Messaging-based Intelligent Processing Unit (m-IPU) for next generation AI computing

Md. Rownak Hossain Chowdhury and Mostafizur Rahman, *Senior Member, IEEE*

***Abstract*— Recent advancements in Artificial Intelligence (AI) algorithms have sparked a race to enhance hardware capabilities for accelerated task processing. While significant strides have been made, particularly in areas like computer vision, the progress of AI algorithms appears to have outpaced hardware development, as specialized hardware struggles to keep up with the ever-expanding algorithmic landscape. To address this gap, we propose the development of a messaging-based intelligent processing unit (m-IPU) capable of runtime configuration to cater to various AI tasks. Central to this hardware is a programmable interconnection mechanism, relying on message passing between compute elements termed Sites. We illustrate the efficacy of m-IPU by implementing matrix multiplication and convolution operations, showcasing lower latency compared to current systolic array-based matrix multipliers. Our experiments, conducted on the TSMC 28nm technology node, reveal minimal power consumption of 44.5 mW with 94200 cells utilization. For 3D convolution operations on (32 X 128) images, each (256 X 256), using a (3 X 3) filter and 4096 Sites at a frequency of 100 MHz, m-IPU achieves processing in just 503.3 milliseconds. These results underscore the potential of m-IPU as a unified, scalable, and high-performance hardware architecture tailored for future AI applications.**

***Index Terms*—Machine Learning, Hardware Accelerator, Matrix Multiplication, Convolution, Reconfigurable Computing.**

I. Introduction

The advent of Artificial intelligence (AI), particularly Deep Neural Networks (DNNs) has prompted a paradigm shift in various fields such as computer vision, natural language processing, robotics, and many more. As a result, modern tech industries are aggressively integrating advanced neural network architectures like AlexNet [1], VGGNet [2], GoogLeNet [3], ResNet [4], LSTM [5], GRU [6], transformers [7] and others in hardware to uplift customer experience and maintain competitive differentiation. However, to materialize the full potential of AI, the underlying hardware architectures needs to be endowed with enhanced computational power to embrace sophisticated AI algorithms and complex neural network architectures [8][9]. The hardware performance of AI applications heavily depends on handling of matrix multiplication operations efficiently as these operations are central to representing neural networks [10][11]. Therefore, the research fraternity has put tremendous efforts in developing cutting-edge hardware accelerators like GPU [12], TPU [13], MEISSA [14], SpArch [15], MatRaptor [16] for efficient implementation of computationally intensive tasks.

However, these existing architectures offer advantages in specific use-cases while overlooking requirements of other use cases. For instance, architectures like SpArch and MatRaptor are useful in applications like Amazon co-purchase network [17] where the network is highly sparse. Similarly, MEISSA is beneficial over TPU in edge devices such as autonomous drone [18] providing less latency whereas TPU shows superiority in applications that necessitates high throughput like data centers. The application-centric nature of these hardware architectures, coupled with their lack of programmability may lead to suboptimal performance while deploying diverse set of implementations considering the evolving landscape of AI algorithms. This necessitates the development of adaptable hardware architectures at runtime along with the innovation of novel computing technology to effectuate neural network operations [19]. Reconfigurable computing, compared to conventional architectures, holds significant promise to suit specific needs of varying applications optimizing computing resources [20]. Hence, several reconfigurable circuits like memristor-based reconfigurable circuit [21], fine-grained polymorphic circuit [22], noise-based configurable computing [23], crosstalk built-in memory [24] have been introduced to enhance the computational capabilities. Despite their versatility in

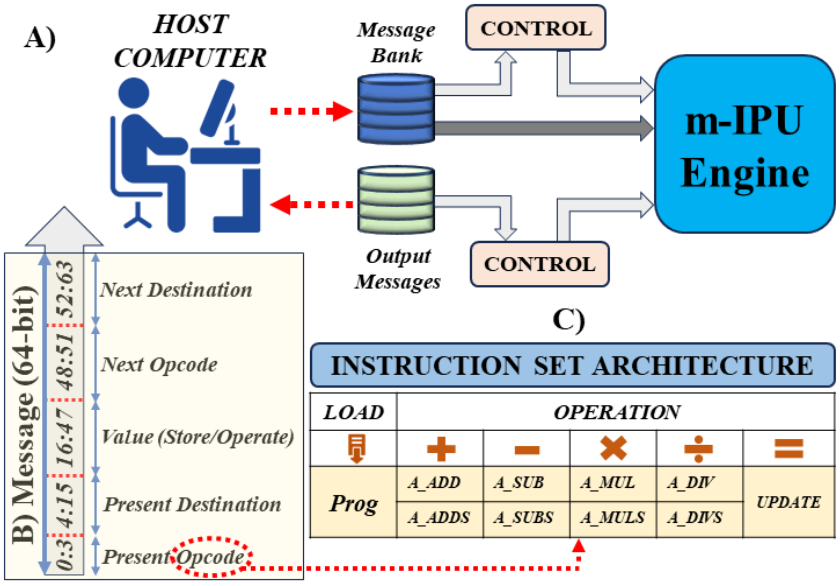


Figure 1 A) System Overview B) Message Encoding Scheme C) Instruction Set Architecture of m-IPU.

developing custom circuits, these reconfigurable architectures pose significant deficiencies when applied for machine learning or data analytics tasks. Therefore, we need a new class of reconfigurable architecture for computationally intensive tasks to map various matrix and manipulate matrix multiplications.

In this work, we introduce a new programmable hardware architecture to address the challenges called Messaging-based Intelligence Processing Unit (m-IPU). The configurability of m-IPU is rooted in its flexible interconnections that enables runtime mapping and processing of vast data volumes inherently via a message passing scheme without host processor intervention. In addition, the processing computing and memory elements are distributed and parallel in nature to facilitate fast computation. Due to these salient features, our evaluation results indicate significant performance advantages. Key contributions of this paper are as follows:

- Details of a new hardware accelerator architecture for computation intensive data analytics and AI applications.
- Details of application mapping to m-IPU architecture.
- Technology evaluation results at TSMC 28nm technology node.
- Throughput and latency comparison results against other prominent architectures.

