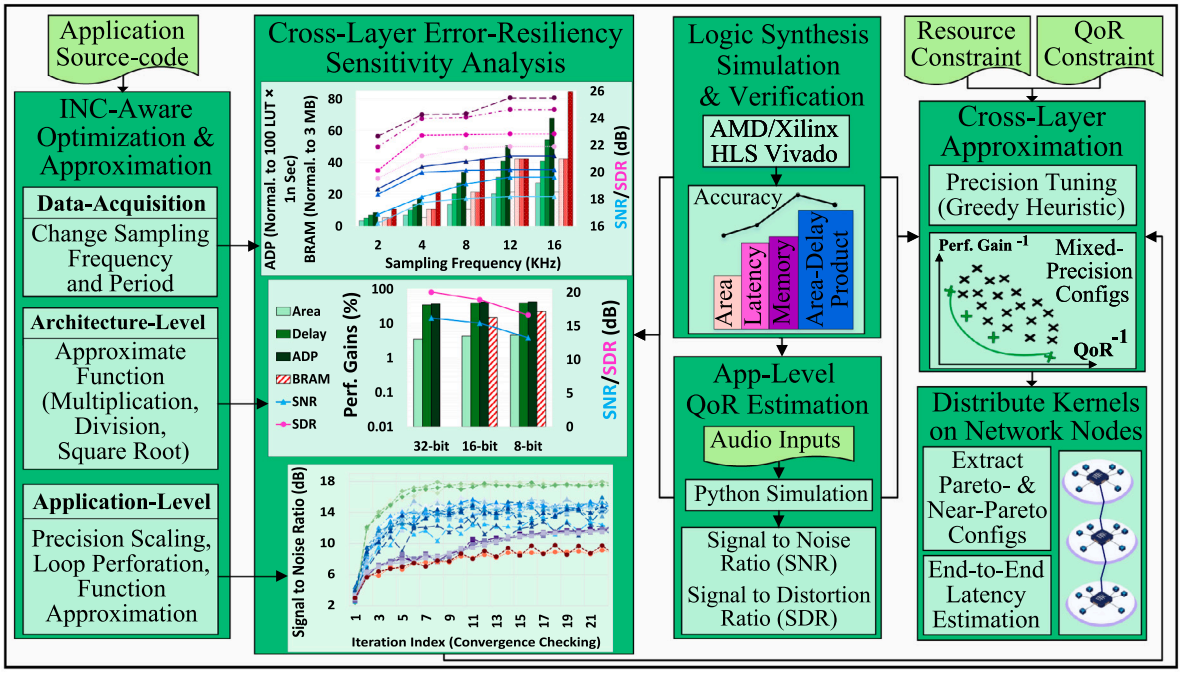

Fig. 1. X-DINC framework, prototyped on an FPGA-based testbed: overall design, approximation, and distribution methodology for an INC environment.

solutions [8,9]. For example, processing a single Neural Network (NN) layer takes 1 ms in the network versus up to 12 ms, when processed by an end-host CPU [2,10]. In fact, since the intermediate switches are part of the network, most of such costs are already paid by the packet processing/forwarding.

However, while this momentum is inspiring, the adoption of INC is currently limited to a few single-kernel applications. Applying INC for multi-kernel applications is even less due to following reasons. The first reason is the limited memory and computing resources, especially in Application Specific Integrated Circuit (ASIC)-based programmable network devices [11]. The second reason is that ASIC switches are still incapable of performing FP and complex arithmetic operations [12, 13], which are often needed by complex applications. Although such shortcomings are less pronounced in FPGA-based network elements, the performance gap between ASIC- and FPGA-based implementations, for *some* applications, may still limit the gains that can be achieved through in-network acceleration. Such challenges have therefore hindered the INC being applied to complex tasks, e.g., running Artificial Intelligence (AI)/Machine Learning (ML) inference tasks based on INC in 5G and the coming 6G networks [14,15]. Hence, to fully unleash the potential of INC, the *computing* approaches for executing multi-kernel applications should be rethought.

To address this challenge, alongside tackling energy and sustainability concerns while enabling real-time processing, prior research has demonstrated promising outcomes by employing various *approximation* techniques. These methods strategically trade off accuracy to achieve significant performance gains, offering a balanced solution for energy-efficient and sustainable computing. Those explored techniques targeted to approximate *data type* (e.g., FP to fixed-point or integer [12,16]), *basic primitives* (multiplication [17], logarithm [18], and trigonometric functions [19]), or *application kernels* (matrix multiplication/division [7,14,20], edge-detection filter [21], and quantizing specific NN layers [10,22–25]). Despite the initial efforts of these works, their approaches have been limited to approximate *single* or *small-size* kernels with *one* particular technique, rather than combining them. As a result, not only their achieved performance gains have been limited, but also the scalability of their approach is not assessed on larger case studies. Although combining multiple techniques seems straightforward, figuring out a proper combination of multiple techniques in consecutive kernels applications is not a trivial task (w.r.t. the error-propagation). These issues raise the quest for an error-aware approximation methodology that utilize the synergistic effects of multiple techniques to effectively convert larger applications enjoying INC-acceleration.

To address this question, we present X-DINC (illustrated in Fig. 1), a cross-layer approximation methodology that achieves significant performance gains by leveraging the error-resiliency of applications. Specifically, X-DINC first identifies a potential chain of approximation and optimization opportunities (when the application is executed within an INC environment). Afterwards, a sensitivity analysis reveals the error-resiliency of individual application kernels to various approximation techniques. Furthermore, by adopting a greedy heuristic algorithm, the approximation knobs in the consecutive kernels are adjusted in a way to maximize the performance-gain while maintaining the QoR at an admissible threshold. The heuristic is able to rapidly generate the optimal or near-optimal accelerator configurations, each of which enables a different trade-off between QoR and performance. At the end, w.r.t. a desired accuracy threshold and number of network nodes, X-DINC selects the optimal or near-optimal configuration of the approximated kernels, resulting in the minimum response time for the application. In short, this article makes the following technical contributions:

- X-DINC modifies the computational structure of application to tailor it for an INC approach (by approximating the implementation of required, yet unsupported complex arithmetic operations in programmable network switches).
- X-DINC identifies a proper chain of cross-layer approximation and optimization opportunities in both data acquisition and computation phases of an application, when it is executed within an INC setup.
- By applying an error-sensitivity analysis and a greedy heuristic, X-DINC adjusts the degree of approximations in multiple techniques, in a way to maximize performance-gain while maintaining an acceptable QoR. The Pareto/near-Pareto mixed-precision configurations of application kernels, swiftly generated by X-DINC, are then distributed over (a chain of) network nodes in a way to minimize transmission costs.

- As a proof-of-concept, X-DINC is evaluated on the widely-used BSS application through an end-to-end implementation, prototyped on an FPGA-based testbed. Compared to an accurate-centralized approach, X-DINC performs the separation job significantly faster with less Area-Delay Product (ADP), when the task is distributed over various nodes.
- *Open-source contribution*: to fuel further research on the in-network/distributed acceleration of multi-kernel applications, our implementations will be available at https://cfaed.tu-dresden.de/pd-downloads.

To the best of our knowledge, this is the first work that attempts to enable the in-network acceleration of multi-kernel applications with the goal of minimizing response time (while guaranteeing an acceptable QoR). The rest of this article is organized as follows: Section 2 presents a brief survey of related works (the background knowledge for this manuscript is presented in Appendix). Section 3 presents the error-resiliency sensitivity analysis of X-DINC. Afterwards, Section 4 elaborates the proposed cross-layer approximation and distribution methodology for an INC setting. Section 5 details the experimental setup and results and the remarks and challenges are discussed in Section 6. Finally, Section 7 concludes the paper with an outlook to the future tracks.

2. Related work

2.1. In-network (distributed) computing

INC: studies in this stream of research targeted the deployment of *individual* kernels of applications to the network. The investigated domains can be summarized in ML [7], data aggregation [26,27], network management [28,29], and edge detection for image processing [21]. In particular, the ML studies have shown promising results for tree-based classification models such as Decision Tree (DT) and Random Forest (RF) or less complicated models such as Support Vector Machine (SVM), and K-Means [30]. However, running resource-hungry models such as NNs on network switches is an arduous task and faced with practical deployment challenges, especially in ASIC-based switches [10,12,22–25]. In fact, even a heavily-quantized binary NN (BNN) with two layers of 64 and 32 neurons already exhausts the resources of an Intel Tofino switch [22].

Distributed INC: recently, some works have attempted to partition applications into multiple kernels and distribute them over a chain of heterogeneous nodes. In this article, a *single* kernel is defined as a combination of operations performing a specific task (e.g., filtering, transformations, feature extraction, encoding/decoding, matrix multiplications, convolutions). A *multi-kernel* application is built from multiple single kernels, which may execute sequentially or in parallel. ClickINC [31] attempts to reduce the search space for the task placement problem by (i) grouping functions into blocks to be mapped together and (ii) grouping the heterogeneous platforms (switches, FPGAs, smart Network Inference Card (NIC)s, etc.) into fewer classes to reduce the complexity of distribution problem. Flightplan [32] disaggregates P4[2] kernels into segments *manually* and then combines graph-based and formal methods to solve the distribution problem with exhaustive search, for finding a near-optimal solution. To reduce the search time, DINC [34] adopts a Multi-objective Integer Linear Programming (ILP) optimization strategy, where the objective function is a weighted linear combination of the execution latency (which disregards the transmission latency) and the resource consumption required for *executing* the task. Finally, Hermes [35] targets minimizing the communication overheads in such inter-switch coordination scenarios by considering the per-packet byte overhead along with the per-packet transmission latency. For solving this multi-objective optimization, the authors formulate a Mixed-ILP problem and adopt a greedy-based heuristic. Nevertheless, they still neglect the error-resiliency potential of applications in their analysis.

[2] P4 [33] is the de-facto language for programming network data plane.

2.2. Compatibility of INC approaches with commodity hardware

In FPGA-based network devices, computations are performed via Look-up Table (LUT)s. In ASIC, computations are carried through applying a chain of match-actions, implemented via Match-Action Tables (M/A or MATs) and Arithmetic Logic Unit (ALU)s [36]. Despite the higher packet-processing speed and power-efficiency of ASIC switches compared to FPGA counterparts, ASICs still suffer from multiple shortcomings. For example, the lack of support for *computing* operations (e.g., square root and multiplication/division of arbitrary operands [13], exhaustively used in e.g., ML and matrix-multiplication based applications). Moreover, the fixed number of pipeline stages along with the limited on-switch memory capacity in a typical ASIC switch (a few 10 s of MB of SRAM [11]) leads to a narrow range of custom functions that can be implemented via M/A tables [12]. These deficiencies along with the expensive hardware update in ASIC devices restrict to deploy complex algorithms to them [14,15].

In fact, major telecommunication enterprises and cloud service providers already rely on broad utilization of FPGA in the edge-to-cloud continuum, including switches, routers, base stations, NICs, and Network Processing Unit (NPU)s [37–39]. In particular, FPGA is used as a co-processor to accelerate various tasks, from control/traffic management and load balancing to co-processing statistics, and cryptography/security analysis [40,41]. The vast deployment of FPGA is actually endowed to their multiple advantages over ASICs: (1) FPGAs are not only well suited for packet processing with different protocols [42], but also enjoy a post-fabrication hardware data-path versatility along with a faster prototyping and lower Non-Recurring Engineering (NRE) cost. (2) Through enabling high parallelism, FPGAs are suitable hardware platforms for the implementation of e.g., matrix-multiplication based applications. Such a feature enabled up to an order-of-magnitude (or more) improvement in throughput and/or latency for a variety of in-network accelerated tasks. (3) Although integrating an FPGA as the co-processor for the switches may increase the power consumption of a *single* switch, the overall energy or *performance per Watt* will be enhanced for certain application kernels [8,41,43–46], when compared to network-deployed CPUs, GPUs, and General Purpose Processor (GPP)s. These reasons encouraged the in-network acceleration with FPGA for many tasks [40,47].

2.3. Approximation for INC

To cope with the limitations in ASIC switches and in general, P4 language, some works have adopted approximation techniques to reduce the complexity of computations. These works are classified and discussed below.

Approximation of data type: the lack of support for FP operations in ASIC switches and P4 forced the designers to use integer or fixed-point precision [12,16]. However, (in contrast to the proposed X-DINC) none of existing works have yet explored the effect of a mixed-precision strategy for the in-network acceleration of multi-kennel applications.

Approximation of primitives/individual kernels: the widely-used primitives (e.g., multiplication [7,14,20], logarithm [18], and trigonometric functions [19]) were simplified either using shift/addition operations or dedicated LUTs/MATs. However, such approximations are applied arbitrarily to applications whenever possible without performing a systematic error-sensitivity analysis (in contrast to X-DINC methodology). Therefore, the applications' ultimate QoR was sacrificed up to 10%, only by adopting a single technique [17]. Also, storing the approximate values in dedicated memories is not feasible for more complex functions.

