Accelerate On-Chip Artificial Neural Network Training: a Customised fNIRS Signals Processing System-on-Chip

1st Yunyi Zhao
Faculty of Medical Science, UCL London, UK
Email: yunyi.zhao.21@ucl.ac.uk

2nd Hubin Zhao
Faculty of Medical Science, UCL London, UK
Email: hubin.zhao@ucl.ac.uk

3th Shufan Yang
School of Mechanical Engineering, University of Leeds, Leeds, UK
Email: s.f.yang@leeds.ac.uk

***Abstract*—During artificial neural network (ANN) training, batch normalisation (BN) techniques improve training performance algorithmically but create hardware implementation challenges. This paper aims to address this specific bottleneck within the broader challenge of on-chip training. We propose an all-on-chip artificial neural network (ANN) training framework that utilises AMD's Versal XCVC1902 heterogeneous SoC with a purpose-built batch-normalisation (BN) accelerator to overcome the two main barriers to edge computing: memory bandwidth and batch normalisation layer latency. By hardwiring batch normalisation calculation to form a hardware primitive and using fixed-point arithmetic, the design improves batch normalisation latency from tens of cycles to nine cycles, improving the efficiency of the ANN training process. The hardware-software co-design strategy demonstrates how deep processing unit cores, BN engine based on FPGA fabric, and CPUs can be orchestrated through a unifying abstraction, pointing toward application-specific SoCs that hide low-level scheduling from machine learning developers. The framework is evaluated as an application of real-time functional near-infrared spectroscopy (fNIRS) signal reconstruction, where cloud off-loading is undesirable for privacy and latency reasons. Although evaluated on fNIRS, the tiled batch normalisation engine and zero-copy dataflow can be reused in any ANN training that spends a significant fraction of time in batch normalisation layers.**

***Index Terms*—fNIRS, Edge Acceleration, Hardware/Software Co-design, Optimisation, System-on-chip**

I. Introduction

There is a growing demand for portable functional Near-Infrared Spectroscopy (fNIRS) devices in clinical and research settings for real-time haemodynamic signal processing, which require low-power on-chip artificial neural network (ANN) training support. Since fNIRS signals often contain personally identifiable information [1], localised data processing reduces the risk of data leakage while enabling the fine-tuning of the personalised model, which is essential for the practical implementation of the system.

However, system-on-chip architectures present significant limitations for ANN training processes due to constrained on-chip memory resources and insufficient computational throughput to accommodate memory-intensive operations. The primary performance bottleneck stems from batch normalisation (BN) operations, which require continuous statistical parameter updates during training phases. For on-chip ANN training, normalisation layers can consume 20.7% to 60.8% of training time [2], due to the intrinsic serial computation in BN such as the calculation of mean and variance of each layer. These computations introduce significant computational overhead and elevated power consumption profiles that adversely impact deployment feasibility on on-chip architecture.

Previous acceleration solutions for on-chip implementation have predominantly relied on floating point calculations [3], which present significant limitations in energy consumption and efficiency for ANN training. Current ANN training optimisation approaches focus on two main areas: improving BN computation from floating-point to fix-point arithmetic and reducing the overhead of two-pass data transfer in BN calculation. For BN optimisation, DNNBuilder assigns accumulation logic per layer based on operational count [4], while DNNExplorer enhances initial ANN layers computation, but it is unable to apply matrix accumulation for later layers [5]. For data transfer optimisation, researchers have developed solutions such as TANGRAM to optimise data flow and reduce power consumption [6]. To minimise off-chip transfers, several methods focus on memory pattern searching to couple with the increasing network scales [7]. The hardware-software co-design paradigm enables the creation of hardware-aware data flows that improve memory access patterns [8], frequently reorganising matrix multiplications to optimise cache utilisation and data locality [9]–[12]. Binary Neural Networks (BNNs) attempted to address both optimisation challenges by quantising weights and activations to binary values, thus drastically reducing the data transfer volume. However, this extreme quantisation compromises the model's accuracy, often leading to a measurable decline in accuracy degradation compared to full-precision counterparts [13]. To the best of our knowledge, this is the first work on unite BN computation acceleration and memory transfer optimisation within a single framework, which is an essential breakthrough that removes a long-standing bottleneck in on-chip ANN training.

