AFRL-RI-RS-TR-2023-024

EMBEDDED DEEP LEARNING AND ADVANCED COMPUTATION

HARVARD UNIVERSITY

FEBRUARY 2023

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2023-024 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
JACK P. LOMBARDI, III
Work Unit Manager

/ S /
GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
FEBRUARY 2023	FINAL TECHNICAL REPORT	AUGUST 2018	SEPTEMBER 2022

4. TITLE AND SUBTITLE

EMBEDDED DEEP LEARNING AND ADVANCED COMPUTATION

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
	FA8750-18-1-0112	61102F
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R2MG

6. AUTHOR(S)

H. T. Kung

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Harvard University
150 Western Avenue
Allston MA 02134

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RITB 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2023-024

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this project, several novel techniques were developed for accelerating deep learning computation on embedded devices with highly constrained computing resources. These techniques include: (1) using variable precision block floating point with stochastic rounding, (2) employing term quantization which quantizes floating point numbers into power-of-two terms, (3) extending pre-trained language models with domain-specific vocabulary, (4) minimizing memory access with schedules using constant bandwidth blocks, (5) applying full-stack optimization in the co-design of algorithms, models and architectures, (6) splitting neural networks for wearable computing, (7) designing algorithms for detecting input to DNNs which is out-of-distribution, (8) packing sparse DNNs for efficient systolic array implementations of DNNs, (9) designing memory-on-logic architectures and systolic building blocks for 3D-IC implementations of DNNs, and (10) leveraging bit-level sparsity in in-memory computing. These methods complement each other and are applicable to all resource-constrained deep learning accelerators.

15. SUBJECT TERMS

Deep learning, edge computing, embedded system, quantization, neural netowork

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	25

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
JACK P. LOMBARDI, III	N/A

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1.0 SUMMARY

This final technical report summarizes the R&D efforts for the AFRL project "Embedded Deep Learning and Advanced Computation," which occurred from September 2018 through September 2022. The project addresses two significant technological trends: deep learning using deep neural networks (DNNs) is quickly becoming a go-to method for many machine-learning applications, and embedded devices on the network edge are becoming widely available. Via the integration of these two technologies, we see a new generation of embedded devices emerging that can perform intelligent tasks such as learning unknown environments and targets of interest, 3D mapping of surroundings, performing data analytics, and offering predictions based on local data. These devices form the so-called Artificial Intelligence of Things (AIoT) [1].

This research project studies foundational techniques that can facilitate future generations of efficient training and inference computing systems for embedded deep learning. The key methods studied include: (1) using variable precision block floating point with stochastic rounding, (2) employing term quantization which quantizes floating point numbers into power-of-two terms, as opposed to conventional uniform quantization, (3) ex-tending pre-trained language models with domain-specific vocabulary, (4) minimizing memory access with schedules using constant bandwidth blocks, (5) applying full-stack optimization in the co-design of algorithms, models and architectures, (6) splitting neural networks for wearable computing, (7) designing algorithms for detecting input to DNNs which is out-of-distribution, (8) packing sparse DNNs for efficient systolic array implementations of DNNs, (9) designing memory-on-logic architectures and systolic building blocks for 3D-IC implementations of DNNs, and (10) leveraging bit-level sparsity in in-memory computing.

2.0 INTRODUCTION

Deep neural networks (DNNs), such as convolutional neural networks (CNNs), have enabled deep learning. They have many layers of non-linear feature transformation, with each additional layer extracting increasingly better features. However, training a large DNN and performing inference on such a network can be time- and energy-consuming. Training is costly for large deep networks such as large language models (LLMs) and generative adversarial networks (GANs). Training and inference become even more challenging on small Artificial Intelligence of Things (AIoT) devices due to the stringent resource constraints of these devices regarding processing capability, memory footprint, and power budget. It is necessary to co-design algorithms, models, and hardware parameters for performing efficient training and inferencing on these embedded devices. Advanced computing techniques, such as parallel processing, managing memory access schedules, and leveraging data sparsity, are essential when training sophisticated deep models for embedded devices.

This project was to conduct broad-based research on critical challenges in embedded deep learning and inference. For example, we explored the interplay between embedded deep learning and advanced computing based on parallel and distributed computing. The project leveraged PI's experience in related subjects, including his early work on systolic arrays and distributed and embedded neural network architectures. We highlight some of our published results in the rest of this report.