The organization of the paper is as follows: Section II provides high level overview of m-IPU interfacing, example use cases such as matrix multiplication and convolutions, Section III elaborates on the technology validation using TSMC 28nm technology and highlights performance for key design metrics such as resource utilization, power profile, latency, and throughput, and provides a comparative analysis, and finally, Section IV concludes the paper and discusses future work.

II. M-IPU ARCHITECTURE

Messaging based Intelligent Processing Unit (m-IPU) is a reconfigurable computing architecture whose computing and memory elements are parallel and distributed. The cornerstone of our programmable hardware architecture is flexible virtual interconnections, which is pivotal for integrating messaging-based intelligence and thereby enabling efficient information processing. The proposed interconnect configuration is analogous to message passing in a human chain. For instance, if there are 5 people in a line and they want to pass messages from person #1 (Source) to person #5 (Destination), then persons #2, #3, and #4 form a virtual link between source and destination in that message passing scheme. This source, destination-based message passing scheme can be applied to computing cores as well.

A system level overview of the m-IPU is depicted in Figure 1(A). Here, a host CPU is required (like GPUs and other accelerators) to interpret high-level language (e.g., C, Python, etc.) and translate them into messages that m-IPU can operate upon and collect outputs form the m-IPU. Inside the m-IPU all communication between computing elements are performed through messages. The encoding scheme of a 64-bit message in m-IPU is shown in Figure 1 (B). A message can be segmented into 5 parts: a) Present Opcode (from bit position 0 to 3), b) Present Destination (from bit position 4 to 15), c) Values to be stored / operated (from bit position 16 to 47), d) Next Opcode (from bit position 48 to 51), and e) Next Destination (from bit position 52 to 63). In this framework, messages are routed to the desired hardware unit within the m-IPU based on the Present Destination, where an operation is performed on the value embedded in the message according to the Present Opcode. Subsequently, the Next Opcode and Next Destination specified in the message are retained inside the m-IPU to generate new message. To program m-IPU on-the-fly, we developed a lightweight instruction set architecture (ISA) as shown in Fig. 1(C) comprising only 10 instructions.

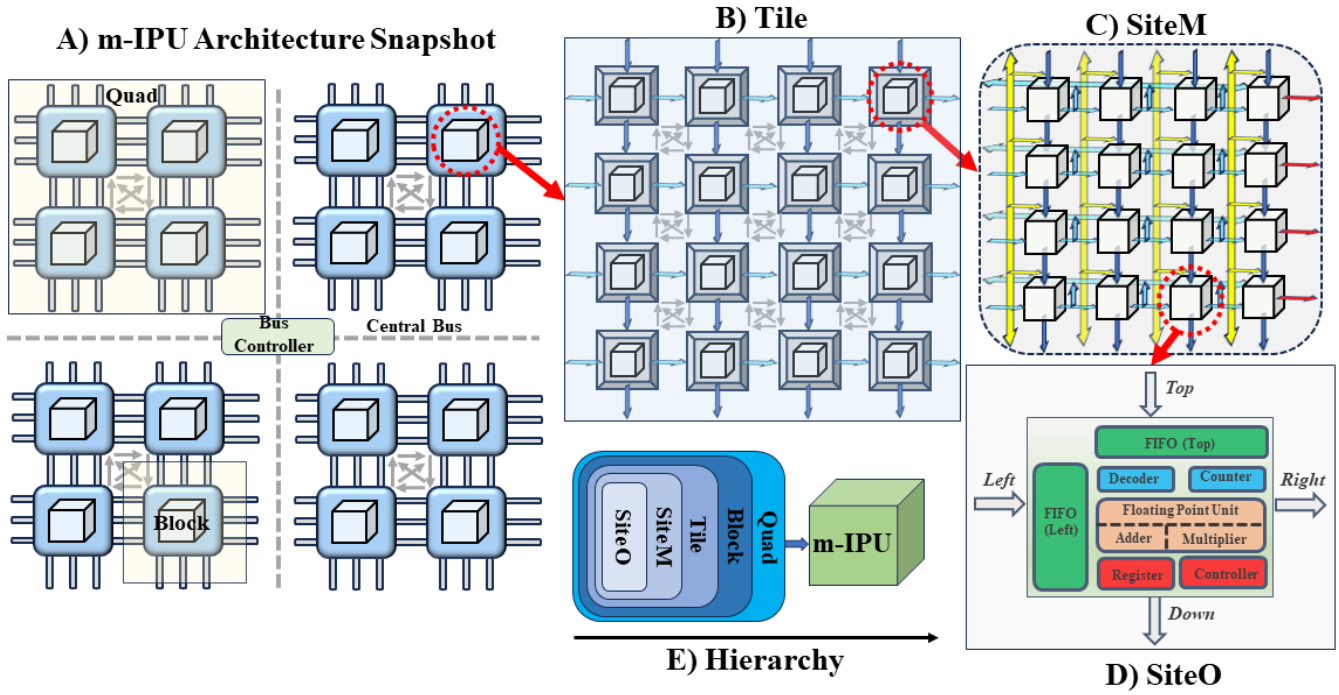


Figure 2 Details of A) m-IPU architecture B) Tile C) SiteM D) SiteO E) Hierarchy.

A. Hardware Details

The m-IPU follows a modular and hierarchical design approach enabling it to be scalable irrespective of data size or model complexity. The construction and segmentation of the m-IPU architecture is outlined in Figure 2 (A-E). Inside the m-IPU engine, there is an array of Quads. A Quad is a collection of 4 Blocks, and a Block is a collection of 16 Tiles. Each Tile consists of 16 SiteMs and each SiteM incorporates 16 SiteOs. The Quads, Blocks, Tiles, and SiteMs hierarchy allows task distribution and parallel computing. The SiteOs are the core elements and are analogous to Threads of GPUs or the Processing Elements (PEs) of TPUs. SiteOs are organized in rows and columns and the programmable interconnections between SiteOs allows the messages to be routed any cores. A SiteM collects all these messages and outputs 12 messages (4 for its own Tile, 4 for other Tiles within the same row, and 4 for different columns/Blocks) at a time. Like SiteMs organization in a Tile, a collection of Tiles is called Blocks. The Blocks communicate with each other through local and global buses. Each block also contains distributed embedded memory elements to store further instructions.