In this paper, we propose an efficient on-chip training frame-

work that leverages optimised matrix tiling strategies with a dedicated BN engine to eliminate computational bottlenecks via a hardware-aware abstraction layer. The principal contributions of this work are as follows. First, we present a hardware-aware abstraction layer that automatically maps convolutional layers to the on-chip deep processing unit, while routing BN and other non-convolutional kernels to customise FPGA logic, thereby enabling developers to focus on high-level neural network design without engaging with the complexities of system-on-chip (SoC) control and resource management. Second, we demonstrate the practicality and efficiency of AI-enabled SoC prototyping on the Versal XCVC1902 platform with competitive inference throughput while reducing on-chip power consumption by up to 40% compared to conventional implementations. Finally, we design and experimentally validate a real-time signal processing pipeline for functional near-infrared spectroscopy (fNIRS), fully implemented on the FPGA-SoC. The system achieves sub-millisecond latency and operates within a low-power consumption, demonstrating the effectiveness of the framework in enabling ultra-low-power, high-performance biomedical applications.

II. Background

A. Batch Normalisation

Batch normalisation (BN) is proposed to improve the accuracy and convergence speed of DNN training by [14]. Generally, it is introduced to normalise activations in a mini-batch by subtracting the batch mean and dividing by the batch standard deviation, as shown in equation 1.

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad y^{(i)} = \gamma \hat{x}^{(i)} + \beta \tag{1}$$

While batch normalisation improves convergence and allows for higher learning rates, it introduces additional memory access and data movement, which can significantly affect energy consumption.

B. Versal Adaptive SoC

Versal XCVC1902 utilise a full heterogeneous System-on-Chip (SoC)–class Adaptable Compute Acceleration Platform (ACAP) powered by the Versal ACAP architecture, which offers advanced AI, ML, and DSP acceleration with specialised hardware for real-time processing [15]. The adaptable logic and on-chip memory allow developers to quickly test and iterate designs without fabricating physical chips. The XCVC1902 includes a Processing System (PS) with a 2 GHz Dual-Core Arm Cortex-A72 for general processing, a 400 MHz Dual-Core Arm Cortex-R5F for real-time tasks, and a 1.25 GHz AI Engine for ML acceleration. It features low-latency and packet-switched fabric connections using a network-on-chip (NoC).

1) Deep Learning Unit: The Xilinx Deep Learning Unit (DPU), a configurable IP block, runs on the PL to execute convolutional neural networks in parallel. It features memory-mapped AXI interfaces for status register configuration and data access, helping to integrate into block designs. DPU core architecture can be adjusted based on performance, PL utilisation, and power specifications.

2) Xilinx Vitis AI: The Vitis AI workflow efficiently optimises, compiles, and deploys AI models on FPGA and SoC hardware. After quantising a pre-trained model using the Vitis AI Quantizer, it is compiled with the Vitis AI compiler [16] and deployed to a Xilinx hardware accelerator in PE. Although the tool provides APIs for model loading, inference, and profiling, optimising all ANN layers is challenging due to functional constraints.

C. DAE Network

The Denoising Autoencoder (DAE) artificial network consists of initial convolution layers with max-pooling, followed by four upsampling layers, and a concluding convolution layer [17]. The BN layers are used for accelerating ANN training to maintain a reasonable gradient. In this paper, the input of ANN is raw light intensity data with motion artefacts, outputting is the de-noised data. For multi-channel fNIRS signals, sliding windows of 512 points create a 512x1 input vector for signal reconstruction. The outputs are saved as CSV files for each channel, combined to reconstruct the signal, and converted back using the MNE python library for quality evaluation and calculation of the haemodynamic response function (HRF).