3.0 METHODS, ASSUMPTIONS, PROCEDURES

3.1. Blocks CAKE: Matrix Multiplication Using Constant-Bandwidth Blocks

We have developed a novel scheduling algorithm for matrix-matrix multiplication computations on computing platforms with memory hierarchies. The method uses constant bandwidth blocks (CB) blocks to improve computation throughput for computing systems limited by external memory bandwidth.

Configuring the shape and size of CB blocks operating from within any memory hierarchy level (e.g., internal SRAM), we achieve high throughput while holding external bandwidth (e.g., with DRAM) constant. We demonstrate how, surprisingly, CB blocks can maintain constant external bandwidth as computation throughput increases. Analogous to partitioning a cake into pieces, we dub our CB-partitioned schedule CAKE [2].

CAKE can outperform state-of-the-art libraries in computation time on real-world systems where external bandwidth represents a bottleneck, demonstrating CAKE's ability to address the memory wall. CAKE achieves superior performance by directly using theoretically optimal CB-partitioned blocks in tiling and scheduling, obviating the need for extensive design search.

Specifically, a constant bandwidth (CB) block is a block with dimensions (n, m, k) shaped and sized according to external bandwidth, as seen in Figure 1. CB block shaping controls the arithmetic intensity, allowing us to match external IO time with computation time. Arithmetic intensity (α) is defined as the ratio of computation volume to data transferred, which is equivalent to the percentage of computation throughput (CT) to external memory bandwidth (BW): $\alpha = CT/BW$. Therefore, we can, for example, increase CT or decrease required BW by using CB blocks to control α. We can also increase the size of the CB block to leverage any additional BW and reduce internal memory size requirements.

We illustrate an example performance result of CAKE. Figure 2 shows CAKE's speedup in computation throughput for square matrices of varying size compared to the state-of-the-art libraries MKL and ARMPL on the Intel i9-10900K and ARM v8 Cortex CPUs, respectively.

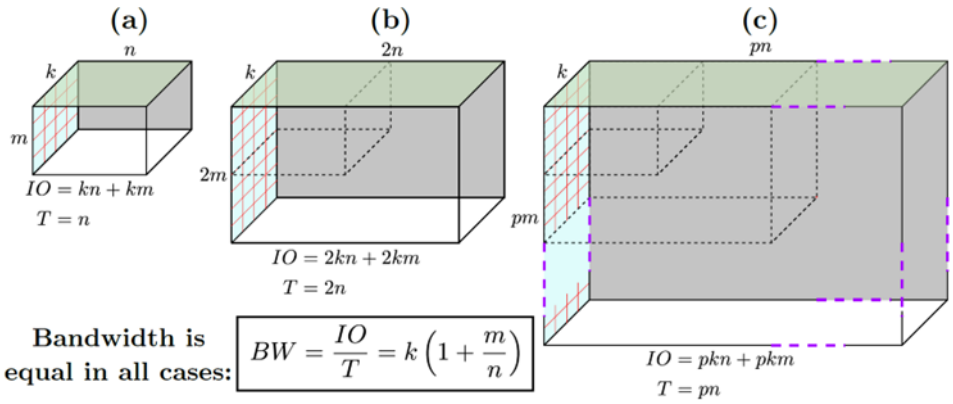

Figure 1: Changing the shape and size of a block can keep a block's external bandwidth (BW) constant when increasing the computation throughput (CT) by increasing the core count. The block gets taller and wider, matching IO and computation time T as volume changes

