Energy Efficient Memory-based Inference of LSTM by Exploiting FPGA Overlay

Krishnendu Guha[1], Amit Ranjan Trivedi[2], Swarup Bhunia[3]
School of Computer Science and Information Technology, University College Cork, Ireland[1],
Department of Electrical and Computer Engineering, University of Illinois at Chicago, USA[2],
Department of Electrical and Computer Engineering, University of Florida, USA[3]
Email: kguha@ucc.ie[1], *amitrt@uic.edu*[2], *swarup@ece.ufl.edu*[3]

***Abstract*—The fourth industrial revolution (a.k.a. Industry 4.0) relies on *intelligent machines* that are fully autonomous and can diagnose and resolve operational issues without human intervention. Therefore, embedded computing platforms enabling the necessary computations for intelligent machines are critical for the ongoing industrial revolution. Especially field programmable gate arrays (FPGAs) are highly suited for such embedded computing due to their high performance and easy reconfigurability. Many Industry 4.0 applications, such as predictive maintenance, critically depend on real-time and reliable processing of time-series data using recurrent neural network models, especially long short-term memory (LSTM). Therefore, the FPGA-based acceleration of LSTM is imperative for many Industry 4.0 applications. Existing LSTM models for FPGAs incur significant resources and power and are not energy efficient. Moreover, prior works focusing on reducing latency and power mainly adhere to model pruning, which compromises the accuracy. Comparatively, we propose a memory-based energy-efficient inference of LSTM by exploiting overlay in FPGA. In our methodology, we pre-compute predominant operations and store them in the available embedded memory blocks (EMBs) of an FPGA. On-demand, these pre-computed results are accessed to minimize the necessary workload. Via this methodology, we obtained lower latency, lower power, and better energy efficiency than state-of-the-art LSTM models without any loss of accuracy. Specifically, when implemented on the ZynQ XCU104 evaluation board, a 3× reduction in latency and 5× reduction in power is obtained then the reference 16-bit LSTM model.**

***Index Terms*—LSTM, ML, FPGA, Memory-based Mapping, Energy Efficiency, Computing with Memory**

I. Introduction

The fourth industrial revolution (a.k.a. Industry 4.0) is leveraging various technological advances such as artificial intelligence (AI), internet-of-things (IoT), digital twin, and high-performance computing to enable machines to automate, diagnose, and resolve issues without interventions from humans. Additionally, the embedded computing platforms to facilitate such data processing on the *intelligent machines* must operate under stringent area, power, and cost constraints. For example, in [1], [2], predictive maintenance of wind turbines is performed by classifying their vibration characteristics but requires the embedded computing platform to operate on a battery, thus under power constraints, due to remote placement. Likewise, privacy-aware healthcare analytics for Industry 4.0 in [3] requires health sensor data to be processed near its user, thus constraining the embedded computing platform to be low cost and low power.

Field programmable gate arrays (FPGAs) are highly suited for such embedded computing on intelligent machines due to their high performance, low cost, and easy reconfigurability [4], [5]. Especially in many Industry 4.0 applications, such as predictive maintenance, the decision-making can be formulated as a recurrent neural network problem and solved using state-of-the-art models, such as long short-term memory (LSTM). Thus, methodologies for FPGA-based efficient acceleration of LSTM are imperative for various Industry 4.0 applications. Present-day FPGAs are associated with several embedded memory blocks (EMBs) to facilitate high-speed data access. EMBs are used for different purposes, such as the storage of inputs or intermediate data. However, they remain under-utilized in many computationally intensive applications, as shown in [6]. Thus, a methodology that can exploit the unused EMBs for memory-based computing of complex functions is desirable to improve FPGA's resource efficiency for complex neural network architectures [7].

Previous works mainly adhered to pruning strategies to reduce latency and power [8], [9], [10], [11], [12] for FPGA-based LSTM's acceleration. Though these are suitable for cloud environments using these for resource-constrained and low-energy budget FPGA-based platforms is an issue.