III. Implementation

This paper presents a unified framework utilising a hardware-aware abstraction layer as Python-based automatic mapping pipeline with runtime support for optimised model deployment. The framework's input parameters include the application model and hardware constraints, enabling automated optimisation and code generation. The parallel scheduling simultaneously implements layer splitting into convolutional and non-convolutional components; weight encoding involving weights and index Huffman encoding for compression (as shown in Fig. 1) and resizing with partitioning for task parallelism across accelerators (i.e., DPU cores and BN engines). Convolutional layers are flattened for processing as matrix multiplication. The BN engine implementation adopts hardware-aware data with fixed-point arithmetic to achieve reduced resource costs. The unified framework is shown in Fig. 1, this framework enables bidirectional data transfers between processing units and accelerators, resulting in considerable performance improvement in speed and reducing energy consumption, as shown in Fig. 1.

A. Hardware-aware Abstraction

The proposed framework provides a hardware-aware abstraction layer that front-loads platform-specific optimisations, such as intelligent memory placement and customised BN engines into the Ryzen AI workflow. The abstraction layer and the Ryzen AI runtime jointly handle SoC scheduling, power tuning, and resource arbitration. This design choice allows researchers to focus on high-level neural network exploration while enabling the speed and energy efficiency gains of tightly coupled accelerator and processing units.

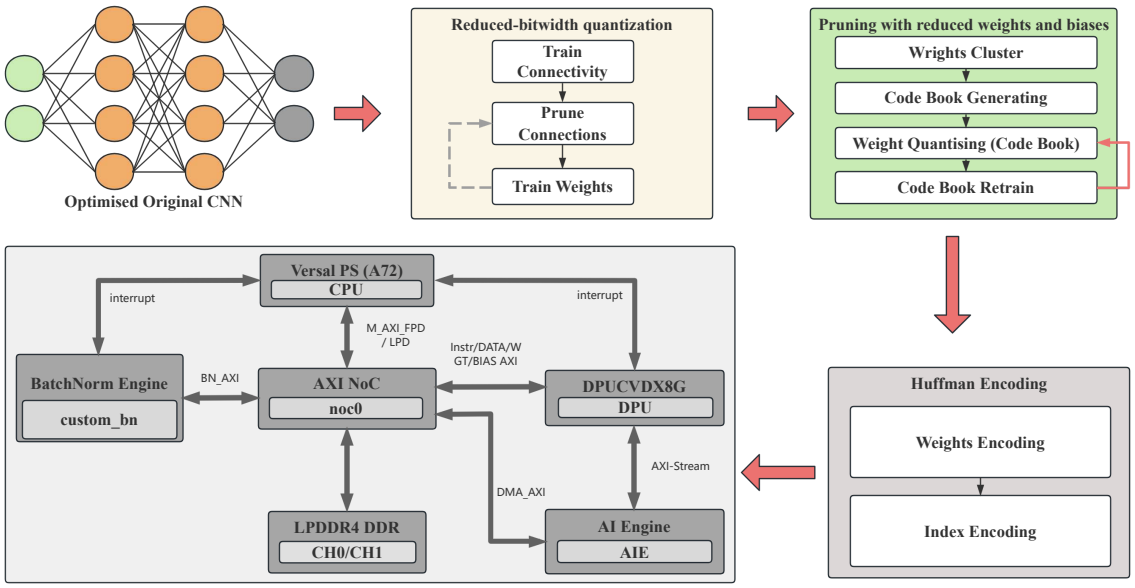

Fig. 1. Flow chart of ANN deployment

The hardware-aware layer abstraction is shown in the grey-shaded subsystem in Fig. 1. Through hardware-aware abstraction, all devices are orchestrated by AMD's Ryzen AI platform, which layers ONNX Runtime with the Vitis AI Execution Provider (VAI-EP). At compile time, VAI-EP quantises the network and decomposes the graph, routing compute-intensive kernels to the Versal convolutional processing unit while retaining control flow on the Cortex-A72 processing system. Crucially, Ryzen AI exposes zero-copy shared buffers, enabling the frequent bidirectional exchanges demanded by the framework to incur negligible transfer overhead.