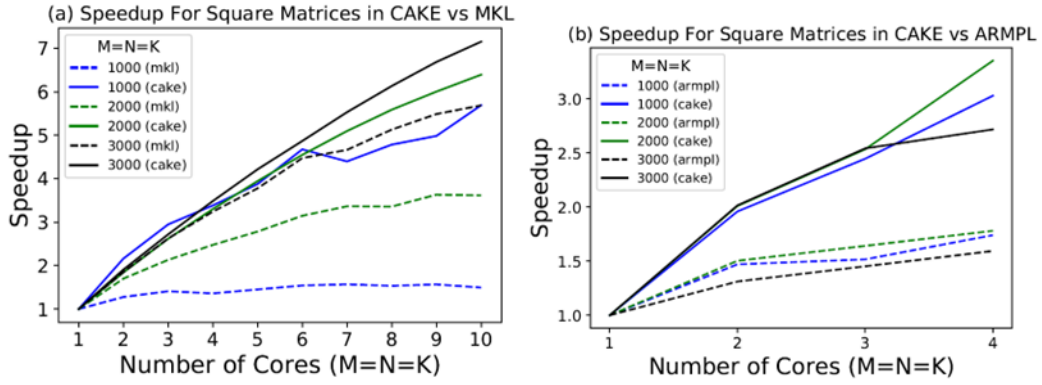

Figure 2: Speedup in computation throughput (speedup in time compared to a single core) for square matrices on Intel i9 10900K and ARM v8 Cortex CPUs. In (a), CAKE's speedup improvement over MKL is more pronounced for small matrices. MKL's performance approaches CAKE as matrix size increases and achievable throughput reaches a steady state. In (b), CAKE consistently outperforms ARMPL on square MM for all sizes. Unlike CAKE, ARMPL cannot scale performance with the number of cores due to limited DRAM BW on the ARM CPU

3.2. Full-stack Model Training Methodology

We have developed a full-stack optimization framework based on model training for accelerating CNN inference and validated the approach with an FPGA implementation [3]. Under the framework, we jointly optimize CNN models, computing architectures, and hardware implementations. Our full-stack system outperformed the state-of-the-art in the trade-off space characterized by inference latency, energy efficiency, hardware utilization, and inference accuracy. An FPGA implementation is used as the validation vehicle for our design, achieving a 2.28ms inference latency for the ImageNet benchmark. Our implementation has 9x higher energy efficiency compared to other implementations while achieving comparable latency. A highlight of our approach, which contributes to the high energy efficiency, is an efficient Selector-Accumulator (SAC) architecture for implementing CNNs with power-of-two model weights. Compared to an FPGA implementation for a traditional 8-bit multiplier–accumulator (MAC), SAC reduces required hardware resources (4.85x fewer lookup tables) and power consumption (2.48x).

We add quantization operations to the CNN training graph to achieve high classification accuracy using power-of-two weights to match the fixed-point weights and data used for inference. Figure 3 shows the training and inference graphs for a single layer in the CNN. In our training graph, we use log quantization for the weights, which quantizes an underlying full-precision weight (shown in blue) to the nearest power of two. During training, backpropagation uses full-precision gradients to update the full-precision weights in each layer.

Additionally, we perform quantization on the batch normalization operations, which follow each convolutional layer. For higher precision weights (e.g., 8-bit weights), these batch normalization parameters can be folded directly into the weights and bias terms of the preceding convolution layer after training so that they have no additional overhead during inference.

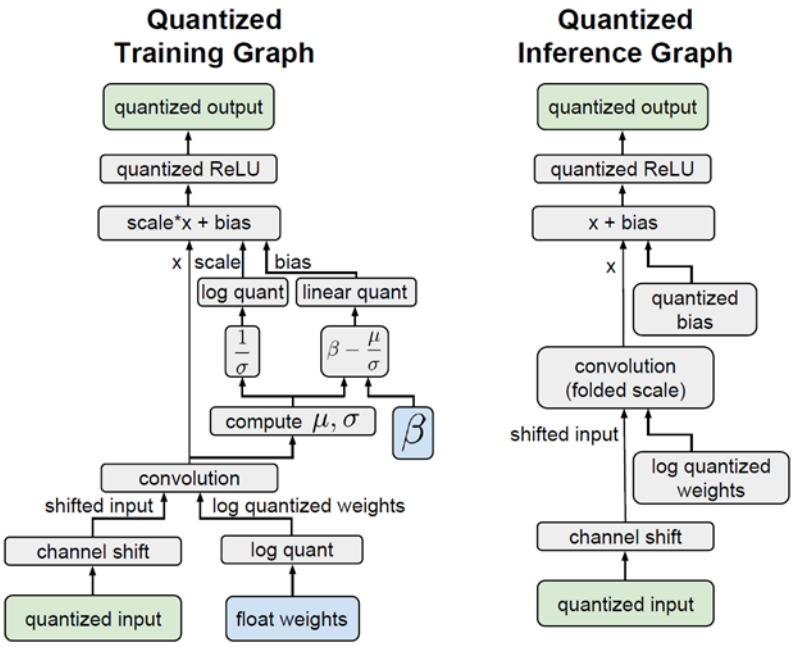

Figure 3: The quantized training graph (left) performs both linear quantization for input data and power-of-two quantization (log quantization) for weight and batch normalization parameters during training. The inference graph (right) uses the quantized version of the full-precision weights learned during training and therefore does not re-quire any floating-point operations

3.3. Term Quantization

We have developed a post-training quantization technique, called Term Quantization (TQ) [4, 5], for improved computational efficiency of pre-trained DNNs.