Approximation of multi-kernel applications: in the context of INC, cutting-edge works have applied binary quantization (as a precision scaling technique) on NNs, by substituting the multiplication operation with XNOR. Beside noticeable accuracy drop, a practical deployment was possible only for a BNN having 3 layers and on modern SmartNICs such as N3IC (which benefit from more resources compared to ASIC switches) [48]. Moreover, such devices are physically deployed, only at specific locations within the network.

Approximate sampling: studies in this track have adopted either a random or stratified sampling approach (which divides the signal to segments and picks multiple samples from each group). For example, StreamApprox [49] proposed an adaptive sampling algorithm that performs the computation over disjoint sub-streams coming from various sources. However, no study explored the effect of reducing samples on the accuracy-performance trade-off of an INC accelerated program. In X-DINC, we investigate this by pursuing a *uniform* stratified approach to keep a homogeneous proportionality of samples, from the whole dataset.

2.4. INC -accelerated BSS

Aside from ML, most of INC works targeted the in-network acceleration of BSS problem[3] [50–57], due to two reasons. (1) BSS is a common task in various signal processing domains from audio to image, and bio-signals [58]. Specifically, acoustic BSS is a crucial step for speech recognition, natural language processing, voice assistants, etc. Moreover, with the emergence of smart factory within the era of 5G, acoustic BSS also plays a pivotal role in quality control and the automated supervision of production line (to detect the malfunction in faulty products at early stages, prevent the product failure, and decrease the downtime and maintenance costs, dubbed as predictive maintenance). (2) Performing controlling tasks such as BSS at real-time is paramount in latency-critical applications (especially anomaly detection and decision-making) so that the necessary responses can be triggered prior to deterioration. However, it has been shown that the performance of centralized cloud-deployed BSS is bounded by data collection and transmission time. The long data transfer time may delay the prompt actions in mission-critical services in the upcoming 6G era [51]. Hence, performing BSS at real-time is highly desired. Tackling these issues, recent studies have attempted to accelerate this application by offloading it to the network, to reduce its Mean Ttime To Respond (MTTR).

Solving BSS problem in those studies is either via a NN-based [52–54] or an algorithmic-based approach [55–57]. For NN-based approach, their key drawbacks hindering practical deployment in an INC setup are: (1) the adopted models such as ResNet and MobileNet are resource-intensive and hence, cannot be easily accelerated on the resource-constrained network nodes; and there exists many complex operations in the structure of BSS application that are not directly supported in P4 (see Section 3.1). These issues have been neglected in all previous studies. (2) It is hard to obtain sufficient labeled audio data for model training [55,59]. (3) Updating and re-training the NN models is costly, especially after being deployed on network nodes.

Algorithmic-based approaches (e.g., Fast Independent Component Analysis (ICA) [60] implemented based on the ICA algorithm) provide an early separation result and then *progressively* optimize its accuracy by iteratively increasing samples over a chain of hops along the network forwarding path. Although these works realized the drawbacks of the NN-based approaches and looked for alternative decentralized/INC-oriented strategy, they still suffer from four shortcomings: (1) if the threshold for the gradient of improvement over consecutive hops is too loose, the construction of the separation matrix may terminate prematurely; if it is too tight, it may never terminate [61]. (2) The complexity of the algorithmic solutions is still high. For example, those algorithms adopted FP precision for their computations, which not only is unsupported in P4, but also is costly in terms of resource-footprint (see Section 3.1). (3) The computation time for such high-precision solutions is still several orders of magnitude larger than the usual packet-size processing time [55]. (4) Finally, the computation itself is delayed, due to caching of the new data at each intermediate network node, the latency of which is not considered in such measurements. Overall, these issues increased the application response time and limited the performance gains from network acceleration. In fact, as also admitted by the authors in [56,61], in some extreme cases (large number of hops), the processing time would be worse than the centralized approach.

In X-DINC, we target the algorithmic approach and address aforementioned issues, through (1) tailoring the computational structure of such multi-kernel applications for an INC setup; and (2) aiming an end-to-end performance gain in a distributed implementation approach, by capitalizing on the error-resiliency of these applications and other optimization opportunities, in a network hierarchy.

3. X-DINC error-resiliency sensitivity analysis

As discussed in Sections 2.2 and 2.3, cramming multi-kernel applications into the network devices is neither straightforward nor a trivial task. In fact, an accurate and high-precision implementation may not be even possible, due to both resource-constraints and the limitation of P4 language. For example, as shown in Fig. 2, not only the area, latency, and energy of accurate multiplication, division, and square root operations are significantly larger than those of addition of the same size, but also these performance metrics grow exponentially when the operand size is increased. Therefore, to address aforementioned issues, we propose to investigate and capitalize on error-resiliency opportunities across the layers of abstraction. To this end, as also shown in Fig. 1, we identify several approximation/optimization knobs in Section 3.1, and then in Section 3.2 we present a generic methodology to measure the error-sensitivity of application kernels to various approximation techniques.

The sensitivity analysis results are then imported to the proposed approximation heuristic of X-DINC (presented in Section 4), where the goal of the algorithm is to maximize the performance gain (i.e., improvements in resource footprint, energy, or response time of application) while maintaining an acceptable QoR via operating the possible knobs. Finally, as our ultimate goal, we distribute the approximated configurations of kernels over a chain of network elements in a way to minimize the transmission latency (with which the application response time will also be minimized). This is particularly important because the existing INC approaches (cited in Section 2.4) may be impractical for a single-node deployment due to limited resource capacity, or in a distributed setting due to the transmission overheads.

In this context, BSS problem is a symbolic case study among other multi-kernel applications to showcase the efficacy of X-DINC methodology, since it inherits all the challenges discussed before and cited as a desirable target for in-network acceleration (recalling Section 2.4). In the following, we will see that a high-precision and accurate implementation of BSS is not feasible in the network nodes, due to both limited resource footprint of network nodes and inability of P4 in supporting complex arithmetic operations.

3.1. Cross-layer approximation Knobs

As listed in Table 1, we have identified various approximation/optimization opportunities in data acquisition and computation phases of the applications, executed in an INC environment. In this article, these knobs – adopted from different layers of abstraction in the

[3] BSS separates a set of signals from their mixed combination without prior knowledge about the sources or mixing process (see Appendix).

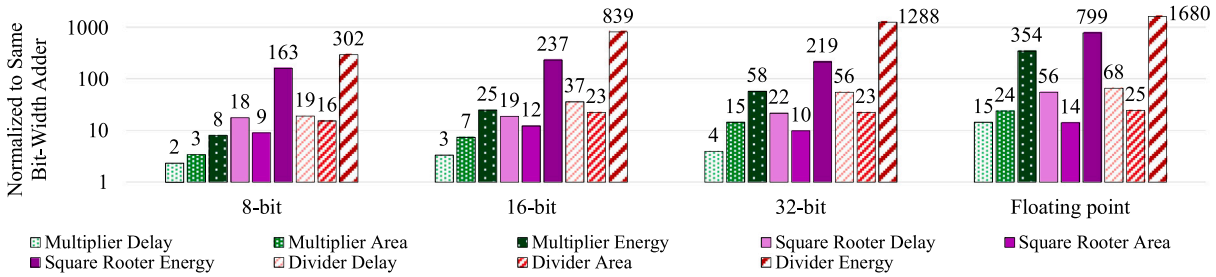

Fig. 2. Comparing area, delay, and energy of arithmetic operations in integer and FP precision (in Virtex-7 AMD/Xilix FPGA). The performance metrics of each operation is normalized to the same-size adder (only FP is normalized to 32-bit integer).

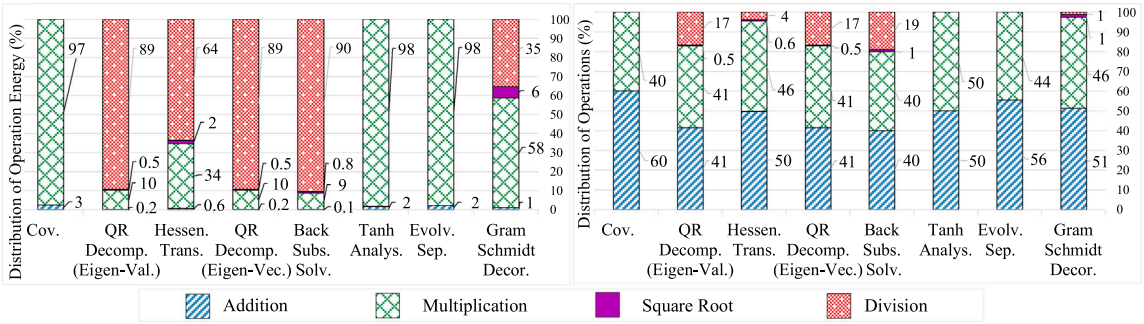

Fig. 3. Cumulative percentage and energy of each operation in the kernels of BSS application, 32-bit fixed-point implementation.

Table 1
The leveraged cross-layer approximation knobs in the sensing and computing phases of applications, for a distributed and INC-setting.

Data acquisition (Sensing)	Computation	
	Architecture-level	Application-level
Sampling period	Multiplication	Precision scaling
Sampling frequency	Division	Loop perforation
	Square root	

computing stack – target the optimization of both the pre-processing and processing kernels of the BSS application.[4]

Sampling period and frequency: analog signals around us, such as audio and bio-signals, are sampled and converted to the digital domain using an Analog to Digital Converter (ADC). These signals often contain substantial correlated or redundant data, partly due to their periodic structures. Thus, reducing the number of samples can significantly improve the processing requirement and response time of these applications [62]. In previous studies [54,55,63,64], the adopted sampling period and frequency for the implementation of audio-based BSS range from ∼1–10 s and 4–16 kHz, respectively. Therefore, as the first knob, we propose to analyze the consequences on the QoR and performance metrics of changing sampling frequency (2–16 kHz) and period (4–10 s). The result of this trade-off is presented in Section 5.2.

Architecture-level: for the second set of knobs, some basic but ubiquitously-used arithmetic operations (i.e., multiplication, division, and square root) are approximated based on techniques with low error-bias.[5] Choosing these operations targets to overcome the inability of P4 for implementing them in ASIC switches; in addition, these operations require significantly higher area, delay, and energy compared to simpler ones such as addition and multiplication (the comparison results can be found in Fig. 2). Note that we did not approximate the addition and subtraction operations because: (1) the accurate implementation of addition/subtraction is already supported in ASIC switches, (2) the cumulative energy of the operations are negligible (e.g., in BSS application), compared to the cumulative energy for aforementioned complex operations (see Fig. 3), and (3) both previous studies [67,70] and our sensitivity analysis showed that the approximation of addition in matrix-multiplication based kernels may result in high accuracy fluctuations, especially when the errors of the inexact adder are biased toward the same sign. Besides, these situations also hold true for other classification, bio-signal, and image-processing applications [66–68,71]. Sensitivity analysis of this type of knobs will be presented in Appendix.

Application-level: three techniques are considered here. First, *precision scaling* is adopted, as it is a widely-used approximation technique. It has been tested on a broad range of programs [10,12,22–24], where significant resources can be saved in both computing and memory aspects. In addition, processing at a lower precision can also reduce application's end-to-end latency, because of the shortened propagation delay of the individual (precision-reduced) operations. Moreover, the *loop perforation*, i.e., skipping loop iterations is another effective method that we applied for example on the BSS application (in Gram Schmidt decorrelation, Back substitution solver, and tan Hyperbolic – tanh – analysis kernels). Last but not least, *function approximations* applied to simplify the implementation of complex functions thus further increase the resource efficiency. For example, to approximate the tanh function, the highly-accurate implementation of Python `PiecewiseLinFit` from `pwlf` library was approximated via a piecewise linear segmentation approach using 4, 8, and 16 segments. Finally, the 8 segments has been adopted because of its negligible accuracy difference compared to the Intellectual Property (IP)-based implementation of tanh function in AMD/Xilinx Vivado [72]. Another example is the Gram–Schmidt de-correlation, which is implemented based on

[4] These knobs are not exclusive to the BSS; they exist and can be leveraged in a wide range of applications.

[5] Low error-bias plays a pivotal role in minimizing the accumulated error in consecutive kernels with an aggregation-based structure [65–69].

Scikit-learn open-source python library [73] and further modified for a resource-efficient implementation. Next, three types of optimization knobs are tested individually and jointly, to understand the (synthetic) effects to the performance of the BSS application.

3.2. End-to-end kernel-wise sensitivity analysis

A sensitivity analysis is needed to understand the consequence of adjusting multiple optimization knobs to the error resiliency of the approximated application. In fact, such an analysis is motivated by the observations in previous studies [66,67,74] which state that not only kernels contribute differently to *each of the* performance metrics (e.g., application latency, area, or memory footprint), but also after approximation, their significance to the performance gains might differ from their importance influencing the error-resiliency. Therefore, revealing the relation (i.e., trade-off) between the gained-performance and QoR-loss for individual kernels is critical to guide us to design an approximation strategy, subject to different approximation techniques.