B. Accelerate Batch Normalisation Engine

Instead of relying on simplified methods such as folded batch normalisation, layer fusion, or online normalisation, this work introduces a tilled engine that deploys 32 multiple BN units in parallel (as shown in Fig. 2). Fig. 2 shows the finite state machine for data transfer between the convolutional kernels and the BN engine. The optimised state machine of the BN engine costs a maximum of 9 clock cycles delay.

C. Memory Transfer Optimisation

For on-chip training, the convolutional layer and the BN layer are implemented as separate hardware modules within a unified architecture comprising the DPU and BN engine. The block diagram of the integrated framework with processing units, the DPU and the BN engine is presented in Fig. 3. Convolutional operations are executed by the DPU cores on the Versal chip. While the BN engine is performed within field programmable logic (FPGA) fabric. The pseudocode for the data orchestration process is outlined in Algorithm 1.

Overall, the design uses a customised operational method based on vectorised computation and parallel data transfer to meet the computational demands of on-chip training.

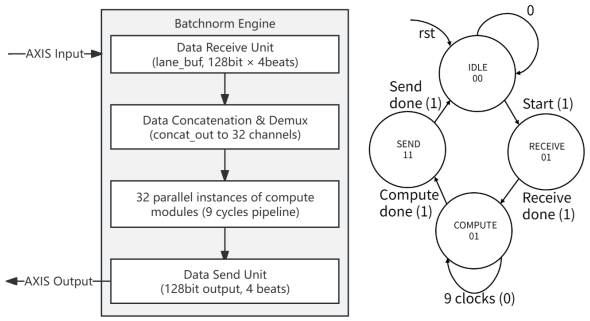

Fig. 2. Batch-norm Engine Architecture

IV. Experiments and Results

A. Test Setting

We used an AMD AI PC with an AMD Ryzen9 7940HS processor [18] and an Nvidia GB202 GPU (RTX5090) as our baseline to compare the implementation with the Versal XCVC1902. AMD Ryzen9 7940HS processor features specialised hardware with utilised Inference Processing Units (IPUs) for enhanced AI tasks. Nvidia GB202 has 575-watt power consumption with 21760 cores. Our test focuses on the energy consumption and accuracy under optimisation implementations to demonstrate on-chip network training efficiency.

The Model inference performance is used fNIRS datasets from a finger-tapping experiment (MNE toolbox [19]) and a simulated fNIRS dataset (MNE's `simulate_nirs_raw` function). The experimental data, from five subjects thumb tapping, were pre-processed with Homer2's tools [20]. The Simulated datasets with spike-like and baseline shift artefacts