TQ operates on power-of-two terms in expressions of values. In conventional uniform quantization (UQ), quantization intervals have integral boundaries. In contrast, each boundary is expressible in a few power-of-two terms for TQ. Since multiplying with a power of two is a simple shift operation, TQ can realize substantial computation savings. In addition, we use TQ to implement tightly synchronized processor arrays, such as systolic arrays, for efficient parallel processing.

In computing a dot-product computation, TQ dynamically selects a fixed number of the largest terms to use from the values of the two vectors. By exploiting weight and data distributions typically present in DNNs, TQ has a minimal impact on DNN model performance (e.g., accuracy or perplexity). Figure 4 shows performance of TQ on an MLP for

MNIST, multiple CNNs for ImageNet, and an LSTM for Wikitext-2. We demonstrate reductions in inference computation costs (between 3-10×) compared UQ for the same level of model performance.

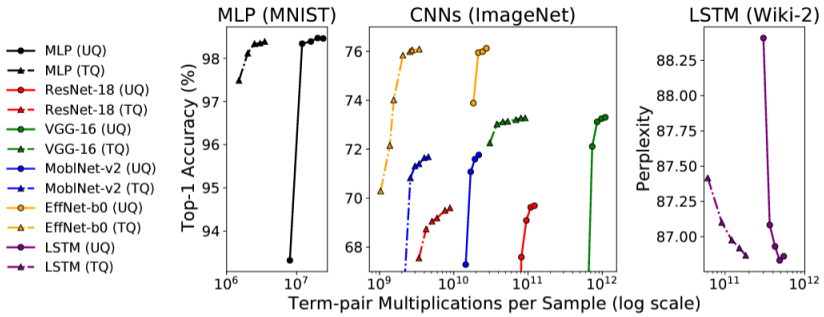

Figure 4: Comparing uniform quantization (UQ) and term quantization (TQ) for an MLP on MNIST (left), CNNs on ImageNet (center), and an LSTM on Wikitext-2 (right). The UQ settings vary the weight bitwidth (from 4 to 8 bits), while the TQ settings vary g (group size) and α (number of terms per group)

3.4. Packing Algorithms for Systolic Array Processing of Sparse CNNs

We have developed column combining, a novel approach to packing sparse convolutional neural networks into a denser format for efficient implementations using systolic arrays [6]. By combining multiple sparse columns of a convolutional filter matrix into a single dense column stored in the systolic array, the utilization efficiency of the systolic array can be increased (e.g., 8x) due to the increased density of nonzero weights in the resulting packed filter matrix. In combining columns, for each row, all filter weights but the one with the largest magnitude is pruned. The remaining weights are retrained to preserve high accuracy. We have studied the effectiveness of this joint optimization for both high utilization efficiency and classification accuracy with ASIC and FPGA designs based on efficient bit-serial implementations of multiplier-accumulators. We demonstrate that in mitigating data privacy concerns, the training can be accomplished with only fractions of the original dataset (e.g., 10% for CIFAR-10). We present analysis and empirical evidence on the superior performance of our column combining approach against prior arts under metrics such as energy efficiency (3x) and inference latency (12x).

In the rest of this section, we provide an overview of the column-combining approach. Given a sparse filter matrix, we first partition it into column groups by grouping columns with minimal conflicts. Then, for each column group, we combine the sparse columns in the group into a single combined column by applying column-combine pruning. We aim to achieve two objectives simultaneously. First, we pack the given sparse filter matrix into a dense matrix, called a packed filter matrix, for efficient systolic array implementations with a small number of columns. Second, we minimize the impact of column-combine pruning on classification accuracy.

For high-density packing, we adopt a dense-column-first combining policy that favors selections of combining columns which result in high-density combined columns, where the density of a column is the percentage of nonzeros in the column. We then retrain the remaining weights after column-combine pruning for high classification accuracy.