The sensitivity analysis performed in X-DINC is inspired from [67]. However, [67] only considered the approximation of multiplication and division operations and precision scaling. We further expand it in this work in the following aspects. First, analyzing the sampling frequency and period needs to be considered in a distributed INC-setup. This is because the data is usually gathered from multiple sources (e.g., sensors) and changing data size can significantly affect the required time and/or resources for processing. Second, our analysis assessed the memory footprint, which is another constraint in a variety of network nodes but missed out in [67]. Third, additional approximation techniques were evaluated in our analysis. For example, loop perforation will significantly affect the response time of an application; and simplifying the square root operation and more complex functions are also necessary, as they are not supported in ASIC switches. Fourth, the error metrics considered in this article are expanded to Signal-to-Noise Ratio (SNR) and Signal-to-Distortion Ratio (SDR) (originally was Peak Signal-to-Noise Ratio (PSNR) in [67]). In fact, SNR and SDR not only consider the background noise, but also reflect the effects of distortion and unwanted artifacts due to both compression and various approximation [56]. Finally, the *integer to integer* precision scaling approach of [67] is changed to a *FP to fixed-point* scaling, to also consider the effect of *data type approximation*.

Our sensitivity analysis has four steps (the results are detailed in Section 5.2): ① We report the performance-accuracy trade-off for executing the whole multi-kernel BSS application at different sampling periods and frequencies and show that processing at a high data rate requires significant amount of resources (numerical comparison can be observed in Fig. 4). This hinders or exacerbates the difficulty of executing multi-kernel applications on resource-constrained network nodes. ② We replace the accurate arithmetic multiplication, division, and square root operations and tanh functions with the INC-tailored (inexact) alternatives, in all application kernels, and demonstrate their marginal effect on the final accuracy (see Fig. 5). ③ To apply the mixed-precision tuning on consecutive kernels, we performed a *kernel-wise* precision scaling investigation (as multi-kernel applications usually show a more diverse error-resiliency spectrum to the precision scaling technique [66,67,74]). To this end, the precision scaling is applied – on top of the previous techniques – on each kernel individually, while the rest of the kernels are accurate (32-bit fixed point precision with *exact* operations). The result of precision scaling analysis is also depicted in Fig. 5. ④ Finally, we analyze the effect of loop perforation (loops that are used to calculate and update the separation matrix in an iterative approach) on the accuracy of the application. Specifically, we analyze the effect of reducing the loop iteration from 200 for a highly-accurate implementation [56] to 1. The loop perforation is applied on top of previous techniques and the result shows that the application successfully converges after 35 iterations, even when it is implemented with a reduced precision and with lower sampling frequency and period (see Fig. 6). The output of this analysis is multiple lists, each belongs to a specific kernel and reports a pair of {ΔADP, ΔQoR} that reveals the end-to-end performance-accuracy trade-off, when only that kernel is approximated while the rest are accurate.

4. X-DINC cross-layer approximation and distribution methodology

4.1. Greedy-based approximation heuristic

Targeting a high accuracy, computations can be carried out in FP precision. However, as discussed earlier, reducing the precision to e.g., 16-bit has also reported to be satisfactory for many applications including BSS [75]. The main goal of the cross-layer approximation methodology is to appropriately adjust the precision of each kernel (on top of other techniques applied on the kernels) in order to maximize the performance gain while minimizing the end-to-end quality loss (or maintaining a predefined accuracy threshold). The importance of such methodology becomes pronounced for preventing the need for an exhaustive Design Space Exploration (DSE) for multi-kernel applications. In this regard, to tune the precision of consecutive kernels in a multi-kernel application, we customized and further improved the greedy-based heuristic of [67], where the pseudo-code of the modified heuristic is depicted in Algorithm 1. Note, the greedy heuristics have a lower complexity than other conventional choices such as ILP and Genetic Algorithm (GA) and faster convergence to the near-optimal solutions than simulated annealing approaches [67]. These are highly important factors for runtime decision-making actions in INC scenarios.

The inputs of the heuristic are the predefined QoR threshold and the information obtained from the sensitivity analysis, i.e., two lists L_1 and L_2 that detail the end-to-end performance-gain and QoR-loss for down-scaling the precision of the kernel to either 16- or 8-bit (while other approximations are already applied on that kernel). Each item of these lists reveals the end-to-end gain/loss only for one approximate kernel, while the rest of the kernels are 32-bit precision. The customization of the heuristic applied in this article involves reducing the number of required lists, from one list of {*performance-gain*, *QoR loss*} (for each approximation technique [67]), to only two (which shows the performance-gain and accuracy trade-off for 16- or 8-bit kernels), overall. This means that in the baseline configuration (input of the heuristic), the loop perforation, operation/function approximations and reducing sampling period/frequency are already applied. Therefore, the heuristic itself only determines the precision of the kernels.

The reason behind is two-fold and is made based on the result of sensitivity analysis. First, the precision scaling technique has shown a more pronounced effect on the kernels than other techniques (see Figs. 4 to 6). Second, pruning the dominated points in the final design space that needs to be explored is highly desirable, especially for runtime decision making in the 5G and upcoming 6G networks. In fact, in runtime scenarios, heterogeneous nodes are dynamically added or removed from the network, necessitating a quick redistribution of application kernels to the updated network nodes. Therefore, compared to the original version of the heuristic [67], we combined the techniques (which have a lower impact on the ultimate application accuracy). This has significantly reduced the size of the design space and hence, the exploration time of the heuristic.

The heuristic runs as follows: first, all application kernels (already approximated by other techniques except the precision scaling), are uniformly set to 32-bit precision. Afterwards, the *Salience List* is constructed by adding the $\frac{\Delta ADP}{\Delta QoR}$ for each pair of {kernel, precision} (i.e., the precision of that kernel is reduced to 16- or 8-bit). Note that this value is obtained in the sensitivity analysis phase. The list is then sorted in a descending order to reveal the significance order of kernels for the *combined* approximations: the kernels at the top of the list can enable a higher end-to-end performance gain while imposing less QoR loss

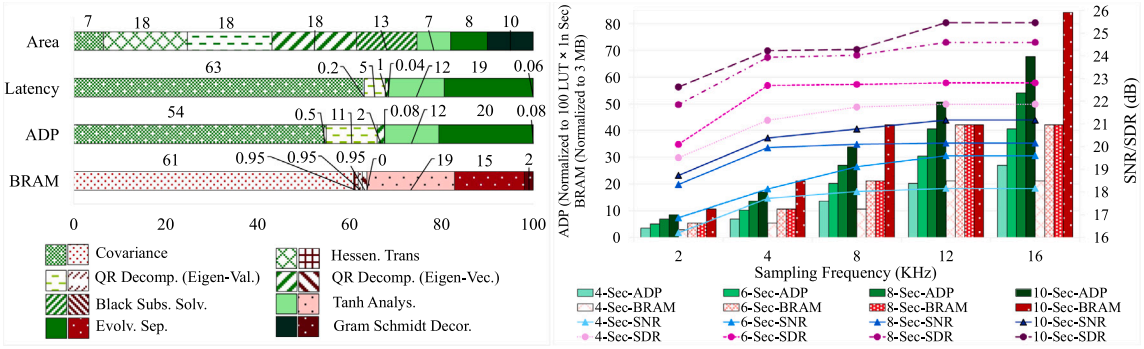

Fig. 4. The contribution of each kernel in the total performance metrics of the BSS application (left). The effect of changing sampling period and frequency on the accuracy and performance metrics of BSS application (right).

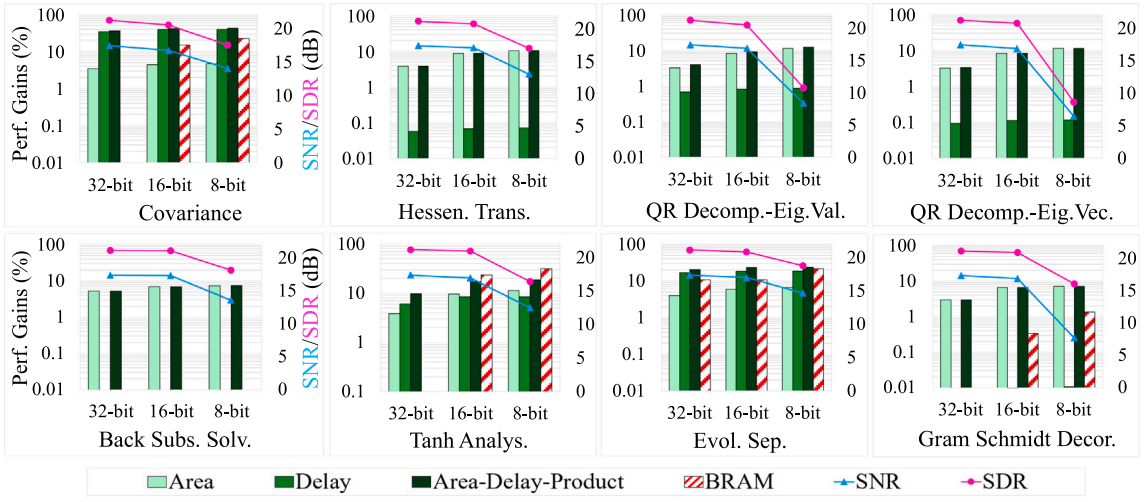

Fig. 5. Sensitivity analysis of BSS application to cross-layer approximation techniques, i.e., precision scaling on top of function approximation (multiplication, division, square root, tanh). The figures show the trade-off between the end-to-end Δperformance and ΔQoR after approximating each kernel, *individually* (while the rest are accurate FP precision).

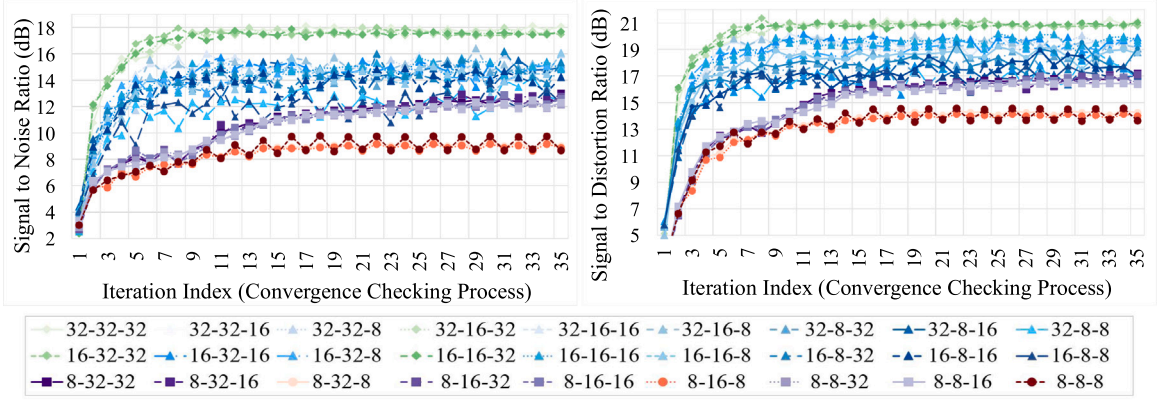

Fig. 6. The effect of loop perforation technique on the convergence and accuracy of the BSS application.

to the ultimate application accuracy (when approximated by all the techniques). Based on this list, the greedy heuristic is applied in an iterative manner: in each iteration, the pair of {kernel, precision} which appears at top of the salience list, is chosen for precision scaling. After generating a new configuration of kernels at each step of the heuristic, the configuration is evaluated on diverse samples to assess whether the final desired QoR threshold is preserved. Whenever the accuracy of the generated configuration crosses the threshold (having up to 5% difference with the user-defined threshold), the heuristic backtracks to the previous accuracy-satisfied configuration and continues the search by evaluating the next candidate at the top of the salience-list.