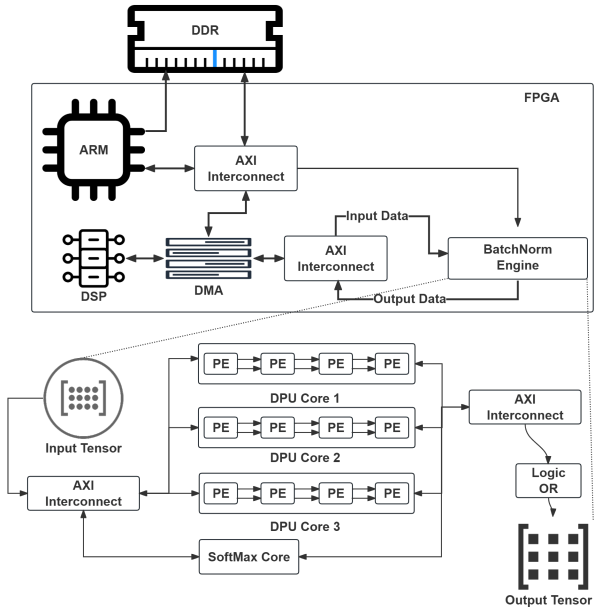

Fig. 3. Programmable Logic and Programmable System Interaction

Algorithm 1 Control Pathway for BN engine with Parallel Processing

Initialize: $L_{out} \leftarrow \emptyset$
Wait for input data: Receive 4×32 values into $lane_buf$ buffer.
Start computation: Trigger 32 parallel $bn_compute$ instances.
for each $bn_compute$ instance (1 to 32) **do**
  Perform batch normalization using:
    $N_{out} = \gamma \frac{L_{in}-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta$
    Newton-Raphson Approximation for Square Root:
    $x_0 = 1/\sqrt{\sigma^2+\epsilon}$
    $x_{n+1} = \frac{1}{2}(x_n + \frac{\sigma^2+\epsilon}{x_n})$
    (Iterate for 2 steps for sufficient precision)
end for
Wait for all $bn_compute$ **outputs to complete (9-cycle latency).**
Aggregate results: Concatenate results into a batch of 128-bit.
Store results: Write to output FIFO buffer.
Transmit results: Send results to output stream in 4 beats.
return L_{out}
=0

were added as noise using a Laplace distribution. This diverse dataset tested model robustness against noise and artefacts.

The model was developed, trained, and evaluated with PyTorch. Training occurred over 50 epochs with a batch size of 32, using the Adam optimiser with a 0.001 learning rate and MSE as the loss function. Inference used a batch size of 1. Float precision was used for Ryzen9 and Nvidia RTX5090, while INT8 precision quantisation was used for the Ryzen9 with IPU, and Versal SOC for better speed and power efficiency comparison. The DPU and the tilling engine used the Vitis-AI runtime with compiled '.xmodel' files. The ONNX runtime was also used for GPU and Ryzen9 processors to maintain test consistency.

To quantify energy consumption, we used AMD's Board Evaluation and Management (BEAM) utility, which reports hardware-level real-time power metrics. The power draw was continuously recorded for 60 s and the mean value was reported over this interval. Because BEAM measures the aggregate power of the entire device, the power attributable to the model was computed only by the difference between the average power during processing and the average power in the idle state. In addition, the board's cooling system operates a dedicated fan rated at 18 W, which runs at maximum speed during model execution; accordingly, a further 18 W was subtracted from the measured average to isolate model-specific power consumption.

B. Baseline Benchmark

In this work, CIFAR10 is adopted as a benchmark dataset [21] due to lacking standard image reconstruction benchmarks in fNIRS. The dataset includes 60,000 images in 10 categories. The architecture of DAE, UNet [22] and 3D-UNet [23] have adapted for the classification of CIFA10. The baseline benchmark is used for CIFAR10 dataset. The timing analysis result of the modified BN engine acceleration is shown in Fig 4. As illustrated in the Fig 4, the `bn_control` finite-state machine accepts a burst of four 128-bit AXI-Stream words while both `s_valid` and `s_ready` are asserted. Immediately after the last input beat, a single-cycle `start_pulse` is generated, which simultaneously enables all 32 parallel `bn_compute` instances. Because the batch-normalisation datapath is fully pipelined, its latency is constant and equal to nine clock cycles; this gap is visible between the `start_pulse` and the first assertion of `m_valid`. Once computation completes, the processed batch is streamed out in four consecutive beats on `m_tdata` whilst `m_ready` remains high, after which the controller returns to the *IDLE* state, making the datapath immediately available for the next transaction. The trace therefore confirms both the correct handshake behaviour and the fixed-latency property of the design.