The algorithm involves some parameters: α (the maximum number of sparse columns that can be combined into a single dense column), β (the pruning rate schedule for unstructured pruning), and γ (the average number of conflicts per row allowed for each group). Typically, α is small (e.g., 2) for early smaller CNN layers and large (e.g., 8) for later large CNN layers, which have more capacity to be pruned. For β, we prune up to a target number of nonzeros after column combining for each layer. A small number of conflicting elements γ are allowed between the columns in a group. For instance, in the blue group, (-3) in column 1 conflicts with (7) in column 3 and -8 in column 5. The conflicting (-3) and (7) weights are pruned, and -8 is kept as it has the largest magnitude. In (b), each group is combined into a single column to be loaded into a column in the proposed systolic array. For γ, a value of 1.75 is sufficient to achieve good packing efficiency (e.g., over 90% nonzeros) after column combining.

Figure 5 depicts a column-combining example. In (a), a filter matrix F, associated with a sparse convolutional layer, is divided along columns into three groups (blue, green, and red). The zero-valued weights in F resulting from previous pruning steps are omitted for illustration clarity. The objective of column grouping is to select sparse columns that, when combined, result in a single combined column with high packing efficiency (i.e., components are mostly nonzero). High packing efficiency translates to an increased utilization efficiency of the systolic array, as more MACs will perform useful computation by storing nonzero weights. A small number of conflicting elements γ are allowed between the columns in a group. For instance, in the blue group, (-3) in column 1 conflicts with (7) in column 3 and -8 in column 5. The conflicting (-3) and (7) weights are pruned, and -8 is kept as it has the largest magnitude. In (b), each group is combined into a single column to be loaded into a column in the proposed systolic array.

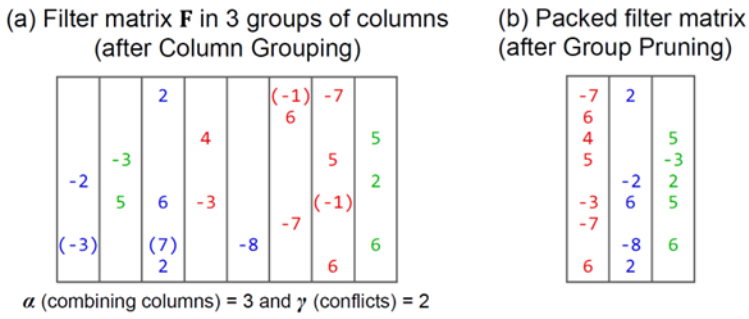

Figure 5: Example of combining columns

3.5. FAST: DNN Training Under Variable Precision Block Floating Point

We developed an efficient training scheme for DNNs, called Fast First, Accurate Second Training (FAST) [7]. With the weights, activations, and gradients represented in Block Floating Point (BFP), FAST supports matrix multiplication with variable precision BFP input operands, enabling incremental increases in DNN precision throughout training. By increasing the BFP precision across both training iterations and DNN layers, FAST can significantly shorten the training time while reducing overall hardware resource usage. Our FAST Multiply-Accumulate (fMAC) supports dot product computations under multiple BFP precisions.

We have validated our FAST system on multiple DNNs with different datasets, demonstrating a 2-6× speedup in training on a single-chip platform over prior work based on mixed-precision or block floating point number systems while achieving similar performance in validation accuracy. We use Time-to-Accuracy (TTA) as the evaluation metric to compare different approaches. Figure 6 shows the TTA for ResNet-18 models trained under various precisions to achieve a validation accuracy of 68% on ImageNet. The training time is normalized by the FAST-Adaptive model, which has the shortest training time while still achieving the target accuracy.