The hardware architecture of SiteO as shown in Figure 2(D) mainly comprises floating point unit (FPU), FIFO, decoder, counter, and register. Associated with each SiteO there is a small 8-word memory buffer to store the next set of instructions. The SiteOs also contain SRAMs to store weights. The SiteOs execute 32-bit IEEE 754 arithmetic operations, such as addition, multiplication, and subtraction, utilizing the Floating-Point Unit (FPU). SiteOs can receive messages either from the top or the left direction and they release outputs either at the right or the bottom direction. They are also aware of their neighbors (i.e., addresses of neighbor SiteOs in right, left, up, and down are stored in each SiteO). There are 2 FIFOs (Left and Top) to store incoming messages and push them towards execution or exit route in a pipelined manner. If the FIFOs are empty, the turnout time for in and out for a message is 1 cycle. If the FIFOs are full, the senders are sent a full signal to stop sending. The phase when stationary values are first loaded is called programming.

When a message arrives at SiteO, it first checks whether the destination of the message is its address, and if it matches, then the message is decoded and the instruction embedded

within the message is executed, otherwise, the message is passed on. After decoding a message, a SiteO can perform either message streaming or message forwarding. Streaming and forwarding are two different tasks; in the case of streaming, the SiteO receiving the message send it to its preferred neighbor by updating the message, whereas, in forwarding, the SiteO just behaves as a buffer to pass messages without intervention. Each Site, upon receiving or generating a message, checks whether the destination is within the same row or not; if it is, then it sends the message to the right and to down otherwise. Eventually, through hopping Sites, a message reaches its destination. If the messages are to be routed/passed downward, those messages are labeled as Tile message and if they are passed rightward (within the same SiteO row), those are labeled as Local messages. To serve two different purposes such as Data loading and mathematical operation, m-IPU needs just 10 instructions. Here, 1 instruction (***Prog***) is required for loading data inside m-IPU and the remaining 9 instructions (***UPDATE***, ***A_ADD***, ***A_ADDS***, **A_SUB**, ***A_SUBS***, ***A_MUL***, ***A_MULS***, ***A_DIV***, ***A_DIVS***) perform mathematical operations.

A SiteM is designed to have 16 SiteOs in rows and columns. Each column and row of the SiteM is equipped with four vertical and horizontal buses, respectively. The bus topology enables messages to be dispatched simultaneously at multiple SiteO locations rather than hopping thereby improving latency. This concept can be mimicked and extended to develop other hierarchical structures like tile, block, and quad. A Tile can have messages destined to itself (i.e., coming from within the Tile or outside the Tile), called Tile messages, and have incoming messages destined for other Tiles within the same row (called Local messages with respect to Blocks) and same column (called Block messages).

III. M-IPU COMPUTATION

Matrix multiplication and convolution operations are key computational kernels in various state-of-the-art AI applications. Hence, the performance of AI accelerators lies in the efficient execution of matrix multiplication and convolution operations. As AI technology advances, the complexity of neural networks continues to increase. For example, CNN architectures encompass multiple layers with varying filter size, padding, and stride to convert input image volume to output preserving class scores [26]. Ideally, the convolution operation is a repetitive sliding dot product according to filter size. However, this conventional approach requires many operations to be performed; hence, effective data mapping is parallelly significant along with efficient hardware architectures to accelerate convolution operations. Moreover, the performance metrics and data processing capabilities of convolution operation vary according to the application. Real-time systems such as autonomous vehicles [27], smartphones [28], medical devices[29], robotics [30] necessitates instant decision making based on sensory inputs. However, due to resource limitations, they are capable of single-batch processing; hence, the key design parameter in those applications is latency. On the other hand, applications like video processing [31], language modeling [32], speech recognition [33] process multiple batches at the same time; as a result, requires high throughput. This section will explore both the matrix multiplication and convolution operations in m-IPU, catering to the varying needs of these AI applications.