4.2. Distributing approximate kernels over heterogeneous network nodes

A distributed approach in INC setup entails decomposing the application into multiple partitions, each of which is assigned to a network

Algorithm 1: Greedy-Based Precision-Tuning Heuristic for Multi-Kernel Applications (customized from [67])

Input: L1: {E2E ADP Gain $_{PS}$, QoR-Loss} ∀ 16-bit Prec. Kernel
Input: L2: {E2E ADP Gain $_{PS}$, QoR-Loss} ∀ 8-bit Prec. Kernel
Input: User-Defined-QoR-Threshold, Kernel-List
Output: Approximated-Kernels [Precision]

Salience-List = Array [];
// Calculate ADP-gain of precision scaling, on each kernel, individually
for *i in Kernel-List* **do**
  // Precision of this kernel is reduced to 16-bit, others are 32-bit
  Salience-List ← L1 [i]$\frac{\Delta ADP}{\Delta QoR}$(16);
  // Precision of this kernel is reduced to 8-bit, others are 32-bit
  Salience-List ← L2 [i]$\frac{\Delta ADP}{\Delta QoR}$(8);
end
Descending Sort (Salience-List);
while *(!timeout)* **do**
  for *i in Salience-List* **do**
    // Apply approximation in descending order of $\frac{\Delta ADP}{\Delta QoR}$
    Config $_{\{\text{Approx, Prec. Reduced}\}}$ = Kernels [Salience-List$_i$];
    Output-QoR = Evaluate (Config $_{\{\text{Approx, Prec. Reduced}\}}$);
    // Also explore temporary configurations
    if *Output-QoR* ≥ *0.95* × *User-QoR-Threshold* **then**
      if *Output-QoR* ≥ *User-QoR-Threshold* **then**
        // Update candidate configuration
        Config $_{temp}$ ← Config $_{\{\text{Approx, Prec. Reduced}\}}$;
      end
      i++;
      go to 10;
    else
      Break;
    end
  end
end

device,[6] and the network devices are connected with a network topology [32,34]. For such a distributed setting, X-DINC methodology needs to distribute the N kernels across M network nodes ($M \in \{2, \dots, N-1\}$) by finding an optimal or near-optimal grouping. The goal is to minimize the transmission latency by identifying the best partitioning points that minimizes the size of the transmitted data (such goal has neither been considered nor explored in any of the related works cited in Sections 2.3 and 2.4). In this context, our approximation approach helps finding an optimal or near-optimal distribution solution: we propose to focus only on Pareto or near-Pareto configurations (generated by Algorithm 1) that not only render the highest ADP gain for a user-given QoR threshold, but also contain more kernels having the lowest precision (8-bit in this article).

Recall that Algorithm 1 tunes the precision of kernels in a way to minimize the $\{resource \times latency\}$ of the application (while satisfying the user-given QoR-level). Therefore, multiple Pareto or near-Pareto configurations are already generated, which have the minimum or near-minimum latency for a given accuracy threshold (see Fig. 7). It is worth highlighting the unique features enabled by the X-DINC, that are not supported by the existing works: through reducing the size (bit-width) of intermediate data that need to be communicated, our methodology minimizes the transmission latency (which could be the dominant portion in the end-to-end latency of the application, as discussed in Section 1). Moreover, X-DINC enables the flexibility to choose among *multiple* kernel configurations having an admissible accuracy. In fact, each of these (mixed-precision) configurations might be suited for a specific distribution scenario in an INC-setup, in which network nodes have heterogeneous resource footprints. To show the efficacy of the proposed distribution approach over the centralized counterpart, we assess various scenarios having different number of network nodes and QoR thresholds in Section 5.4.

[6] Finding the optimal re-partitioning and re-distributing application kernels w.r.t. network dynamic changes (e.g., in distributed training or inference of NNs) is an interesting track, out of the scope of this paper.

5. Results and discussion

5.1. Experimental setup

5.1.1. Application partitioning and mapping

We developed the C++ implementation of BSS application based on [56,76] and afterwards, it was synthesized with Vitis High-Level Synthesis (HLS) 2020 on a commodity FPGA (AMD/Xilinx Virtex-7 VC709). For the exact performance analysis, the Hardware Description Language (HDL)-generated design from HLS was further passed to the downstream implementation phase, placed and routed on Virtex-7 through AMD/Xilinx Vivado. In the experiments, the Digital Signal Processing (DSP) units are disabled and only the 6-LUTs are used for the synthesis. The reason behind is two-fold. First, utilizing only 6-LUTs eases the comparison of different approaches in terms of area (as estimating the required number of LUTs to be replaced with a DSP varies for each function and hence, is not a straightforward task). Second, adopting an LUT-based implementation is also recommended by many FPGA vendors, for low bit-widths operations (e.g., at 8-bit) [77]. It should be mentioned that the kernels include all the computational segments of the application and only the non-computational and critical parts (e.g., memory or loop index calculation) are exempted from the approximations. Finally, as shown in Appendix (Fig. 10), the application is partitioned into eight computational kernels and analyzed by the sensitivity analysis process.

5.1.2. Benchmark

Similar to State of the Art (SoA) works [52,54–56], we have utilized the widely-used acoustic data-set of Malfunctioning Industrial Machine Investigation and Inspection (MIMII) [78] that collects real-world data from industry machines. MIMII contains 26 092 normal and abnormal operating acoustic data from four types of machines: valves, pumps, fans, and slide rails. Every segment has a duration of 10-s audio, sampled at the frequency of 16 kHz.

5.1.3. QoR metrics

The accuracy of an approximation configuration is assessed w.r.t. different metrics with simulations. We measured SDR and SNR as the indicators for the separation quality of the audio-based BSS application. These metrics are the most widely used audio accuracy metrics in the BSS studies, comprehensively covering different types of errors [56]. SDR and SNR are defined in Eqs. (1) and (2) respectively, wherein S and $\hat{S}$ denote the original and estimated source. e_{interf}, e_{noise}, and e_{artif} represent the errors attributed from interference, noise, and artifacts, respectively.

$$SNR = 20 \sum_{i=1}^{n} \log_{10} \frac{|S_i|}{|\hat{S}_i - S_i|}, \; i = 1, 2, \dots, m \tag{1}$$

$$SDR = 10 \log_{10} \frac{\|S^2\|}{\|e_{interf} + e_{noise} + e_{artif}\|^2} \tag{2}$$

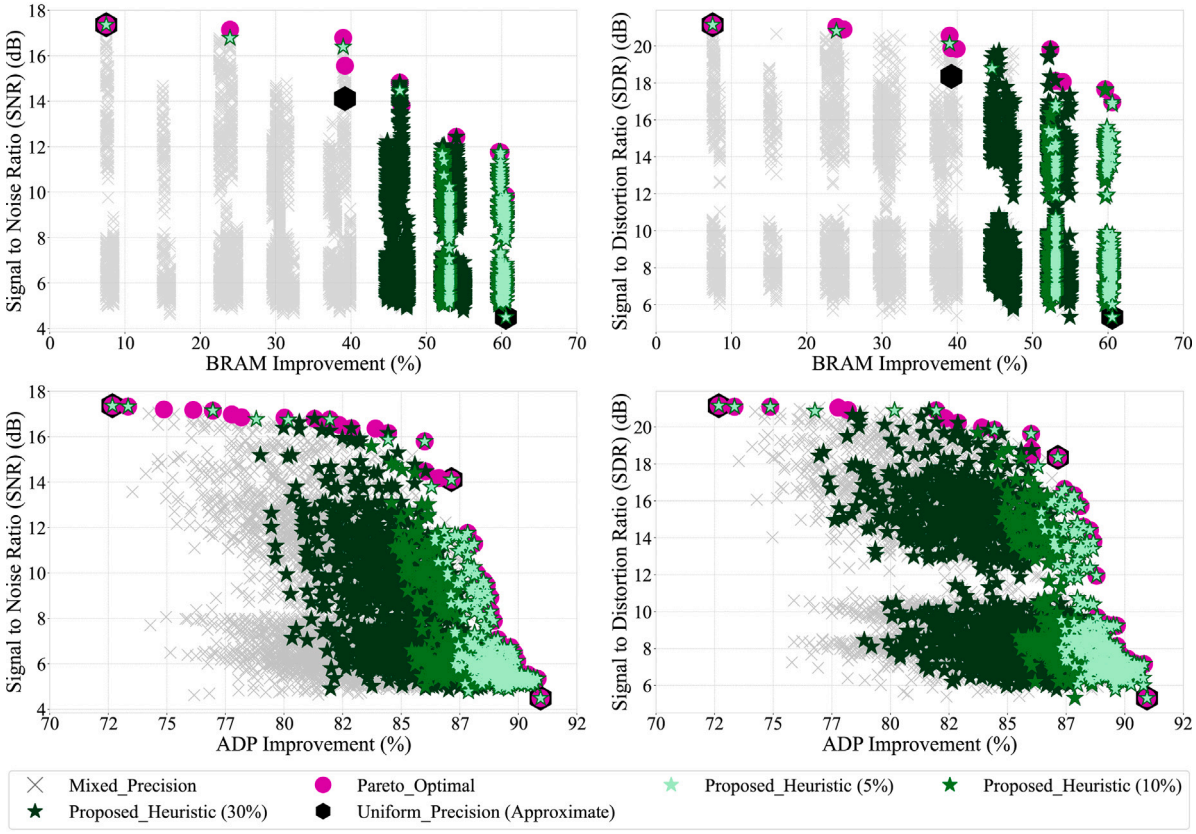

Fig. 7. Different performance-QoR trade-offs are enabled by the *approximate, mixed-precision* configurations which also achieve similar/higher gains compared to *uniform* configs. The heuristic also finds many Pareto/near-Pareto points in the 10%, 20%, and 30% time of the exhaustive search.

5.2. Cross-layer error-sensitivity analysis results

In the left-hand side of Fig. 4, we first detail the contribution of each kernel, when the BSS application is implemented in accurate mode. The evaluated metrics are the end-to-end latency (the summation of kernels' delays), 6-LUT count, and ADP as a representative of energy.[7] This figure shows that the kernels contribute differently to each of the performance metrics (e.g., application latency or energy). This information is important to be considered by the designer, when targeting a high-performance or energy-efficient design.

In the following, we present the results of sensitivity analysis. As discussed in 3.2, this process is applied in four steps. The right side of Fig. 4 demonstrates the result of the first step, i.e., the sensitivity (i.e., separation accuracy) of BSS to various sampling periods and frequencies. Afterwards, Fig. 5 reveals the low sensitivity of individual kernels to replacing the complex operations/functions in all kernels with the approximated alternatives. Moreover, this figure also shows the sensitivity of individual kernels to precision scaling, by applying this technique with a one-kernel-at-a-time strategy discussed earlier. Finally, Fig. 6 shows the effect of loop perforation (on top of other approximation techniques) on the accuracy of the application and proves the convergence of separation matrix, for all the scenarios. The key observations from the sensitivity analysis are pin-pointed herein.

- *Changing the sampling period and frequency*: as can be observed in the right-side of Fig. 4, implementing the BSS application at the high accuracy of FP precision with high sampling frequency and period (similar to [52,54–56]) is significantly resource-hungry. For example, the right-most Green bar and Red bar in this sub-figure represent the requirement of 50 860 LUTs and 84 MB memory, respectively, for implementing the BSS configuration that can support processing a 10 s audio, sampled at the frequency of 16 kHz. Such a resource-hungry implementation hinders the practical deployment of this multi-kernel application in most of FPGA- or ASIC-based network switches. In fact, only few FPGAs from Xilinx/AMD Versal, UltraScale, or Virtex have the memory capacity of higher than 30 MB. However, as can be observed in this figure, reducing the sampling frequency and period to 4-s and 4 kHz (Green and Red bars on the left side), not only reduces the required resources significantly (up to 93% reduction in memory and 42% reduction in 6-LUTs), but also maintains the accuracy at an acceptable level (>16 dB) and satisfies the Nyquist-rate sampling criteria [79]. It is worth noting that the SDR/SNR value above 15 dB is considered as an acceptable separation quality for BSS application [53,80,81]. Therefore, based on this observation we have chosen the audio period of 4 s, sampled at the frequency of 4 kHz, for the rest of the experiments.
- *Approximation of operations (multiplication, division, square root) and functions* (tanh): as can be seen in Fig. 5, replacing the exact operations with the SIMDive-based version [82] (which is P4-compatible[8]) not only can significantly improve the ADP of the kernel, but also marginally affects the accuracy in most kernels, when the precision is 32- or 16-bit (all bars with 32- or 16-bit label have a high

[7] A comprehensive and detailed energy calculation for such a large multi-kernel application is arduous and considered for the follow-up track.

[8] Further reasons for using SIMDive MUL/DIV is discussed in Appendix.

SDR/SNR accuracy of more than 16 dB). In fact, even when these operations/functions are approximated in all application kernels, the SDR (SNR) metrics are higher than 18 (14) dB (see the black hexagon markers in Fig. 7). Our analysis has shown that this negligible quality drop is endowed to the near-zero biased errors of SIMDive multiplication/division, which can cancel out each other in consecutive kernels. Moreover, as mentioned earlier, cutting-edge studies have also shown that a low error-bias error plays a pivotal role in minimizing the error accumulation in applications having multiple kernels with an aggregation-based structure (i.e., mostly addition/multiplication operations) [65–69]. Based on this observation, SIMDive-based operations have been deployed in all pre-processing and processing kernels of the BSS application. The replacement of accurate multiplication/division with SIMDive, in all kernels, has pruned the dominated points and efficiently reduced the size of design space that needs to be explored, in which only the precision of kernels is required to be changed (see Section 5.3).

In contrast to the approximation of multiplication and division, our analysis shows that truncating the output of the addition/subtraction operations (more than few bits) can significantly affect the accuracy, especially when applied in kernels such as Covariance. The high-sensitivity of addition to approximation for other matrix multiplication-based applications is also endorsed by [70]. Therefore, addition and subtraction operations are maintained accurate in our implementations. Moreover, the small contribution of these operations in the total energy of application kernels, (Fig. 2) justifies our decision that it is better to preserve the accurate structure of the these operations.