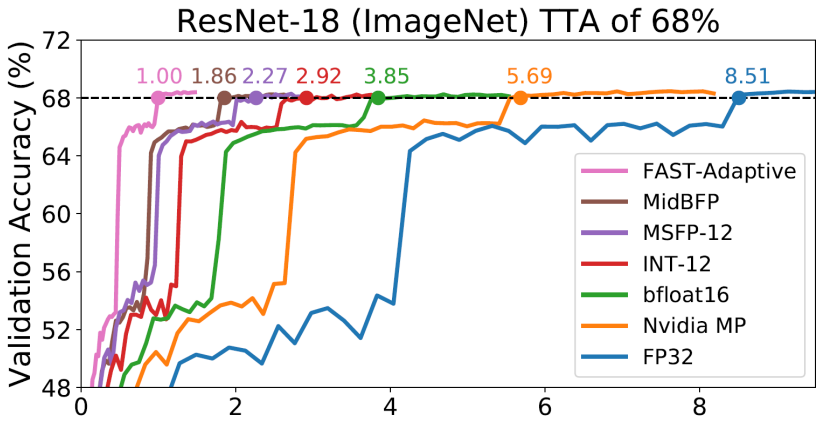

Figure 6: The validation accuracy curves and TTA values for achieving the 68% accuracy for ImageNet (dotted line)

4.0 RESULTS AND DISCUSSION

This project consisted of five techniques for enhancing embedded deep learning and thus provides five different results on different aspects of deep learning enhancement. The results are summarized in Table 1. These results show the different results and aspects they improve, which allow for a broad, large-scale improvement in deep learning for constrained applications.

Table 1: Summary of Results

Method	Result
CAKE	2.5-7x speedup compared to MKL for computation throughput on square matrices
Full-stack optimization framework	Quantization of training and interference processes. Selector-Accumulator (SAC) architecture reduces lookup tables by 4.85x and power consumption by 2.48x compared traditional 8-bit multiplier–accumulator (MAC)
Term Quantization	Reductions in inference computation costs between 3-10x with similar accuracy, as compared to uniform quantization
Packing Algorithms for Systolic Array Processing of Sparse CNNs	Column combining approach to packing sparse convolutional neural networks into an efficient implementation using systolic arrays, giving 3x energy efficiency and 12x inference latency over prior state of the art
FAST: DNN Training Under Variable Precision Block Floating Point	Use of multiple floating point precisions 2-6× speedup in training over previous mixed precision and 8.51x reduction in training time for ResNet-18

5.0 CONCLUSIONS

In this project, we have developed several novel techniques to facilitate future generations of efficient training and inference computing systems for embedded deep learning. These techniques include minimizing memory accesses, quantizing models using power-of-two terms, packing sparse filters for parallel processing, and performing variable-precision training. These methods are applicable to all resource-constrained deep learning accelerators.

6.0 REFERENCES

[1] J. Zhang and D. Tao, "Empowering things with intelligence: a survey of the progress, challenges, and opportunities in artificial intelligence of things," *IEEE Internet of Things Journal,* vol. 8, p. 7789–7817, 2020.

[2] H. T. Kung, V. Natesh and A. Sabot, "CAKE: Matrix Multiplication Using Constant-Bandwidth Blocks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2021.

[3] B. McDanel, S. Q. Zhang, H. T. Kung and X. Dong, "Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation," in *Proceedings of the ACM International Conference on Supercomputing*, 2019.

[4] H. T. Kung, B. McDanel and S. Q. Zhang, "Term quantization: furthering quantization at run time," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[5] S. Q. Zhang, B. McDanel, H. T. Kung and X. Dong, "Training for multi-resolution inference using reusable quantization terms," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[6] H. T. Kung, B. McDanel and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[7] S. Q. Zhang, B. McDanel and H. T. Kung, "FAST: DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding," in *28th IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)*, 2022.

APPENDIX A – PUBLICATION

Zhang SQ, McDanel B, Kung HT, "DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding," in 2022 IEEE Int. Sym. on High-Performance Computer Architecture (HPCA-28), April 2022

Dong X, Guo J, Li A, Ting W-T, Liu C, Kung HT, "Neural Mean Discrepancy for Efficient Out-of-Distribution Detection," in Computer Vision and Pattern Recognition Conference (CVPR 2022), June 2022

Dong X, De Salvo B, Li M, Liu C, Qu Z, Kung HT, Li Z, "Split Nets: Designing Neural Architectures for Efficient Distributed Computing on Head-Mounted Systems," in Computer Vision and Pattern Recognition Conference (CVPR 2022), June 2022

Zhang SQ, McDanel B, Kung HT, Dong X, "Training for Multi-resolution Inference using Reusable Quantization Terms," ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2021), April 2022

Fernandes M, Kung HT, "A Novel Training Strategy for Deep Learning Model Compression Applied to Viral Classifications," in Int. Joint Conf on Neural Networks Purpose: Public conference, July 2021