TABLE I
BNN(INT8) PERFORMANCE AND POWER CONSUMPTION USING BN ENGINE METHOD ON VERSAL XCVC1902

Model	FPS	SoC Power (W)	Inference Time (ms)
DAE	4755.47	10.47	0.21
UNet	2626.58	5.56	0.38
3D-UNet	11.4406	11.37	87.41

Table I summarises the performance characteristics of three INT8-quantised networks: DAE, UNet, and 3D-UNet, running on a Versal XCVC1902 with the proposed BN engine. The first column lists the effective processing rate in frames per

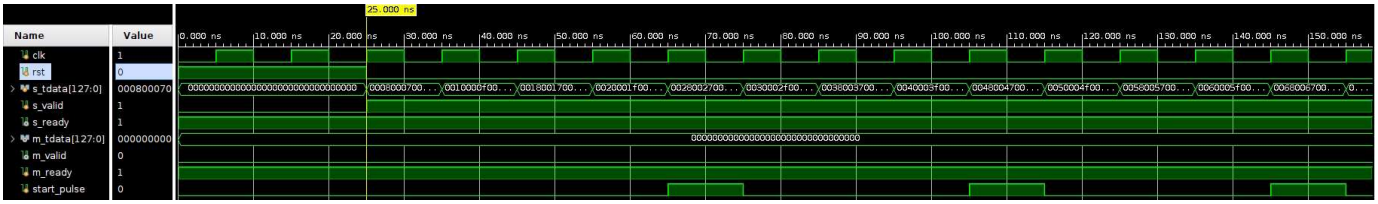

Fig. 4. Behavioural simulation of the `bn_control` module for one batch.

second (FPS), where one frame corresponds to a single CIFAR-10 image; FPS is therefore a direct measure of throughput. The average on-chip power drawn during steady-state inference is given in the second column. The final column reports the inference time per sample, which is simply the reciprocal of the FPS (time = 1000/FPS in milliseconds). Under these conditions, the DAE achieves the highest throughput at 4755FPS with an inference time of 0.21ms, while 3D-UNet, due to its much larger computational workload, operates at only 11.4FPS and 87.4ms per image despite drawing comparable power.

C. Energy Consumption Analysis

Additionally, our platform delivers the fastest inference time at 0.21 ms (as shown in Table I), significantly outperforming other similar encoder-decoder ANN architectures (i.e. UNet and 3D-UNet). Such rapid processing is vital for real-time applications. Furthermore, the platform demonstrates excellent power efficiency, requiring only a few milliwatts per image processing. This represents a considerable improvement over the UNet and 3D-UNet without a purpose-built BN engine, making it highly suitable for low-power systems, as shown in Fig. 5. Although 3D-UNet and UNet share an encoder-decoder architecture, their memory transfer mechanisms are not optimised for processing fNIRS signals. By uniting a purpose-built BN engine with optimised memory transfer, the proposed framework delivers a balanced on-chip training solution that offers significant advantages for high-performance, real-time, and resource-constrained applications.

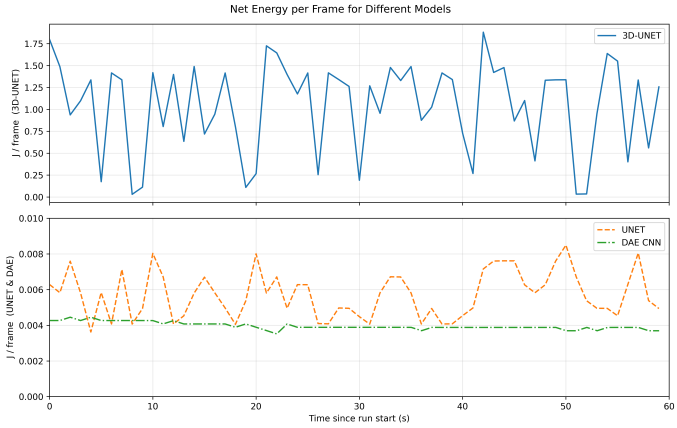

Fig. 5. Energy consumption graph measured with BEAM tool by AMD

D. fNIRS Signal Processing

The haemodynamic response function (HRF) is used to map brain activity by indicating changes in blood flow, volume, and oxygenation after neural activity [24]. As depicted in Eq. 2, the HRF peaks shortly after neural activity begins and then gradually returns to baseline.