- *Kernels contribute differently to the QoR fluctuations and the gained performance*: as discussed earlier, the kernels contribute differently, to each of the performance metrics (Fig. 4 left side). Therefore, the importance order of kernels (for approximation) would be different, w.r.t. the ultimate design goal. For example, Fig. 5 shows that when a high-performance design with a shorter separation time is the goal, approximation of the *covariance* kernel results in higher latency reduction (due to its division, acting as the speed bottleneck operation). However, when resource (LUT) saving is the goal, the approximation of *Hessenberg* and *QR Decomposition* kernels is more beneficial. Moreover, this figure also reveals that the significance order of kernels in their *QoR loss* can vary from their order in the *gained performance* (after applying the same approximation technique). For example, as can be seen in Fig. 5, the most error-resilient kernel is *Evolving Separation*, while, in general, the approximation of *Covariance* can result in higher performance gain.

  Overall, aforementioned insights further corroborate our initial statement, i.e., that the relation between ΔPerformance and ΔQoR should be considered for adjusting the approximation knobs in a multi-kernel application, and not based on the primary appearance order of kernels (or their contributions), in the accurate configuration.
- *Effect of loop perforation on accuracy and convergence of separation matrix*: Fig. 6 shows that the accuracy fluctuations and convergence of the separation matrix after applying the loop perforation (on top of other techniques) on the last three kernel of BSS (which consist of loops and can be tuned to 8−, 16−, or 32−bit precision). As can be seen in this figure, reducing the number of iterations in the convergence checking to ∼25–35 not only maintains an acceptable QoR-level for many of the kernel configurations (i.e., the configurations which maintain the SDR above 15 dB), but the separation matrix also successfully converges. Skipping further and unnecessary iterations can greatly reduce the response time of the application. Therefore, we have opted to update the separation matrix until 35 times in our experiments.

The outputs of the sensitivity analysis (the lists of end-to-end {Δperformance gain, ΔQoR}) are given to the precision-tuning heuristic in the next step.

5.3. Performance gains of approximate configurations obtained by the cross-layer approximation methodology

Herein, the overall efficacy of the cross-layer approximation methodology is evaluated. Fig. 7 illustrates the accuracy and performance gain of approximate configurations over the baseline double-precision FP configuration having the accurate operations (each of the eight kernels can have the precision of 8-, 16-, or 32-bit, resulting in $3^8 = 6561$ approximate configurations). Moreover, Fig. 7 also demonstrates the efficiency of the greedy heuristic in finding the Pareto- or near-Pareto points by distinguishing the ones that are found by the heuristic, in different exploration time-limits. The results show that not only new ADP-QoR trade-off levels (mixed-precision configurations) are generated by the X-DINC methodology, but also these approximate configurations reach a higher performance gain than the uniform-precision ones. Please note that not only one unique, but a set of configurations reach permissible ADP-QoR trade-offs, which can be utilized for network nodes having different resource-footprint. For example, it is worth highlighting that the proposed X-DINC methodology has been able to generate the Pareto-optimal configuration of the BSS application which can be fit into one single FPGA (in terms of both memory and LUT) while maintaining an acceptable accuracy of >16 dB (see overlapping pink circles and green stars in e.g., subfigures at the right side of Fig. 7). X-DINC can also support dynamic scenarios in multi-hop networks and energy management routines for heterogeneous resources: arbitrary heuristic-generated configurations can be stored in FPGA SPI/BPI flash memory and loaded at runtime, enabling X-DINC to adapt to e.g., network congestion, node failures, or bandwidth changes, by switching configurations and redistributing approximated kernels.

Table 2 also details two commonly utilized metrics, i.e., the fraction of Pareto points (which are found in 5% to 50% of the exhaustive search time) and the coverage ratio for the total number of non-dominated design points and the significance of these points, known as the *hypervolume* indicator [83]. Although a heuristic algorithm may not always find all globally optimal solutions, the results show that our heuristic has been able to find more than 80% of the Pareto points, within the first 5% of the exhaustive DSE time (as shown in Table 2). This Table also asserts the efficiency of X-DINC methodology in saving the exploration time, which would be highly desired for the larger design spaces, where a brute-force search is not time-wise tractable. For example, simulating the accuracy of 6561 possible kernel configurations in C++ for BSS takes 312 h (on a Rack Server equipped with Intel Xeon E5-2667 CPU @ 3.20 GHz and 512 GB RAM), when analyzed on 25 set of 4-s audio samples from the MIMII database. Please note, many *near*-Pareto points (that are found by the heuristic) also render admissible resource-accuracy trade-off but are not reported in this Table.

We have also compared the greedy strategy of [74,84] against ours. Such approaches only considered the ΔQoR as the deciding metric in navigating the heuristic (i.e. the precision of which kernel should be reduced at each step). In contrast, our approach considers the $\frac{\Delta ADP}{\Delta QoR}$ as the salience metric. We have compared these heuristics by analyzing various accuracy levels for the BSS QoR threshold. Fig. 8 compares the ADP and memory requirement of the configurations generated by the heuristics (normalized to the respective values for the FP-based configuration). As can be seen in this figure, for most of accuracy-thresholds, our heuristic can achieve higher performance gain in terms of both ADP and memory (and also in a shorter searching time). The effectiveness of our heuristic is also even more pronounced, when the accuracy threshold is higher: as can be seen in Fig. 8, the proportional gap between the heuristics (in terms of resource savings) is higher for the higher accuracy levels. Such an efficient heuristic can therefore obviate the need of an exhaustive search (the size of the design space increases exponentially w.r.t. the number of application kernels or

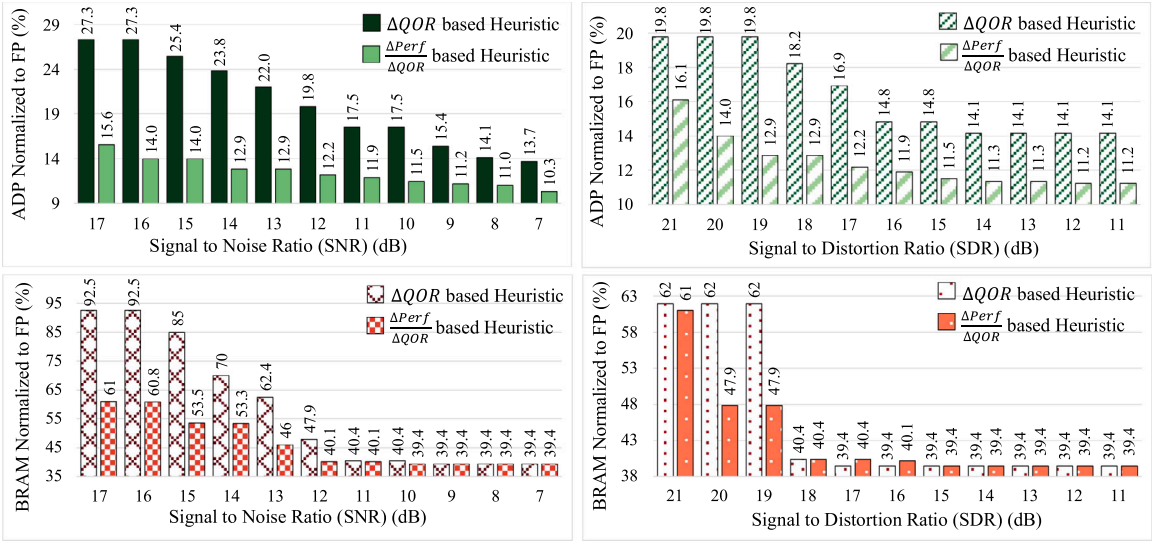

Fig. 8. Comparing the proposed $\frac{\Delta ADP}{\Delta QoR}$ based versus the ΔQoR based greedy heuristic [74,84] in various QoR levels: the former (X-DINC) finds configurations with higher performance gain and lower memory requirement.

Table 2
Percentage of 35 Pareto points, found in different exploration time of the heuristic.

Heuristic iteration (% of exhaustive search)	5	15	25	35	50
Coverage ratio of Pareto points (%)	80	85.7	85.7	85.7	88.6
Coverage ratio of Hypervolume (%)	61.8	69.8	69.8	69.8	72.5

Table 3
Grouping 8 kernels of BSS application into 2–7 nodes for distributing on network nodes in an INC scenario.

Distribution scenario	Grouping	Distribution scenario	Grouping
2-Node	{1,2},{3,4,5,6,7,8}	5-Node	{1},{2},{3},{4,5},{6,7,8}
3-Node	{1},{2},{3,4,5,6,7,8}	6-Node	{1},{2},{3},{4},{5},{6,7,8}
4-Node	{1},{2},{3,4,5},{6,7,8}	7-Node	{1},{2},{3},{4},{5},{6,7},{8}

nodes within the network) and expedite the selection of a Pareto or near-Pareto configuration at run-time and speed up decision-making actions.

5.4. End-to-end performance gains of distributed (INC) over centralized approach

For the final evaluation of proposed INC-tailored approximation methodology, we have considered various scenarios from *Accurate-Centralized*, *Accurate-Distributed (INC)*, *Approximate-Centralized*, and *Approximate-Distributed (INC)*. The experimental testbed for the distributed scenarios is established by connecting one to six hops, each is equipped with a network-connected Virtex-7 FPGA. For each distribution scenario, we have also measured the end-to-end latency of the application for six levels of QoR (i.e., SDR accuracy-levels of 14, 16, 18, 20, 21, and the FP precision as the highest accurate one). The definition of the end-to-end latency denoted as l_{e2e} of INC-accelerated scenarios for the m number of network nodes is given in Eq. (3). In this equation, t_c, t_p, t_u, and t_d represent the delays for computation, propagation, upload, and download, respectively.[9]

$$l_{\text{e2e}} = m \cdot t_c + (m-1) \cdot (t_u + t_p + t_d) \tag{3}$$

The representative values of network's communication parameters are adopted from the 5G International Mobile Telecommunications (IMT)-2020 Key Performance Indicator (KPI) standards [85] for a connected network in an industrial automation setting [86]. In this setting, the nodes are connected by links having a propagation delay of 1 ms which shows the transmission latency (t_p) between two consecutive hops. The *user experienced data rates* for downlink and uplink are also 100 and 50 mbps, respectively. Note, while t_p is constant, the other variables are computed based on the size of the data that is transmitted between consecutive hops. Fig. 9 compares the end-to-end latency of the centralized versus distributed (INC-accelerated) scenarios, for different QoR-levels. Table 3 also shows the grouping of 8 kernels into 2−7 nodes (for SDR = 20 dB). The following deductions can be inferred based on this figure:

- First, the left part of Fig. 9 compares the end-to-end latency of centralized implementations for accurate versus approximate configurations (i.e., 14 dB ≤ SDR ≤ 21 dB). Results show that the proposed approximation methodology significantly improves application's response time, compared to the accurate approach (having 76 ms latency), while QoR is also kept at an admissible threshold.
- Second, as shown in the right part of this figure, the associated costs of download, upload, and propagation, altogether are growing linearly w.r.t. the number of network links and are much less than of the computation latency. In fact, the transmission cost for 2-nodes is 2% of the total latency and it becomes at most 12% for the 7-node scenario.[10] This is attributed to the proposed approximation methodology which (1) efficiently reduces the required number of samples and (2) reduces the bit-width of the data that needs to be transmitted between the hops by applying precision scaling. This means that the data separation job can be performed faster via the in-network acceleration, through the network hops. The results also pronounce the efficacy of the proposed in-network acceleration methodology, especially when the resource footprint of one single node may not be sufficient for implementing the whole application kernels *at a high precision*: in contrast to the Accurate-Centralized FP-based implementation (which require 84 MB of memory and infeasible to be implemented in one single FPGA), all the proposed Approximate-Distributed scenarios can be practically implemented in

[9] Although packing/unpacking data is also a communication overhead, evaluation of [34] shows it is a minor cost (<2%) in the overall traffic.

[10] Communication overhead will be >50% for the 8-node scenario, as all input samples must be transferred again to the *Tanh* kernel, if separated.

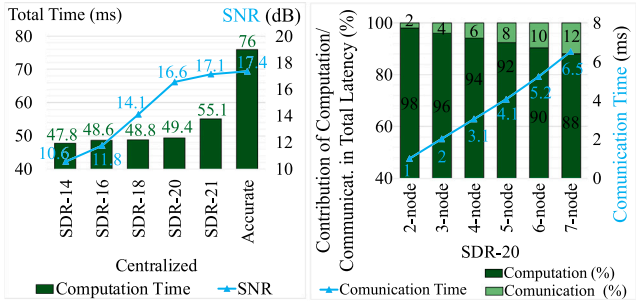

Fig. 9. The end-to-end latency (response time) of BSS application in the centralized (left) versus INC-accelerated and distributed approach (right).

one single FPGA, with the memory requirement of less than 30 MB per node.