Kung HT, Natesh V, Sabot A, "Cake: Matrix Multiplication Using Constant Bandwidth Blocks," in Int. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC21), November 2021

Kung HT, Natesh V, Sabot A, "Saturation RRAM Leveraging Bit-Level Sparsity Resulting from Term Quantization," in IEEE Int. Symp. on Circuits and Systems (ISCAS 2021), May 2021

Tai W, Kung HT, Dong X, Comiter M, Kuo C, "exBERT: Extending Pre-trained Models with Domain-specific Vocabulary Under Constrained Training Resource," in Findings of the 2020 Conf on Empirical Methods in Natural Language Processing (EMNLP 2020), November 2020

Teerapittayanon S, Kung HT, "DaiMoN: A Decentralized Artificial Intelligence Model Network," in IEEE Int. Conf. on Blockchain (Blockchain-2019), July 2019

Teerapittayanon S, Kung HT, "Term Quantization: Furthering Quantization at Run Time," in Int. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC20), November 2020

McDanel B, Zhang SQ, Kung HT, Dong Xin, "Full-stack Optimization for Accelerating CNNs Using Powers-of-Two Weights with FPGA Validation," in ACM Int. Conf. on Supercomputing (ICS), June 2019

McDanel B, Zhang SQ, Kung HT, Dong Xin, "Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization," in

ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019), April 2019

Comiter, M, Teerapittayanon S, Kung HT, “CheckNet: Secure Inference on Untrusted Devices,” in IEEE Int. Conf. on Machine Learning and Applications (ICMLA 2019), May 2019

Comiter, M, Teerapittayanon S, Kung HT, “Maestro: A Memory-on-Logic Architecture for Coordinated Parallel Use of Many Systolic Arrays,” in IEEE Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP 2019), July 2019

Tillet P, Kung HT, Cox D, “Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations,” in ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2019), June 2019

Kung HT, McDanel B, Zhang SQ, Wang CT, Cai J, Chen CY, Chang VCY, Chen MF, Sun JYC, Yu D, “Systolic Building Block for Logic-on-Logic 3D-IC Implementations of Convolutional Neural Networks,” in IEEE Int. Sym. on Circuits and Systems (ISCAS 2019), May 2019

APPENDIX B – PRESENTATIONS

Meeting name: IEEE Int. Sym. on High-Performance Computer Architecture (HPCA-28)
Purpose: Public conference
Location: Seoul, South Korea
Date: April 2022
Attendees from this project: Sai Zhang (Remote attendance)
Presentation: "DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding"

Meeting name: Computer Vision and Pattern Recognition Conference (CVPR 2022)
Purpose: Public conference
Location: New Orleans, Louisiana
Date: June 2022
Attendees from this project: Xin Dong
Presentation: "Neural Mean Discrepancy for Efficient Out-of-Distribution Detection"

Meeting name: Computer Vision and Pattern Recognition Conference (CVPR 2022)
Purpose: Public conference
Location: New Orleans, Louisiana
Date: June 2022
Attendees from this project: Xin Dong
Presentation: "SplitNets: Designing Neural Architectures for Efficient Distributed Computing on Head-Mounted Systems"

Meeting name: ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2021)
Purpose: Public conference
Location: Virtual Meeting
Date: April 2022
Attendees from this project: Brad McDanel and Sai Zhang (Remote attendance)
Presentation: "Training for Multi-resolution Inference using Reusable Quantization Terms"

Meeting name: Int. Joint Conf on Neural Networks
Purpose: Public conference
Location: Shenzhen, China
Date: July 2021
Attendees from this project: Marcelo Fernandes (Remote attendance)
Presentation: "A Novel Training Strategy for Deep Learning Model Compression Applied to Viral Classifications"

Meeting name: Int. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC21)
Purpose: Public conference
Location: St. Louis, Missouri
Date: November 2021

Attendees from this project: Vikas Natesh and Andrew Sabot
Presentation: “Cake: Matrix Multiplication Using Constant Bandwidth Blocks”

Meeting name: IEEE Int. Symp. on Circuits and Systems (ISCAS 2021)
Purpose: Public conference
Location: Daegu, Korea
Date: May 2021
Attendees from this project: Brad McDanel (Remote attendance)
Presentation: “Saturation RRAM Leveraging Bit-Level Sparsity Resulting from Term Quantization”