$$h(t) = \frac{t^{a_1-1}e^{-(t/b_1-1)}}{b_1^{a_1}\Gamma(a_1)} - c \cdot \frac{t^{a_1-1}e^{-(t/b_1-1)}}{b_1^{a_1}\Gamma(a_1)} \tag{2}$$

The parameters a_1, b_1, a_2, and b_2 shape the response curve, with t as time. The equation models the initial blood flow increase and a delayed undershoot. Traditionally, brain activity maps are created by interpolating HRF parameters from multiple fNIRS channels. We propose DAE network to estimate HRF and minimise motion artefacts, outperforming conventional methods. The process generates time series or spatial maps of estimated brain activity using the HRF model on reconstructed fNIRS data. Figure 6 shows HRF activation patterns during a finger-tapping task for "tapping left" and "tapping right" conditions. The quality of the map is assessed by comparing the reconstruction results with the truth of the ground. An initial model had a baseline shift, reducing light intensity and eliminating weaker responses, while the DAE model with optimised BN accurately reconstructed most responses with high similarity to the ground truth.

The energy consumption of the DAE network training in CIFAR-10 using a Nvidia GPU and our BN engine is illustrated in Table II, demonstrating the performance of the engine in reducing zero copy latency and much lower power consumption while maintaining high accuracy.

TABLE II
PERFORMANCE COMPARISON BETWEEN DIFFERENT DEVICES

Device	Average Power (W)	BN back-propagation Time (ms)	MSE
GPU	226.65	0.84	GroundTruth
Accelerator	48.74	1.8e-5	9.11919e-07

V. CONCLUSION

In this work, we address a long-standing bottleneck in edge computing for efficient on-device training when batch normalisation dominates training cost. The paper introduces a custom BN engine, implemented in FPGA fabric, reducing BN latency from tens of cycles to 9 cycles. It supports 32 parallel BN units, optimising performance for real-time

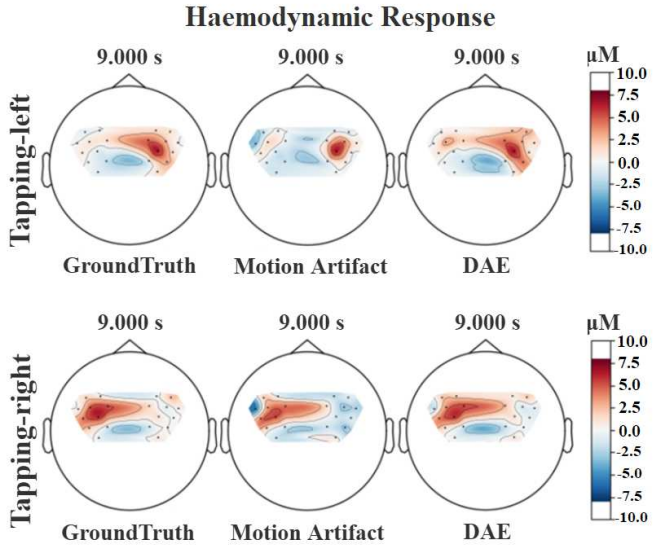

Fig. 6. Topographical map of haemoglobin concentration changes during finger-tapping tasks.

applications. It demonstrates that the framework of a biomedical application (functional near-infrared spectroscopy), achieves sub-millisecond latency and preserving data privacy by doing all computation locally. By keeping fNIRS signals and training computation on-chip, this work enhances patient privacy and supports real-time brain monitoring in clinical or wearable devices. While evaluated on fNIRS data, the BN engine and SoC deployment framework are reusable across DAEs and UNets, making this a versatile platform for edge computing in ANN-based applications. Importantly, this platform enables developers to deploy high-level ANN models without dealing with hardware complexities. Supports runtime scheduling, quantisation, and memory tuning via AMD's Vitis AI stack. In the BN engine, memory transfer will be evaluated and the performance on live NIRS sensors to test robustness to real-time motion artefact detection and fast image reconstruction.