- Note that the performance gains reported here, compared to the centralized approach, are based on distributing the kernels over a chain of FPGA-based nodes. The gains in the computation time become even more significant when the kernels are offloaded to other types of hardware, especially (ASIC) switches having orders of magnitude higher processing speed than a general purpose processor, embedded in a stand-alone host (i.e., in a centralized approach). In fact, as discussed before, besides the computation time, the proposed approximation and distribution methodology also reduces the transmission time. Therefore, such a methodology is highly desirable, especially in distributed scenarios with a higher number of network nodes, where the transmission latency might become the dominant portion of the end-to-end latency.
- The optimal number of distributed nodes depends on application structure (e.g., kernel/operation parallelizability), required end-to-end response time, computational/memory size and node resources, and kernel output data size. X-DINC can speed up such optimization by leveraging the error-resiliency of kernels, reducing the required resources and, consequently, the number of network nodes needed to implement the target application.

6. Remarks and challenges

6.1. Remarks

Scope of X-DINC: the focus of this article is narrowed to analyze the in-network acceleration of multi-kernel applications from the approximation perspective. We shown that by capitalizing on error-resiliency potentials across the abstraction layers within a network hierarchy, properly approximating and distributing application kernels over the network hops can result in a shorter (application) response time and resource requirement, when compared to the accurate-centralized approach. A further analysis on a packet-processing basis is an interesting future track, different from the scope of this article and its claimed contributions.

Applicability to heterogeneous devices and other applications: X-DINC attempts to cope with the limitations of switches and P4 by introducing and adjusting various approximation/optimization knobs, which are agnostic from the underlying architecture. For instance, our imprecise MUL/DIV/SQRT algorithms use only shift and ADD/SUB operations. Additionally, reducing processed samples and skipping loop iterations remain hardware-agnostic. Precision scaling (4, 8, 16-bit) is supported in SoA SmartNICs from vendors like NVIDIA (BlueField), Intel (Infrastructure Processing Unit (IPU)), and Netronome. While programmable ASIC switches do not natively support this, P4 enables bit-masking and shifting to emulate lower-precision operations for both storage and computation. Similarly, loop perforation is also applicable to such devices through skipping the iterations in the so-called *recirculation* process. Thus, while we prototyped our methodology on the BSS case study and on an FPGA-based testbed, the proposed approximation techniques and methodology are applicable to other multi-kernel applications and heterogeneous network devices as well. For example, many ML algorithms, especially NNs, can significantly benefit from the proposed kernel-wise sensitivity analysis and the mixed-precision strategy (for a layer-wise quantization in the scope of NNs), considering that the multiplication, division, and square root operations are also abundantly used in such algorithms.

Task scheduling and placement: While multiple configurations with different resource-accuracy trade-offs, generated by X-DINC, can aid addressing dynamic scenarios in multi-hop networks and energy management techniques for heterogeneous resources, a Software-Defined Networking (SDN) controller is responsible for such coordination and dynamic resource allocation tasks, adapting the workloads w.r.t. the network conditions, and orchestrating distributed tasks across network infrastructures [7]. Such task management strategies (briefly discussed in the studies cited in Section 2.1) are not within the scope of this paper.

6.2. Challenges

Compatibility of INC-based approach with current network setup and infrastructures: although the preliminary results in this paper and those cited in Section 2 have shown the advantages of INC, the question that how INC should be efficiently integrated in the current network setting (to minimize the additional overheads) is an ongoing research field for the 6G network providers. In fact, to the best of our knowledge, there is no existing work or simulation tool that models and compares the performance of applications when implemented in an INC setup. Going into details, utilizing the existing forwarding layer to support INC is a non-trivial job[11] that is partly investigated in [55,87]. Moreover, although there is currently no hardware platform that is specifically designed for in-network based approaches, Intel has announced that its IPU carries great potential to facilitate the INC solutions [9].

Traffic management in a heterogeneous co-design approach: coupling an FPGA to an ASIC switch in a co-design approach necessitates further implementation efforts. First, the computations should be efficiently divided to two different architectures. Second, matching and merging the traffic rates (without leading to congestion) requires meticulous attention. In fact, transferring packets between an ASIC switch and its co-processor FPGA is not straightforward, since the interconnect between them may act as a performance bottleneck. Such packet rates can be fixed or workload dependent, therefore, the switch should adapt to such dynamic scenarios. Addressing this, either all switch pipelines can connect to the FPGA via a single egress port or each pipeline can connect independently, via a dedicated port [36]. Moreover, the switch also needs to differentiate between new incoming traffic and returning packets from the FPGA [41]. Few potential solutions for such concerns are already proposed in [41] which includes bus adaptation and traffic encapsulation.

7. Conclusions and future work

This paper presented X-DINC, a cross-layer approximation and distribution methodology to enable or facilitate the in-network acceleration of multi-kernel applications, by capitalizing on the error-resiliency of the applications. X-DINC introduces various approximation and optimization knobs across the abstraction layers (within a network hierarchy) and efficiently adjust them to generate multiple optimal or near-optimal configurations with acceptable performance-QoR trade-offs. Finally, X-DINC distributes the (mixed-precision) approximate

[11] The application computation problem is always decoupled from the data-forwarding, in literature studies.

kernel configurations over a set of network nodes in a way to minimize the transmission overhead and end-to-end response time of the application (which are significantly shorter compared to the traditional accurate-centralized approach). We believe these contributions make steps towards the in-network acceleration of multi-kernel applications, as both intermediate network nodes and Internet of Things (IoT) edge devices have heterogeneous resource footprint and processing speed. Another potential use-case the X-DINC methodology could be an approximate-aware partitioning of the NN models, especially for an edge-cloud collaborative setting, which is a cutting-edge topic in the literature. In this context, utilizing the proposed approximation methodology can help to adjust the load of computation w.r.t the such requirements.

In the envisioned follow-up work, we will expand the X-DINC approximation and distribution methodology for the in-network acceleration of NNs in various distribution scenarios. Furthermore, we will present a co-design approach through which the application kernels will be efficiently grouped to be offloaded to an ASIC-switch and its co-processing FPGA. Another interesting track could be to implement a runtime coordination/management process which finds the best re-distribution scenario, that can address the dynamic changes with minimum reconfiguration overhead.

CRediT authorship contribution statement

Zahra Ebrahimi: Writing, Visualization, Review & editing, Conceptualization, Ideation, Methodology, Resources, Funding acquisition, Supervision, Project administration. **Maryam Eslami:** Methodology, Visualization. **Xun Xiao:** Supervision, Review & editing. **Akash Kumar:** Supervision, Project administration, Review & editing.

Acknowledgment

This research is co-funded by the following projects: X-DNet (BMBF, Förderkennzeichen 01|S17044, Software Campus program), X-ReAp (DFG, United States, Number 380524764) and in part by EXIGENCE (HORIZON-JU-SNS-2023, GA: 101139120).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Background and preliminaries

A.1. Blind source separation problem

Consider n sources of signals that are sampled in m time slots, the size of the data matrix is $S_{n\times m}$. Due to the mutual interference of original signal(s) and possible noises emitted by other sources, the sensed data, is inevitably a mixture of multiple sources (mixed-data matrix X). The relation between S and X can be formulated as $X = A \times S$, wherein A is the interference matrix that distorted the original signal. In reality, the coefficients of the original signal S and the interference matrix A are unknown, making the receiver *blind* to the source signal. BSS attempts to invert the mixing process and recover a near-optimal estimation of original signal S. Mathematically this can be formulated as $\widetilde{S} = A^{-1} \times X$. This is obtained by calculating the so-called *separation-matrix* W and then applying it on the mixed-signal X through $\widetilde{S} = W \times X$. BSS consists eight kernels, which are discussed herein (also depicted in Fig. 10).

First, *whitening* process is performed which decorrelates the sampled data and makes the sources as statistically-independent as possible. Whitening is applied through the five steps. First, *covariance matrix calculation* captures the statistical relationships, i.e., deriving a covariance C between different features of the data. After transforming the covariance matrix to the *Hessenberg* form (which reduces the computation complexity of the next phase), the *QR eigenvalue and eigenvector decomposition* steps break down the covariance matrix C into its constituent eigenvectors and eigenvalues ($C = E \times D \times E^T$, where E is the matrix of eigenvectors and D is the diagonal matrix of eigenvalues). The set of eigenvectors represent the directions along which the data varies the most, while the corresponding eigenvalues determine the variance along each of those eigenvector directions. Finally, *Back Substitution Solver* derives the *whitening* matrix W as ($W = D^{-\frac{1}{2}} \times E^T$) and apply it to the zero-centered input data to decorrelate the data and removes the dependencies between different data dimensions.

After whitening, the *Independent Component Analysis* process should be applied. FastICA is the most widely used approach for solving this extraction problem. FastICA algorithm iteratively enhances a randomly-initiated matrix, until convergence happens. Each iteration has three sub-steps, as follows. *Tanh Analysis* is an optimization step which improves the convergence of the algorithm through maximizing the statistical independence (or non-Gaussianity) of the data and reduces the sensitivity of separation matrix to outliers. Afterwards, *Evolving Separation* refines the estimation of the independent components by updating the separating matrix (i.e, applying a contrast function on the current separation matrix). After performing the update process, the estimated independent components may still be correlated. Therefore, the *Gram–Schmidt decorrelation* algorithm [88] is applied to ensure that the components are decorrelated, i.e., orthogonal to each other. Finally, the independent components are normalized, the weight vectors in the separation-matrix are updated, and the convergence condition is assessed (by comparing the Euclidean distance of two consecutive separation matrix).

A.2. Mitchell-based approximation of multiplication, division, and square root

As the accurate implementations of multiplication, division, and square root are not supported in P4/ASIC switches, we have utilized the approximation of SIMDive [82] due to two main reasons. First, Mitchell's algorithm translates multiplication (division) to addition (subtraction) in the logarithmic representation, as shown in Eqs. (4) and (5). Such a conversion not only facilitates a resource-efficient implementation of complex multiplication and division operations but also only utilizes simple operations (*if-else*, *addition/subtraction*, and *shift*), all of which are supported in P4 language. Hence, such Mitchell-based approximate variants can be easily adopted for implementation in a variety of P4-based network nodes. Second, the SoA SIMDive hybrid multiplier and divider unit has significantly improved the accuracy of Mitchell's algorithms to 99.2% and bound the average of absolute relative error Average of Absolute Relative Error (ARE) to <0.8%. Moreover, SIMDive enjoys a negligible error bias of less than 0.05%, a metric that is reported to play a pivotal role in minimizing the accumulated error in consecutive kernels of multi-kernel applications.

Mitchell's multiplication and division algorithms [89]: The binary representation of N-bit input A can be written as Eq. (6), wherein k shows the position of the leading one. The rest of the bits (starting from position $k-1$ to 0) are considered as the fractional part and are in the range of $0 \le x < 1$.

$$P = A \times B \xrightarrow[\log]{Approx.} \widetilde{\log_P} = \widetilde{\log_A} + \widetilde{\log_B} \xrightarrow[Anti-log]{Approx.} \tilde{P} = 2^{\widetilde{\log_P}} \tag{4}$$

$$D = A \div B \xrightarrow[log]{Approx.} \widetilde{\log_D} = \widetilde{\log_A} - \widetilde{\log_B} \xrightarrow[Anti-log]{Approx.} \tilde{D} = 2^{\widetilde{\log_D}} \tag{5}$$

$$A = 2^k + \sum_{i=0}^{k-1} 2^i b_i = 2^k(1+x) \xrightarrow{e.g.} 58 = 2^5(1+0.11010)_2, 18 = 2^4(1+0.001)_2 \tag{6}$$

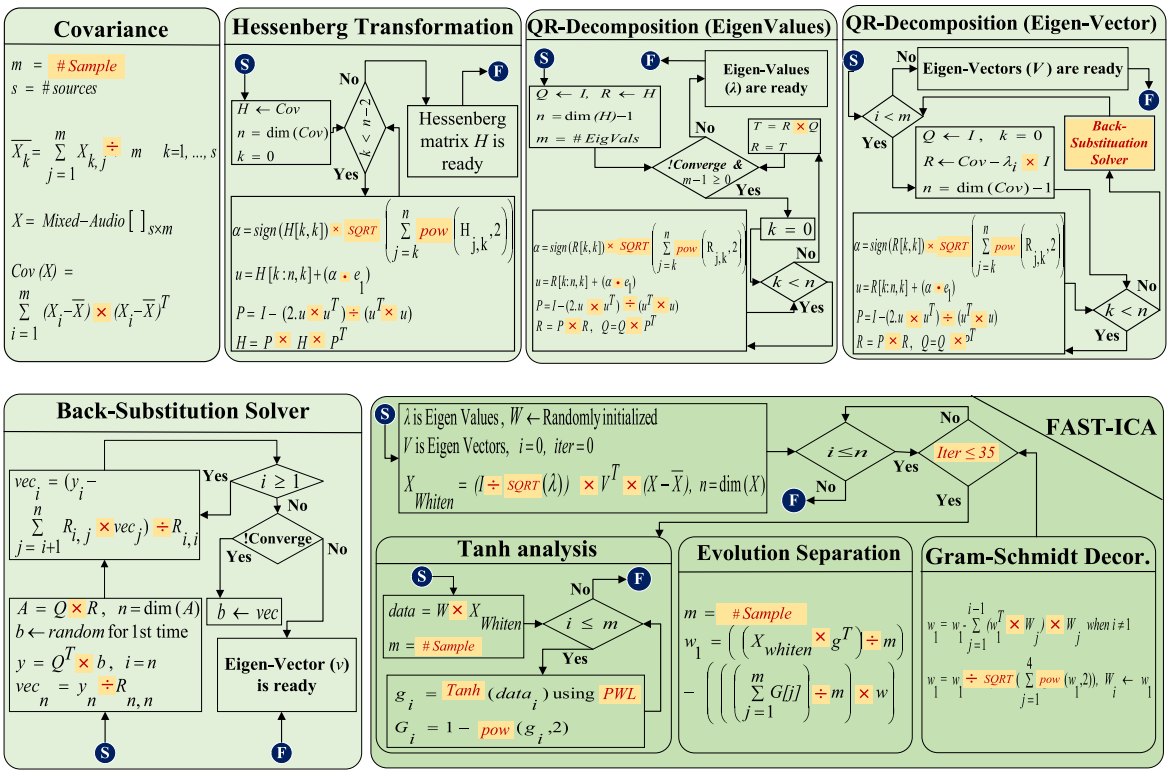

Fig. 10. The kernel structure of Blind Source Separation application (whitening and FastICA) and the detailed instruction flow in each kernel. The approximation knobs are highlighted in the figure.