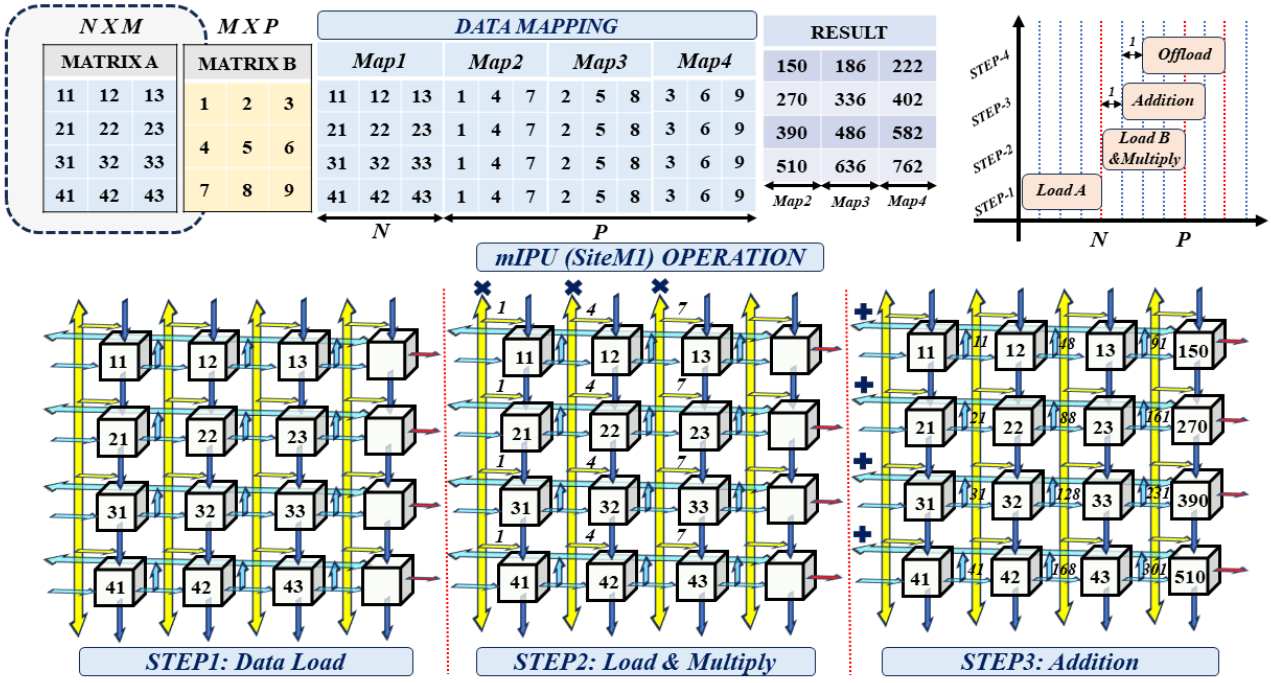


Figure 3 Matrix Multiplication Approach in m-IPU.

A. Matrix-Matrix Multiplication

The matrix multiplication operation inside m-IPU can be divided into four steps: a) Data (Matrix A) Load, b) Data (Matrix B) Load and multiply, c) Addition, and d) Offload. Figure 3 illustrates a matrix-matrix multiplication example within m-IPU utilizing a SiteM. Here, the inputs for the matrix multiplication are Matrix A (4 X 3) and Matrix B (3 X 3). According to the data mapping indicated in Figure 3, matrix A will be loaded only once following Map1. After that, each column of the matrix B will be loaded sequentially according to Map2, Map3, and Map4. Thus, the matrix multiplication can be surmised as three (3) matrix-vector multiplication operations. Therefore, the matrix-vector multiplication between matrix A and each column of matrix B will be repeated 3 times in one SiteM. The corresponding result of each matrix-vector multiplication will be accumulated to achieve the final output. Alternatively, the intended multiplication operation can also be completed using 3 SiteM fabrics of the m-IPU; where, 3 SiteMs will perform 3 matrix-vector multiplications parallelly in 3 SiteMs. Thus, the matrix-matrix multiplication can be acquired within the same time of matrix-vector multiplication using 3 SiteMs. Mathematically, the number of SiteOs and the latency needed to execute a matrix-matrix multiplication of Matrix A (N X M) and Matrix B (M X P) can be calculated using the equation (1) and (2) respectively.

$$SiteO_{MM} = \{(N \; X \; M) + N\} \; X \; P \quad (1)$$

$$T_{MM} = N + P + 2 \quad (2)$$

B. Convolution Operation

In our method to perform convolution operation, we have kept the filter values stationery and pixel values are streamed inside m-IPU according to stride value defined for convolution. A 2D convolution operation in m-IPU is shown in Figure 4 utilizing three SiteMs. Here, the convolution operation is performed for a 5X5 gray scale image using a 3X3 filter and the outcome of the convolution is a 3X3 matrix. In convolution operation, the image data will be divided into several data chunks equal to filter size based on convolution parameters like filter size, padding, and stride. The example shown in Figure 4 splits the 5 X 5 image data into 9 data chunks for filter size 3 X 3, padding 0 and stride 1. After creating those data chunks, the convolution operation can be surmised as 9 pointwise matrix-matrix multiplication and add operation where the two inputs are filter and one of the data

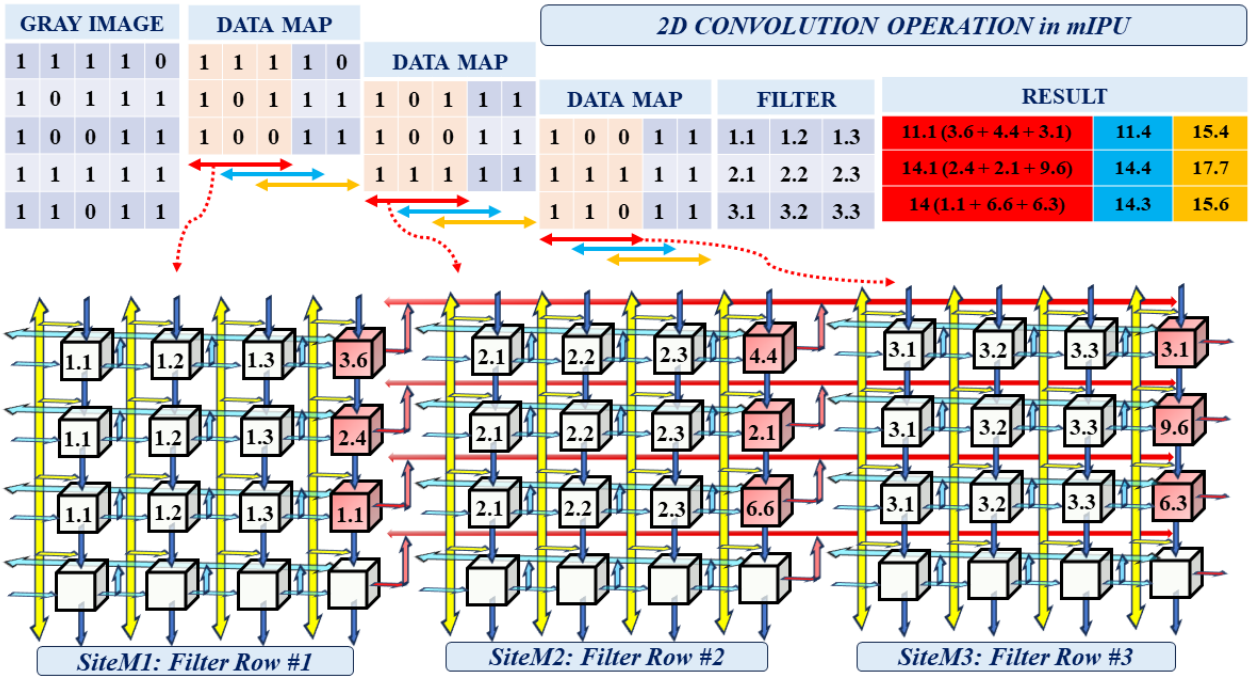


Figure 4 2D Convolution Operation in m-IPU.

chunks. In this case, 3 SiteMs will work parallelly to complete the 2D convolution where each SiteM will execute 3 pointwise matrix multiplications according to 3 data maps (red, blue, yellow). However, in Figure 3, we have only shown mapping for one portion (red) of the entire data for ease of demonstration. Here, each SiteM is initially programmed with a filter row. Subsequently, three data chunks, marked in red, will be sent to three SiteMs for concurrent multiplication. After that, the multiplication results of each row will be added and then the addition result from SiteM1 and SiteM2 will be forwarded to SiteM3 for final addition. Finally, the convolution result will be collected from SiteM3 to get the 1st column of the desired 3 X 3 convolution result.