References

[1] Y. Zhao, Y. Xia, R. Loureiro, H. Zhao, U. Dolinsky, and S. Yang, "Fpl demo: A learning-based motion artefact detector for heterogeneous platforms," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2023, pp. 366–366.

[2] D. Han, S. Kang, S. Kim, J. Lee, and H.-J. Yoo, "Energy-efficient dnn training processors on micro-ai systems," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 2, pp. 259–275, 2022.

[3] H. Dai, H. Wang, X. Zhang, and H. Sun, "Memory-efficient batch normalization by one-pass computation for on-device training," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 6, pp. 3186–3190, 2024.

[4] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[5] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[6] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 807–820.

[7] Z. Wang, Y. Zhong, X. Cui, Y. Kuang, and Y. Wang, "A spiking neural network accelerator based on ping-pong architecture with sparse spike and weight," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.

[8] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks." *Proceedings of Machine Learning and Systems*, vol. 1, pp. 1–13, 2019.

[9] J. Wang, L. Lun, Z. Dai, Y. Jiang, and X. Cui, "A 16.41 tops/w cnn accelerator with event-based layer fusion for real-time inference," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5.

[10] M. Wang, X. Wu, J. Lin, and Z. Wang, "An fpga-based accelerator enabling efficient support for cnns with arbitrary kernel sizes," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5.

[11] Y. Chen, B. Liu, Y. Xu, J. Wu, X. Chen, P. Liu, Q. Zhou, and Y. Han, "Accelerating frequency-domain convolutional neural networks inference using fpgas," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5.

[12] S. Yang and Z. Yu, "A highly integrated hardware/software co-design and co-verification platform," *IEEE Design & Test*, vol. 36, no. 1, pp. 23–30, 2018.

[13] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 12 949–13 013, 2023.

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[15] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versaltm architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 84–93.

[16] AMD, "Vitis ai development environment," 2024, accessed: 26-Oct-2024. [Online]. Available: https://www.amd.com/en/products/software/vitis-ai.html

[17] Y. Gao, H. Chao, L. Cavuoto, P. Yan, U. Kruger, J. E. Norfleet, B. A. Makled, S. D. Schwaitzberg, S. De, and X. R. Intes, "Deep learning-based motion artifact removal in functional near-infrared spectroscopy," *Neurophotonics*, vol. 9, no. 4, p. 041406, 2022. [Online]. Available: https://doi.org/10.1117/1.NPh.9.4.041406

[18] Intel, "The ai pc opportunity," 2024, accessed: 28-Oct-2024. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2024-01/the-ai-pc-opportunity-white-paper.pdf

[19] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, and M. S. Hämäläinen, "MEG and EEG data analysis with MNE-Python," *Frontiers in Neuroscience*, vol. 7, no. 267, pp. 1–13, 2013.

[20] T. J. Huppert, S. G. Diamond, M. A. Franceschini, and D. A. Boas, "Homer: a review of time-series analysis methods for near-infrared spectroscopy of the brain," *Applied optics*, vol. 48, no. 10, pp. D280–D298, 2009.

[21] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, pp. 234–241. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28

[23] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2016, pp. 424–432. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46723-8_49

[24] R. Ercan, Y. Xia, Y. Zhao, R. Loureiro, S. Yang, and H. Zhao, "A real-time machine learning module for motion artifact detection in fnirs," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5.