In the linear approximation of log function, $\log_2(1+x)$ is approximated to x, when $0 \le x < 1$. Hence, the approximate log of A is achieved by concatenating the integer part (exponent k) and fractional part (rest of the bits, starting from position $k-1$ to 0), as shown in Eq. (7):

$$\log_2(A) \simeq k + x \rightarrow \log_2(58) \simeq (101.11010)_2, \log_2(18) \simeq (100.001)_2 \tag{7}$$

By applying the same process on the second input and obtaining its approximate log, the summation (subtraction) of two parts can is calculated by Eq. (8) (Eq. (9)).

$$\widetilde{\log_2}(\tilde{P}) = (k_1 + k_2) + (x_1 + x_2) \rightarrow K_s = (1001)_2, X_s = (0.1111)_2 \tag{8}$$

$$\widetilde{\log_2}(\tilde{D}) = (k_1 - k_2) + (x_1 - x_2) \rightarrow K_s = (1)_2, X_s = (0.1011)_2 \tag{9}$$

Lastly, the *anti-log* (which mathematically a shift operation) is applied to derive the binary representation of the approximate product (quotient), as shown in Eq. (10) (Eq. (11)):

$$\widetilde{P} = \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2), & x_1 + x_2 < 1 \\ 2^{k_1+k_2+1}(x_1 + x_2), & x_1 + x_2 \ge 1 \end{cases} \rightarrow \widetilde{P} = 992, P_{acc} = 1044 \tag{10}$$

$$\widetilde{D} = \begin{cases} 2^{k_1-k_2-1}(2 + x_1 - x_2), & x_1 - x_2 < 0 \\ 2^{k_1-k_2}(1 + x_1 - x_2), & x_1 - x_2 \ge 0 \end{cases} \rightarrow \widetilde{D} = (11)_2 = D_{acc} = 3 \tag{11}$$

Square root: this operation is implemented via non-restoring algorithm [90] (due to being resource-friendly), wherein the accurate multiplication is replaced with the SIMDive version.

Data availability

We will open-source the work (codes and data), when the article is accepted.

References

[1] M. Chowdhury, M. Zaharia, J. Ma, M.I. Jordan, I. Stoica, Managing data transfers in computer clusters with orchestra, in: ACM Conference for Special Interest Group on Data Communications, Which Specializes in the Field of Communication and Computer Networks, SIGCOMM, Association for Computing Machinery, New York, NY, USA, 2011, pp. 98–109, URL https://doi.org/10.1145/2018436.2018448.

[2] S. Kianpisheh, T. Taleb, A survey on in-network computing: Programmable data plane and technology specific applications, IEEE Commun. Surv. Tutor. 25 (1) (2023) 701–761.

[3] Cisco, Cisco annual internet report (2018–2023) white paper, 2020,

[4] N. Hu, Z. Tian, X. Du, M. Guizani, An energy-efficient in-network computing paradigm for 6G, IEEE Trans. Green Commun. Netw. (TGCN) 5 (4) (2021) 1722–1733.

[5] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, P. Kalnis, In-network computation is a dumb idea whose time has come, in: Proceedings of the 16th ACM Workshop on Hot Topics in Networks, HotNets '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 150–156, URL https://doi.org/10.1145/3152434.3152461.

[6] Microsoft, Unlocking the potential of in-network computing for telecommunication workloads, 2023, https://azure.microsoft.com/en-us/blog/unlocking-the-potential-of-in-network-computing-for-telecommunication-workloads/.

[7] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, N. Zilberman, In-network machine learning using programmable network devices: A survey, IEEE Commun. Surv. Tutor. (2023) 1–1.

[8] Y. Tokusashi, H.T. Dang, F. Pedone, R. Soulé, N. Zilberman, The case for in-network computing on demand, in: Proceedings of the Fourteenth EuroSys Conference, EuroSys '19, Association for Computing Machinery, New York, NY, USA, 2019, URL https://doi.org/10.1145/3302424.3303979.

[9] W. Jiang, H. Jiang, J. Wu, Q. Chen, Accelerating distributed cloud storage systems with in-network computing, IEEE Netw. 37 (4) (2023) 64–70.

[10] D. Sanvito, G. Siracusano, R. Bifulco, Can the network be the AI accelerator? in: Morning Workshop on in-Network Computing, NetCompute, Association for Computing Machinery, New York, NY, USA, 2018, pp. 20–25, URL https://doi.org/10.1145/3229591.3229594.

[11] Intel, Intel tofino 2 specification, 2022, https://www.intel.de/content/www/de/de/products/sku/218648/intel-tofino-2-12-8-tbps-20-stage-4-pipelines/specifications.html.

[12] Z. Xiong, N. Zilberman, Do switches dream of machine learning? Toward in-network classification, in: ACM Workshop on Hot Topics in Networks, HotNets,

Association for Computing Machinery, New York, NY, USA, 2019, pp. 25–33, URL https://doi.org/10.1145/3365609.3365864.

[13] M. Malekpourshahraki, B.E. Stephens, B. Vamanan, ADA: Arithmetic operations with adaptive TCAM population in programmable switches, in: IEEE 42nd International Conference on Distributed Computing Systems, ICDCS, 2022, pp. 1–11.

[14] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, N. Zilberman, Automating in-network machine learning, 2022, arXiv Preprint arXiv:2205.08824.

[15] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, M. Mitzenmacher, PINT: Probabilistic in-band network telemetry, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 662–680, URL https://doi.org/10.1145/3387514.3405894.

[16] S. Patel, R. Atsatsang, K.M. Tichauer, M.H.L.S. Wang, J.B. Kowalkowski, N. Sultana, In-network fractional calculations using P4 for scientific computing workloads, in: Proceedings of the 5th International Workshop on P4 in Europe, in: EuroP4 '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 33–38, URL https://doi.org/10.1145/3565475.3569083.

[17] N.K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, S. Peter, Evaluating the power of flexible packet processing for network resource allocation, in: USENIX Symposium on Networked Systems Design and Implementation, NSDI, USENIX Association, Boston, MA, 2017, pp. 67–82, URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/sharma.

[18] D. Ding, M. Savi, D. Siracusa, Estimating logarithmic and exponential functions to track network traffic entropy in P4, in: IEEE/IFIP Network Operations and Management Symposium, NOMS, 2020, pp. 1–9.

[19] I. Kunze, R. Glebke, J. Scheiper, M. Bodenbenner, R.H. Schmitt, K. Wehrle, Investigating the applicability of in-network computing to industrial scenarios, in: 4th IEEE International Conference on Industrial Cyber-Physical Systems, ICPS, 2021, pp. 334–340.

[20] C. Zheng, Z. Xiong, T.T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, N. Zilberman, Iisy: Practical in-network classification, 2022, arXiv Preprint arXiv:2205.08243.

[21] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, K. Wehrle, Towards executing computer vision functionality on programmable network devices, in: ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms, ENCP, Association for Computing Machinery, New York, NY, USA, 2019, pp. 15–20, URL https://doi.org/10.1145/3359993.3366646.

[22] G. Siracusano, R. Bifulco, In-network neural networks, 2018, arXiv Preprint arXiv:1801.05731.

[23] J. Luo, W. Liu, M. Tan, H. Chen, Binary neural network with P4 on programmable data plane, in: IEEE International Conference on Mobility, Sensing and Networking, MSN, 2022, pp. 960–965.

[24] M.C. Luizelli, R. Canofre, A.F. Lorenzon, F.D. Rossi, W. Cordeiro, O.M. Caicedo, In-network neural networks: Challenges and opportunities for innovation, IEEE Netw. 35 (6) (2021) 68–74.

[25] M. Saquetti, R. Canofre, A.F. Lorenzon, F.D. Rossi, J.R. Azambuja, W. Cordeiro, M.C. Luizelli, Toward in-network intelligence: Running distributed artificial neural networks in the data plane, IEEE Commun. Lett. 25 (11) (2021) 3551–3555.

[26] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, P. Richtarik, Scaling distributed machine learning with In-Network aggregation, in: 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 21, USENIX Association, 2021, pp. 785–808, URL https://www.usenix.org/conference/nsdi21/presentation/sapio.

[27] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, M. Swift, ATP: In-network aggregation for multi-tenant learning, in: 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 21, USENIX Association, 2021, pp. 741–761, URL https://www.usenix.org/conference/nsdi21/presentation/lao.

[28] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, I. Stoica, NetCache: Balancing key-value stores with fast in-network caching, in: ACM Symposium on Operating Systems Principles, SOSP, Association for Computing Machinery, New York, NY, USA, 2017, pp. 121–136, URL https://doi.org/10.1145/3132747.3132764.

[29] A. Lerner, R. Hussein, P. Cudré-Mauroux, The case for network accelerated query processing, in: IEEE/ACM Conference on Innovative Data Systems Research, CIDR, 2019, URL https://api.semanticscholar.org/CorpusID:58013750.

[30] B.M. Xavier, R. Silva Guimarães, G. Comarela, M. Martinello, MAP4: A pragmatic framework for in-network machine learning traffic classification, IEEE Trans. Netw. Serv. Manag. 19 (4) (2022) 4176–4188.

[31] W. Xu, Z. Zhang, Y. Feng, H. Song, Z. Chen, W. Wu, G. Liu, Y. Zhang, S. Liu, Z. Tian, B. Liu, ClickINC: In-network computing as a service in heterogeneous programmable data-center networks, in: Proceedings of the ACM SIGCOMM 2023 Conference, in: ACM SIGCOMM '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 798–815, URL https://doi.org/10.1145/3603269.3604835.

[32] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. DeHon, B.T. Loo, Flightplan: Dataplane disaggregation and placement for P4 programs, in: 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 21, USENIX Association, 2021, pp. 571–592, URL https://www.usenix.org/conference/nsdi21/presentation/sultana.

[33] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: programming protocol-independent packet processors, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95, URL https://doi.org/10.1145/2656877.2656890.

[34] C. Zheng, H. Tang, M. Zang, X. Hong, A. Feng, L. Tassiulas, N. Zilberman, DINC: Toward distributed in-network computing, Proc. ACM Netw. 1 (CoNEXT3) (2023) URL https://doi.org/10.1145/3629136.

[35] X. Chen, H. Liu, Q. Xiao, K. Guo, T. Sun, X. Ling, X. Liu, Q. Huang, D. Zhang, H. Zhou, F. Zhang, C. Wu, Toward low-overhead inter-switch coordination in network-wide data plane program deployment, in: IEEE 42nd International Conference on Distributed Computing Systems, ICDCS, 2022, pp. 370–380.

[36] N. Gebara, A. Lerner, M. Yang, M. Yu, P. Costa, M. Ghobadi, Challenging the stateless quo of programmable switches, in: Proceedings of the 19th ACM Workshop on Hot Topics in Networks, in: ACM Workshop on Hot Topics in Networks (HotNets) '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 153–159, URL https://doi.org/10.1145/3422604.3425928.

[37] C. Bobda, J.M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J.C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, R. Tessier, The future of FPGA acceleration in datacenters and the cloud, ACM Trans. Reconfigurable Technol. Syst. 15 (3) (2022) URL https://doi.org/10.1145/3506713.

[38] X. Wang, Y. Niu, F. Liu, Z. Xu, When FPGA meets cloud: A first look at performance, IEEE Trans. Cloud Comput. 10 (2) (2022) 1344–1357.

[39] C. Jin, V. Gohil, R. Karri, J. Rajendran, Security of cloud FPGAs: A survey, 2020, arXiv Preprint arXiv:2005.04867.