IV. Validation

A. Methodology

To validate our design, we have developed a digital design flow based on CAD tools under TSMC 28nm technology

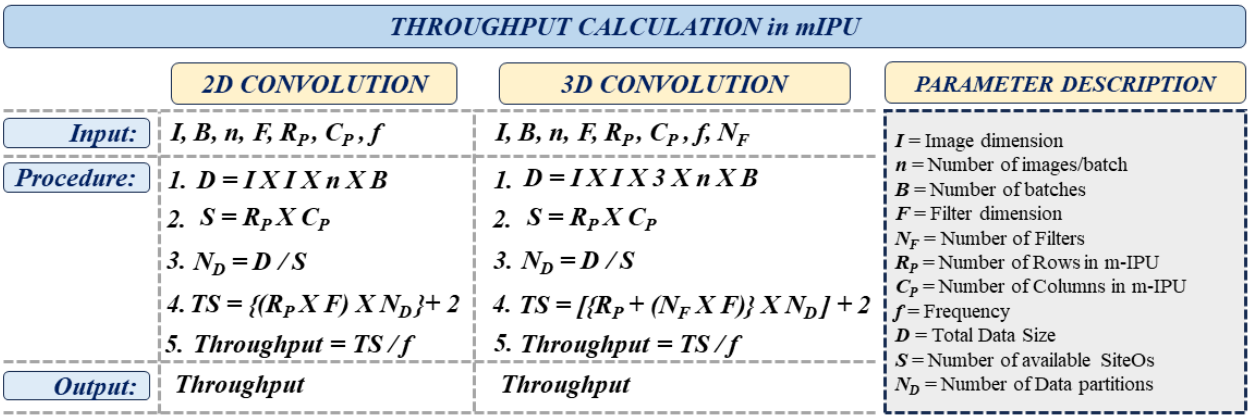


Figure 5 Throughput calculation of 2D and 3D convolution in m-IPU.

node. Initially, we designed the RTL/behavioral models of m-IPU and iteratively checked whether the RTL is free from linting errors and whether the RTL is synthesis friendly. During the RTL code development, we utilized Xilinx Vivado platform for initial measurement and simulation. Later, we followed a digital design flow based on standard methodologies and CAD tools for implementing the m-IPU design. We have used a high-performance compact mobile computing plus (CLN28HPC+) process from TSMC 28nm commercial PDK with 8 metal layers and supply voltage of 0.9V. Here, the RTL was verified for functional correctness. We have also performed property checking to verify the RTL implementation and that the specification matches. After that, we set the design environment, including the technology file and other environmental attributes. We have also defined the design constraints file for synthesis, usually called an SDC synopsys_constraints or dc_synopsys_setup file, specific to the synthesis tool. We have defined the environment by specifying operating conditions, wire load models, and system interface characteristics. Operating conditions include temperature, voltage, and process variations. Wire load models estimate the effect of wire length on design performance. System interface characteristics include input drives, input and output loads, and fan-out loads. Since the environment model directly affects design synthesis results, we have methodically constrained designs by describing the design environment, target objectives, and design rules. We also constrained timing and area information and iteratively run the synthesis on the design to meet the design specifications. Once the constraints file is set, we have provided synthesis inputs to the Cadence Genus. The input files are the library files (which have the functional/timing information available for the standard cell library and the wire load models for the wires based on the fan-out length of the connectivity), RTL files, and the design constraints files. The synthesis tool performs the synthesis of the RTL files and