Meeting name: Conf. on Empirical Methods in Natural Language Processing (EMNLP 2020)
Purpose: Public conference
Location: Virtual Meeting
Date: November 2020
Attendees from this project: Wen Tai (Remote attendance)
Presentation: “exBERT: Extending Pre-trained Models with Domain-specific Vocabulary Under Constrained Training Resource”

Meeting name: IEEE Int. Conf. on Blockchain (Blockchain-2019),
Purpose: Public conference
Location: Atlanta, GA
Date: July 2019
Attendees from this project: Surat Teerapittayanon
Presentation: “DaiMoN: A Decentralized Artificial Intelligence Model Network”

Meeting name: Int. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC20)
Purpose: Public conference
Location: Virtual meeting
Date: November 2020
Attendees from this project: H. T. Kung, Brad McDanel and Sai Zhang
Presentation: “Term Quantization: Furthering Quantization at Run Time”

Meeting name: ACM Int. Conf. on Supercomputing (ICS)
Purpose: Public conference
Location: Phoenix, Arizona
Date: June 2019
Attendees from this project: H. T. Kung and Brad McDanel
Presentation: “Full-stack Optimization for Accelerating CNNs Using Powers-of-Two Weights with FPGA Validation”

Meeting name: ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019)
Purpose: Public conference
Location: Providence, Rhode Island

Date: April 2019
Attendees from this project: H. T. Kung, Brad McDanel and Xin Dong
Presentation: “Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization”

Meeting name: IEEE Int. Conf. on Machine Learning and Applications (ICMLA 2019)
Purpose: Public conference
Location: Boca Raton, Florida
Date: May 2019
Attendees from this project: Marcus Comiter
Presentation: “CheckNet: Secure Inference on Untrusted Devices”

Meeting name: IEEE Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP 2019)
Purpose: Public conference
Location: New York, NY
Date: July 2019
Attendees from this project: H. T. Kung and Brad McDanel
Presentation: “Maestro: A Memory-on-Logic Architecture for Coordinated Parallel Use of Many Systolic Arrays”

Meeting name: ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2019)
Purpose: Public conference
Location: Phoenix, AZ
Date: June 2019
Attendees from this project: Philippe Tillet
Presentation: “Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations”

Meeting name: IEEE Int. Sym. on Circuits and Systems (ISCAS 2019)
Purpose: Public conference
Location: Sapporo, Japan
Date: May 2019
Attendees from this project: H. T. Kung
Presentation: “Systolic Building Block for Logic-on-Logic 3D-IC Implementations of Convolutional Neural Networks”

APPENDIX C – ABSTRACT

In this project, several novel techniques were developed for accelerating deep learning computation on embedded devices with highly constrained computing resources. These techniques include: (1) using variable precision block floating point with stochastic rounding, (2) employing term quantization which quantizes floating point numbers into power-of-two terms, (3) extending pre-trained language models with domain-specific vocabulary, (4) minimizing memory access with schedules using constant bandwidth blocks, (5) applying full-stack optimization in the co-design of algorithms, models and architectures, (6) splitting neural networks for wearable computing, (7) designing algorithms for detecting input to DNNs which is out-of-distribution, (8) packing sparse DNNs for efficient systolic array implementations of DNNs, (9) designing memory-on-logic architectures and systolic building blocks for 3D-IC implementations of DNNs, and (10) leveraging bit-level sparsity in in-memory computing. These methods complement each other and are applicable to all resource-constrained deep learning accelerators.

7.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ADC	Analog-to-Digital Conversion
AFRL	Air Force Research Lab
AIoT	Artificial Intelligence of Things
ARMPL	Arm Performance Libraries
ASIC	Application Specific Integrated Circuit
BFP	Block Floating Point
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
DRAM	Dynamic Random Access Memory
FLOPS	Floating Point Operations Per Second
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
HMD	Head-Mounted Display
LLM	Large Language Model
LSTM	Long Short-Term Memory networks
IO or I/O	Input/Output
MAC	Multiply-Accumulate
MKL	Math Kernel Library
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology database

PI	Principal Investigator
R&D	Research & Development
SRAM	Static Random Access Memory
TQ	Term Quantization
TTA	Time-to-Accuracy
UQ	Uniform Quantization