[40] S. Ibanez, G. Brebner, N. McKeown, N. Zilberman, The P4-netfpga workflow for line-rate packet processing, in: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–9, URL https://doi.org/10.1145/3289602.3293924.

[41] P. Bressana, N. Zilberman, D. Vucinic, R. Soulé, Trading latency for compute in the network, in: Proceedings of the Workshop on Network Application Integration/CoDesign, NAI '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 35–40, URL https://doi.org/10.1145/3405672.3405807.

[42] A. Fiessler, S. Hager, B. Scheuermann, A.W. Moore, HyPaFilter: A versatile hybrid FPGA packet filter, in: Proceedings of the Symposium on Architectures for Networking and Communications Systems, ANCS '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 25–36, URL https://doi.org/10.1145/2881025.2881033.

[43] R.A. Cooke, S.A. Fahmy, Quantifying the latency benefits of near-edge and in-network FPGA acceleration, in: Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking, EdgeSys '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 7–12, URL https://doi.org/10.1145/3378679.3394534.

[44] Y. Tokusashi, H. Matsutani, N. Zilberman, Lake: The power of in-network computing, in: International Conference on ReConFigurable Computing and FPGAs, ReConFig, 2018, pp. 1–8.

[45] S.A. Fahmy, K. Vipin, S. Shreejith, Virtualized FPGA accelerators for efficient cloud computing, in: IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom, 2015, pp. 430–435.

[46] L.R. Gobatto, P. Rodrigues, M.S.P. de Carvalho Tirone, W.L. da Costa Cordeiro, J.R.F. Azambuja, Programmable data planes meets in-network computing: A review of the state of the art and prospective directions, J. Integr. Circuits Syst. 16 (2) (2021) 1–8.

[47] N. Zilberman, Y. Audzevich, G.A. Covington, A.W. Moore, NetFPGA SUME: Toward 100 gbps as research commodity, IEEE Micro 34 (5) (2014) 32–41.

[48] A.T.-J. Akem, M. Gucciardo, M. Fiore, et al., Flowrest: Practical flow-level inference in programmable switches with random forests, in: IEEE International Conference on Computer Communications, INFOCOM, 2023.

[49] D.L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, T. Strufe, StreamApprox: Approximate computing for stream analytics, in: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Middleware '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 185–197, URL https://doi.org/10.1145/3135974.3135989.

[50] F.E.R. Cesen, L. Csikor, C. Recalde, C.E. Rothenberg, G. Pongrácz, Towards low latency industrial robot control in programmable data planes, in: IEEE Conference on Network Softwarization, NetSoft, 2020, pp. 165–169.

[51] J. Rüth, R. Glebke, K. Wehrle, V. Causevic, S. Hirche, Towards in-network industrial feedback control, in: Morning Workshop on in-Network Computing, NetCompute, Association for Computing Machinery, New York, NY, USA, 2018, pp. 14–19, URL https://doi.org/10.1145/3229591.3229592.

[52] H. Wu, J. He, M. Tömösközi, J. Zhang, F.H.P. Fitzek, You only hear once: Lightweight in-network AI design for multi-object anomaly detection, in: IEEE 21st Mediterranean Electrotechnical Conference, MELECON, 2022, pp. 1217–1222.

[53] H. Wu, J. He, M. Tömösközi, F.H. Fitzek, Y-Net: A dual path model for high accuracy blind source separation, in: IEEE Globecom Workshops, 2020, pp. 1–6.
[54] H. Wu, J. He, M. Tömösközi, Z. Xiang, F.H. Fitzek, In-network processing for low-latency industrial anomaly detection in softwarized networks, in: IEEE Global Communications Conference, GLOBECOM, 2021, pp. 01–07.
[55] H. Wu, Y. Shen, X. Xiao, G.T. Nguyen, A. Hecker, F.H.P. Fitzek, Accelerating industrial IoT acoustic data separation with in-network computing, IEEE Internet Things J. 10 (5) (2023) 3901–3916.
[56] H. Wu, Y. Shen, X. Xiao, A. Hecker, F.H. Fitzek, In-network processing acoustic data for anomaly detection in smart factory, in: IEEE Global Communications Conference, GLOBECOM, 2021, pp. 1–6.
[57] H. Wu, Z. Xiang, G.T. Nguyen, Y. Shen, F.H. Fitzek, Computing meets network: COIN-aware offloading for data-intensive blind source separation, IEEE Netw. 35 (5) (2021) 21–27.
[58] P. Comon, C. Jutten, Handbook of Blind Source Separation: Independent Component Analysis and Applications, first ed., Academic Press, Inc., USA, 2010.
[59] H. Wu, M. Tömösközi, R. Bassoli, J. Zhang, F.H. P. Fitzek, Experimental proof of the energy advantage of in-network intelligence, in: International Conference on Electrical, Computer, Communications and Mechatronics Engineering, ICECCME, 2022, pp. 1–6.
[60] A. Hyvarinen, Fast and robust fixed-point algorithms for independent component analysis, IEEE Trans. Neural Netw. 10 (3) (1999) 626–634.
[61] H. Wu, Y. Shen, J. Zhang, H. Salah, I.A. Tsokalo, F.H. Fitzek, Adaptive extraction-based independent component analysis for time-sensitive applications, in: IEEE Global Communications Conference, 2020, pp. 1–6.
[62] M.A. Razzaque, C. Bleakley, S. Dobson, Compression in wireless sensor networks: A survey and comparative evaluation, ACM Trans. Sen. Netw. 10 (1) (2013) URL https://doi.org/10.1145/2528948.
[63] C. Charoensak, F. Sattar, A single-chip FPGA design for real-time ICA-based blind source separation algorithm, in: IEEE International Symposium on Circuits and Systems, 2005, pp. 5822–5825 Vol. 6.
[64] J. He, H. Wu, X. Xiao, R. Bassoli, F.H.P. Fitzek, Functional split of in-network deep learning for 6G: A feasibility study, IEEE Wirel. Commun. 29 (5) (2022) 36–42.
[65] G. Zervakis, H. Saadat, H. Amrouch, A. Gerstlauer, S. Parameswaran, J. Henkel, Approximate computing for ML: State-of-the-art, challenges and visions, in: Asia & South Pacific Design Automation Conference, ASP-DAC, Association for Computing Machinery, New York, NY, USA, 2021, pp. 189–196, URL https://doi.org/10.1145/3394885.3431632.
[66] Z. Ebrahimi, A. Kumar, GREEN: An approximate SIMD/MIMD CGRA for energy-efficient processing at the edge, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2024) 1–1.
[67] Z. Ebrahimi, D. Klar, M.A. Ekhtiyar, A. Kumar, Plasticine: A Cross-Layer Approximation Methodology for Multi-Kernel Applications through Minimally Biased, High-Throughput, and Energy-Efficient SIMD Soft Multiplier-Divider, ACM Trans. Des. Autom. Electron. Systems (TODAES) 27 (2) (2021) URL https://doi.org/10.1145/3486616.
[68] Z. Ebrahimi, M. Zaid, M. Wijtvliet, A. Kumar, RAPID: AppRoximAte Pipelined Soft MultIpliers and Dividers for High-Throughput and Energy-Efficiency, IEEE Trans. Comput.- Aided Des. Integr. Circuits Syst. (TCAD) (2022) 1–1.
[69] Z. Ebrahimi, S. Ullah, A. Kumar, LeAp: Leading-one detection-based softcore approximate multipliers with tunable accuracy, in: Asia and South Pacific Design Automation Conference, ASP-DAC, 2020, pp. 605–610.
[70] H. Jiang, F.J.H. Santiago, H. Mo, L. Liu, J. Han, Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications, Proc. IEEE 108 (12) (2020) 2108–2135.
[71] Z. Ebrahimi, A. Kumar, BioCare: An Energy-Efficient CGRA for Bio-Signal Processing at the Edge, in: IEEE International Symposium on Circuits and Systems, ISCAS, 2021, pp. 1–5.
[72] Xilinx, Vitis high-level synthesis user guide (UG1399), 2024, https://docs.amd.com/r/en-US/ug1399-vitis-hls/HLS-Math-Library?tocId=evTCuuTJjRDKk3~TAmqNmw.
[73] P. Lafaye de Micheaux, S. van der Walt, G. Varoquaux, B. Thirion, A. Gramfort, D. A. Engemann, Scikit-learn github repository, 2024, https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/decomposition/_fastica.py.
[74] Z.-G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, J. Henkel, Weight-Oriented Approximation for Energy-Efficient Neural Network Inference Accelerators, IEEE Trans. Circuits Syst. I (TCAS- I): Regul. Pap. 67 (12) (2020) 4670–4683.
[75] U. Meyer-Baese, C. Odom, G. Botella, A. Meyer-Baese, Independent component analysis algorithm FPGA design to perform real-time blind source separation, in: Proceeding of the SPIE, Vol. 9496, 2015, pp. 180–191.
[76] PULP-Team, FastICA algorithm for PULP platform – GitHub, 2022, https://github.com/nihil21/fast-ica-pulp/tree/main.
[77] AMD, Divider generator v5.1, logicore IP product guide, 2021, https://docs.amd.com/v/u/en-US/pg151-div-gen.
[78] H. Purohit, R. Tanabe, K. Ichige, T. Endo, Y. Nikaido, K. Suefusa, Y. Kawaguchi, MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection, 2019, arXiv:1909.09347.
[79] H. Nyquist, Certain topics in telegraph transmission theory, Trans. Am. Inst. Electr. Eng. 47 (2) (1928) 617–644.
[80] Y. Zhang, Y. Zhao, Modulation domain blind speech separation in noisy environments, Speech Commun. 55 (10) (2013) 1081–1099, URL https://www.sciencedirect.com/science/article/pii/S0167639313000873.
[81] S.M. Kim, H.K. Kim, Direction-of-arrival based SNR estimation for dual-microphone speech enhancement, IEEE/ ACM Trans. Audio Speech Lang. Process. 22 (12) (2014) 2207–2217.
[82] Z. Ebrahimi, S. Ullah, A. Kumar, SIMDive: Approximate SIMD soft multiplier-divider for FPGAs with tunable accuracy, in: ACM/IEEE Great Lakes Symposium on VLSI, GLSVLSI '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 151–156, URL https://doi.org/10.1145/3386263.3406907.
[83] E. Zitzler, D. Brockhoff, L. Thiele, The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration, in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Evolutionary Multi-Criterion Optimization, 2007.
[84] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N.E. Jerger, R. Urtasun, A. Moshovos, Reduced-precision strategies for bounded memory in deep neural nets, 2015, arXiv Preprint.
[85] E. Mohyeldin, Minimum technical performance requirements for IMT-2020 radio interface (s), in: ITU-R Workshop on IMT-2020 Terrestrial Radio Interfaces, 2016, pp. 1–12.
[86] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J.J. Ramos-Munoz, J.M. Lopez-Soler, A survey on 5G usage scenarios and traffic models, IEEE Commun. Surv. Tutor. 22 (2) (2020) 905–929.
[87] I. Kunze, K. Wehrle, D. Trossen, Transport protocol issues of in-network computing systems, Internet Eng. Task Force (2020).
[88] Å. Björck, Numerics of gram-Schmidt orthogonalization, Linear Algebra Appl. 197–198 (1994) 297–316, URL https://www.sciencedirect.com/science/article/pii/0024379594904936.
[89] J.N. Mitchell, Computer Multiplication and Division using Binary Logarithms, IRE Trans. Electron. Comput. (IRETEC) 11 (4) (1962).
[90] Advanced Micro Devices - GitHub, Introductory examples for vitis HLS, 2024, https://github.com/Xilinx/Vitis-HLS-Introductory-Examples/blob/master/Modeling/fixed_point_sqrt/fxp_sqrt.h.

Zahra Ebrahimi is a PhD student at Technische Universität Dresden and a research associate at Ruhr University Bochum. She is also the manager of the BMBF project (X-DNet), from which this article is funded from. Her research interests include approximate computing, reconfigurable accelerator design, in-network and edge computing. Zahra has no financial and personal relationships with other people or organizations that could inappropriately influence or bias this work.

Maryam Eslami is a research associate at Ruhr University Bochum. Her research interests include approximate computing and fault-tolerance design. Maryam has no financial and personal relationships with other people or organizations that could inappropriately influence or bias this work.

Xun Xiao is a principal researcher in Munich Research Center, Huawei Technologies, Germany. Currently, he is a standard delegate in ISG permissioned distributed ledger (PDL) in European Telecommunication Standardization Institute (ETSI) since 2022. His research interests mainly focus on distributed algorithms, networking, and computing in distributed systems. He has no other financial and personal relationships with other people or organizations that could inappropriately influence or bias this work.

Akash Kumar was a Professor with Technische Universität Dresden, Germany. Since April 2024, he is directing the chair of Embedded Systems at Ruhr University Bochum, Germany. His research interests include the design and analysis of low-power embedded multiprocessor systems, and designing secure systems with emerging nano-technologies. He has no other financial and personal relationships with other people or organizations that could inappropriately influence or bias this work.