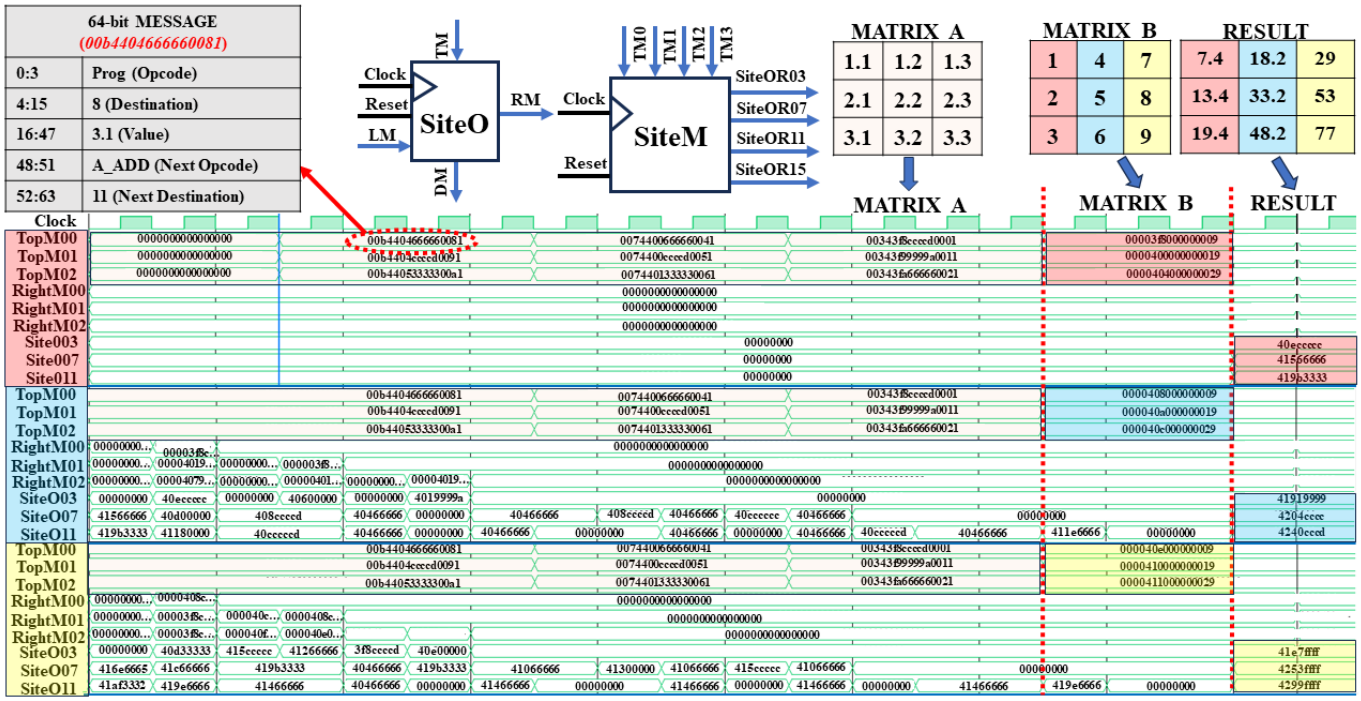


Figure 6 Simulation result of Matrix Multiplication between Matrix A (3 X 3) and Matrix B (3 X 3)

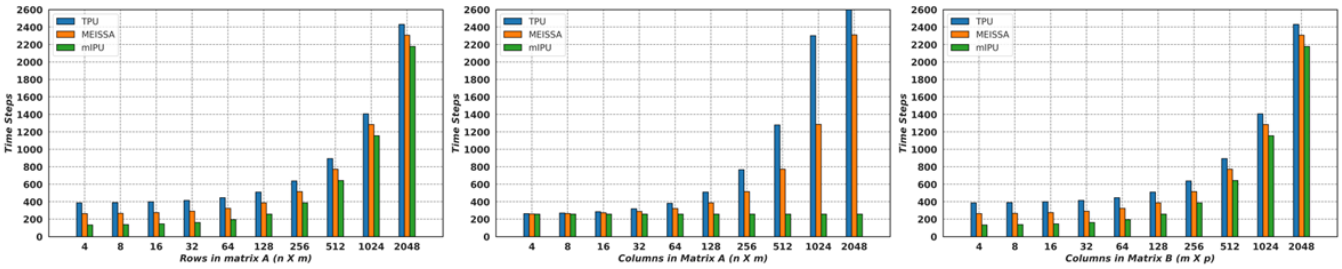


Figure 7 Comparison of the total time steps for matrix multiplication when A) n varies from 2 to 2048, m = 128, p = 128, B) m varies from 2 to 2048, n = 128, and p =128, and c) p varies from 2 to 2048, n = 128, m = 128.

maps and optimizes them to meet the design constraints requirements. Design optimization constraints define timing and area optimization goals for Cadence Genus. We specify these constraints. Genus optimizes the synthesis of the design by these constraints, but not at the expense of the design rule constraints. That is, Genus never attempts to violate the higher-priority design rules. We had to modify the design constraint several times to meet the design requirements. After performing the synthesis, we performed functional verification with the synthesized netlist to confirm that the synthesis tool has not altered the functionality. The throughput calculation of both 2D and 3D convolution operations utilizing available hardware resources is appended in Figure 5. We have measured the throughput for both 2D (Gray Image) and 3D (Color Image) convolution and during our observation, we varied both image and filter sizes. We have considered 32 data batches where each batch contains 128 images with 100 MHz clock frequency and 4096 available SiteOs. The padding and stride are kept 0 and 1 in both cases. For 3D convolution, we assumed that 64 filters are involved in the convolution process.

TABLE I
A Summary of Matrix Multiplication Approach between TPU, MEISSA, and m-IPU

TYPE	PROCESSING ELEMENT	RESOURCE UTILIZATION	SHAPE		METHOD	DATA LOAD		LATENCY
			Matrix A	*Matrix B*		*Matrix A*	*Matrix B*	
TPU	MAC	Multipliers: N X P Adders: M X P	N X M	M X P	Systolic Array	Stored	Left to Right	N + 2M + P -2
MEISSA	Multipliers & Adder Trees	Multipliers: M X P Adders: P X (M - 1)	N X M	M X P	Systolic Array	Stored	Left to Right	N + M + P + log (M) -2
m-IPU	SiteO	SiteOs: {(N X M) + N} X P	N X M	M X P	Messaging-based	Program	Vertical Bus	N + P + 2

B. Evaluation

To verify our matrix-matrix multiplication approach, we have simulated our design with numerous values, however, a matrix-matrix multiplication between matrix A (3X3) and matrix B (3X3) is appended in Figure 6 for demonstrating our matrix-matrix multiplication process. Here, 9 messages are sent initially at 3 different steps to program 9 SiteOs with matrix A. In this stage, data reaches respective SiteOs through hopping. After that, each column of matrix B is passed to the m-IPU using vertical bus for multiplication. Once the multiplication is finished, results are sent to the desired SiteO for addition. This process is repeated until all columns are transferred and finally results are collected.

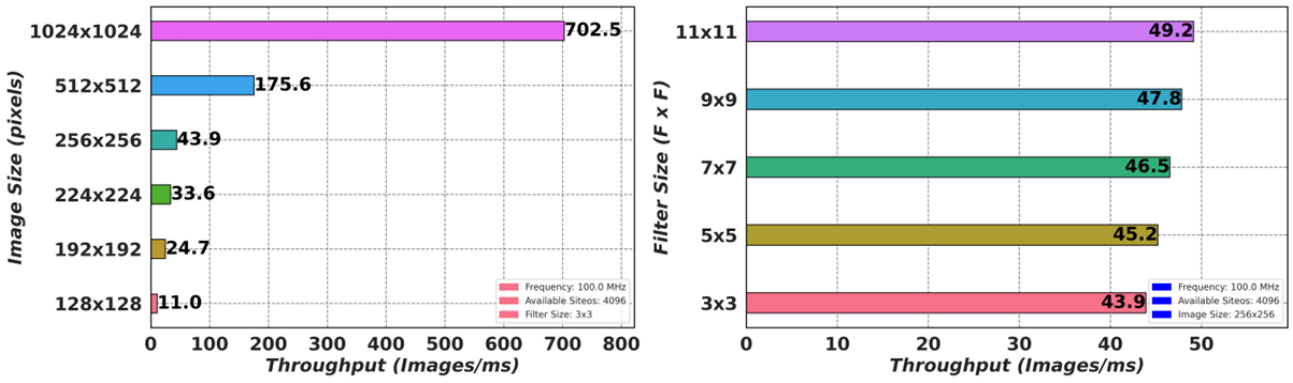
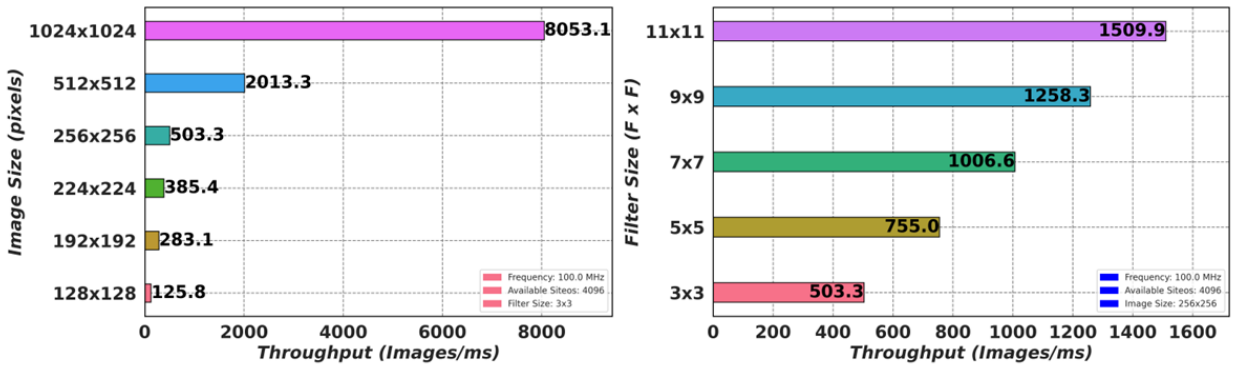


Figure 8 Throughput calculation of 2D convolution varying A) Image Size B) Filter Size

Table I summarizes the key characteristics including processing element, resource utilization, latency of matrix multiplication approaches between matrix A (N X M) and matrix B (M X P) in three different hardware accelerators (TPU, MEISSA, m-IPU). Both TPU and MEISSA utilize systolic array to perform matrix multiplication whereas our design incorporates messaging-based intelligent dataflow mechanism to program hardware at run-time. The latency of our matrix-matrix multiplication unit has been observed varying different (rows and columns) dimensions of matrix A and matrix B. Figure 7 (A), (B), and (C) represents the comparison of latency among MEISSA, TPU, and our design by varying N, M, and P individually from 4 to 2048 keeping other two parameters constant (128). In all cases, m-IPU outperforms the remaining two state-of-the art matrix multiplication architectures.



A) **B)**

Figure 4 Throughput calculation of 3D convolution varying A) Image Size B) Filter Size

Table II
m-IPU (SiteM) Design parameters

Technology	TSMC 28nm			
Process	HPC+			
Metal Layer	1P8M			
Voltage (V_{DD})	0.9V (nominal)			
Package	Wire Bond			
Frequency	100 MHz			
Power (mW)	**Leakage**	**Dynamic**	**Total**	
	0.42	44.08	44.5	
Cell Count	**Sequential**	**Inverter**	**Logic**	**Total**
	25692	4203	64305	94200

Besides, we evaluate power consumption and resource utilization under TSMC 28nm technology to demonstrate our architecture's efficiency. The key design parameters of the m-IPU architecture are listed in Table II that indicates the design consumes only 44.08 mW dynamic power at 0.9V operating voltage. The low dynamic power consumption suggests our design's efficient task partitioning at runtime due to intelligent programmability and effective data mapping. Moreover, messages are loaded only from one direction (Top) and after multiplication the SiteOs transfer data through horizontal bus; as a result, m-IPU involves less signal transmission at runtime.

Next, we identify an effective data mapping strategy to accomplish faster convolution operation with less hardware resources. Finally, we benchmark our design by calculating throughput for multi-batch convolution operation. Figure 8(A) represents throughput varying image sizes from (128 X 128) to (1024 X 1024) maintaining a fixed filter size of (3 X 3) for 2D convolution. On the other hand, figure 8(B) highlights throughput for 2D convolution for different filter sizes (from 3 X 3 to 11 X 11) while keeping a fixed image dimension of (256 X 256). Our methodology completes 2D convolution operation of (32 X 128) images of size (1024 X 1024) with a filter of size (3 X 3) utilizing 4096 SiteOs and 100 MHz frequency within just 702.5 milliseconds. Again, the time required to execute a 2D convolution operation for (32 X 128) images of dimension (256 X 256) using a (11 X 11) filter keeping the other parameters unchanged is only 49.2 milliseconds. Similarly, figure 9 (A) and 9 (B) demonstrate the throughput for 3D convolution varying image and filter size respectively.

V. Conclusion & Future Work

The trend of deploying machine learning and deep learning algorithms in widespread applications necessitates an efficient hardware accelerator for executing matrix multiplication operation within a limited hardware budget. To do so, developing a unified hardware architecture that features reconfigurability, low-power, low-latency, and high-throughput is essential. In this work, we presented an innovative messaging-based computing paradigm that inherently programs hardware units at runtime. In addition to this, we proposed a matrix multiplication mapping scheme that outperforms existing systolic array-based matrix multiplication architectures such as TPU and MEISSA. Our design has been validated through simulations under TSMC

28nm technology. We have also established a methodology to benchmark convolution operation utilizing available hardware resources and achieved high throughput. The impressive performance on both latency and throughput, combined with the novel computing technology of the m-IPU, indicates its unique position for the next generation AI applications. Our future work will involve a more thorough evaluation of m-IPU performance using software-hardware based emulations, m-IPU chip fabrication and demonstration of a software framework for application offloading.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, May 2012

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Computer Vision and Pattern Recognition, Sep. 2014.

[3] C. Szegedy *et al*., "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 1-9.

[4] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770-778.

[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[6] Chung, Junyoung, et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." ArXiv.org, 2014, arxiv.org/abs/1412.3555.

[7] A. Vaswani et al., "Attention Is All You Need," arXiv.org, Jun. 12, 2017. https://arxiv.org/abs/1706.03762.

[8] P. Dhilleswararao, S. Boppu, M. S. Manikandan and L. R. Cenkeramaddi, "Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey," in *IEEE Access*, vol. 10, pp. 131788-131828, 2022.

[9] L. R. Juracy, A. M. Amory and F. G. Moraes, "A Comprehensive Evaluation of Convolutional Hardware Accelerators," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 3, pp. 1149-1153, March 2023.

[10] S. Hadjis, F. Abuzaid, C. Zhang, and C. Re, "Caffe con troll: Shallow ideas to speed up deep learning," in Proceedings of the Fourth Workshop on Data analytics in the Cloud. ACM, 2015, p. 2.

[11] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "Cudnn: Efficient primitives for deep learning," arXiv preprint arXiv:1410.0759, 2014.

[12] W. J. Dally, S. W. Keckler and D. B. Kirk, "Evolution of the Graphics Processing Unit (GPU)," in *IEEE Micro*, vol. 41, no. 6, pp. 42-51, 1 Nov.-Dec. 2021.

[13] N. P. Jouppi *et al*., "In-datacenter performance analysis of a tensor processing unit," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, Toronto, ON, Canada, pp. 1-12, 2017

[14] B. Asgari, R. Hadidi and H. Kim, "MEISSA: Multiplying Matrices Efficiently in a Scalable Systolic Architecture," *2020 IEEE 38th International Conference on Computer Design (ICCD)*, Hartford, CT, USA, pp. 130-137, 2020

[15] Z. Zhang, H. Wang, S. Han and W. Dally, "SpArch: Efficient Architecture for Sparse Matrix Multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, San Diego, CA, USA, pp. 261-274, 2020.

[16] N. Srivastava, H. Jin, J. Liu, D. Albonesi and Z. Zhang, "MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Athens, Greece, pp. 766-780, 2020.

[17] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The Dynamics of Viral Marketing," Trans. on the Web (TWEB), 2007.

[18] Autel, Autel X-Star Quadcopter, 2020 (accessed March 15, 2024). [Online]. Available: https://www.autelrobotics.com/x-star-camera-drone.

[19] P. Holzinger and M. Reichenbach, "The HERA Methodology: Reconfigurable Logic in General-Purpose Computing," in IEEE Access, vol. 9, pp. 147212-147236, 2021.

[20] R. Tessier, K. Pocek and A. DeHon, "Reconfigurable Computing Architectures," in Proceedings of the IEEE, vol. 103, no. 3, pp. 332-354, March 2015.

[21] H. Xiao, X. Hu, T. Gao, Y. Zhou, S. Duan and Y. Chen, "Efficient Low-Bit Neural Network With Memristor-Based Reconfigurable Circuits," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 1, pp. 66-70, Jan. 2024.

[22] Md Arif Iqbal, Srinivas Rahul Sapireddy, S. Dasari, Kazi Asifuzzaman, and M. Rahman, "A review of crosstalk polymorphic circuits and their scalability," Memories - *Materials Devices Circuits and Systems*, vol. 7, pp. 100094–100094, Apr. 2024

[23] N. K. Macha, B. T. Repalle, M. A. Iqbal and M. Rahman, "Crosstalk-Computing-Based Gate-Level Reconfigurable Circuits," in *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 30, no. 8, pp. 1073-1083, Aug. 2022

[24] P. Samant, Naveen Kumar Macha, and M. Rahman, "A Neoteric Approach for Logic with Embedded Memory Leveraging Crosstalk Computing," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 19, no. 1, pp. 1–16, Dec. 2022

[25] M. Rahman, A. Iqbal, and S. Rahul, "A Messaging based Intelligent Computing Approach for Machine Learning Applications." Accessed: Mar. 20, 2024. [Online]. Available: https://computing-lab.com/wp-content/uploads/2022/02/m-IPU-v1.pdf

[26] T. Wiatowski and H. Bölcskei, "A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction," in *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1845-1866, March 2018.

[27] A. Gupta, K. Illanko and X. Fernando, "Object Detection for Connected and Autonomous Vehicles using CNN with Attention Mechanism," *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, Helsinki, Finland, 2022, pp. 1-6.

[28] Y. Li, J. Luo, S. Deng and G. Zhou, "CNN-Based Continuous Authentication on Smartphones with Conditional Wasserstein Generative Adversarial Network," in *IEEE Internet of Things Journal*, vol. 9, no. 7, pp. 5447-5460, 1 April1, 2022

[29] D. R. Sarvamangala and R. V. Kulkarni, "Convolutional neural networks in medical image understanding: a survey," Evolutionary Intelligence, vol. 15, Jan. 2021

[30] J. Guo, H. -T. Nguyen, C. Liu and C. C. Cheah, "Convolutional Neural Network-Based Robot Control for an Eye-in-Hand Camera," in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 8, pp. 4764-4775, Aug. 2023

[31] V. Sharma, M. Gupta, A. Kumar, and D. Mishra, "Video Processing Using Deep Learning Techniques: A Systematic Literature Review," in *IEEE Access*, vol. 9, pp. 139489-139507, 2021.

[32] S. Yang, "Natural Language Processing Based on Convolutional Neural Network and Semi Supervised Algorithm in Deep Learning," *2022 International Conference on Artificial Intelligence in Everything (AIE)*, Lefkosa, Cyprus, 2022, pp. 174-178.

[33] O. Abdel-Hamid, A. -r. Mohamed, H. Jiang, L. Deng, G. Penn and D. Yu, "Convolutional Neural Networks for Speech Recognition," in *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533-1545, Oct. 2014.