This work proposes a methodology for memory-based inference of complex neural network models, specifically LSTM, on FPGAs by leveraging unused EMBs. The fundamental computation in LSTM-based inference is matrix-vector multiplications, followed by vector additions, element-wise multiplication, adder, sigmoid and tanh evaluations. Our methodology first determines the available memory space on FPGA and divides it into several memory access blocks (MABs) that can be loaded with pre-computed data. Subsequently, a data flow graph (DFG) is developed to determine the computational nodes and their dependency. Then, when possible, we fuse the nodes in DFG and arrange them according to their expected power consumption. Based on the availability of MABs, we map the nodes into a single MAB or a group of them. Our methodology is characterized

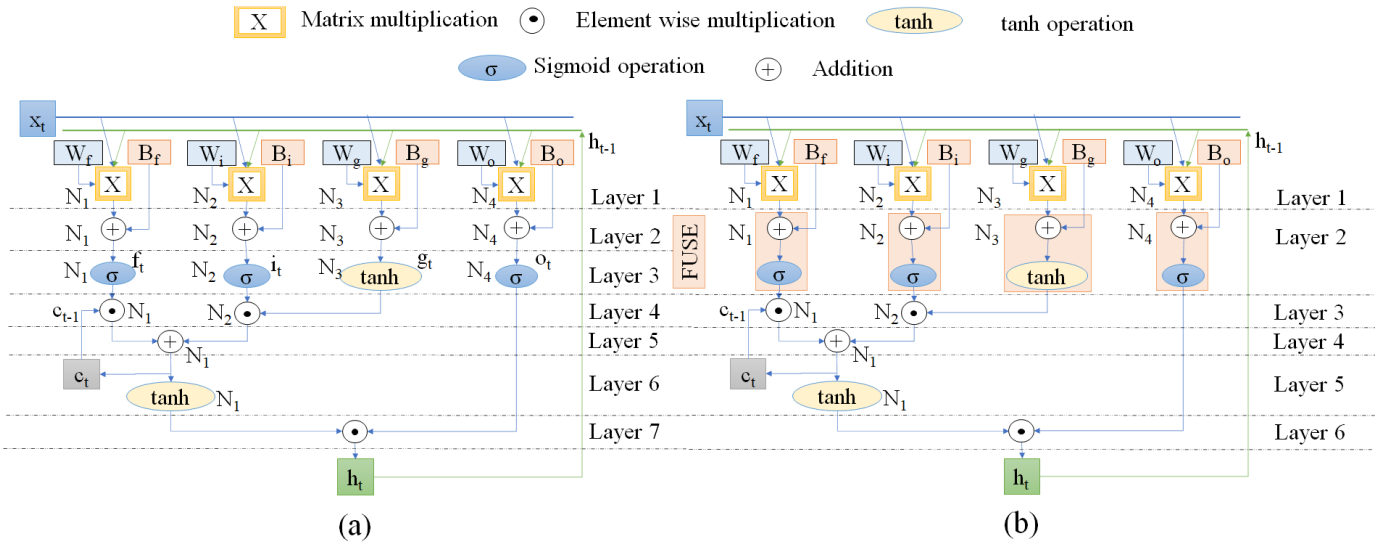

Fig. 1: (a) LSTM architecture. (c) LSTM architecture after node fusion.

by different-sized LSTMs and compared to state-of-the-art FPGA-based computing approaches. Low latency, low power consumption, and high energy efficiency obtained in our experimental results demonstrate the suitability of our methodology for FPGA-based edge computing platforms.

The key contributions of our work are as follows:

- We have proposed a memory-based inference approach where unused EMBs on FPGA can be utilized to map computationally intensive operations using overlay. We have specifically demonstrated the mapping and execution of LSTM on FPGA using the proposed mechanism; however, presented concepts can be extended to other complex neural networks such as Transformers.
- Our experiments demonstrate a significant reduction in resource, latency, and power, with enhancement in energy efficiency than existing approaches on FPGA-based LSTM execution.
- By not relying on model pruning, our methodology does not compromise accuracy under constrained latency and power budget.
- Providing more significant insights on the proposed method, we also illustrate how energy efficiency varies for the availability of EMBs for the 16-bit LSTM model.

The paper is organized as follows. Section II presents a brief background and motivation. The proposed mechanism is presented in Section III. Section IV discusses the implementation and results. Section V concludes.

II. Background and Motivation

A. Fundamentals of LSTM

LSTM is a type of recurrent neural network (RNN), initially proposed in [13]. LSTM comprises memory cells to store information regarding long-term dependencies in time-series data. The generic architecture of LSTM is shown in Fig. 1(a). The basic equations of LSTM are:

$$i_t = W_i(x_t, h_{t-1}) + B_i \tag{1}$$

$$f_t = W_f(x_t, h_{t-1}) + B_f \tag{2}$$

$$g_t = W_g(x_t, h_{t-1}) + B_g \tag{3}$$

$$o_t = W_o(x_t, h_{t-1}) + B_o \tag{4}$$

$$c_t = sigmoid(f_t) \odot c_{t-1} + sigmoid(i_t) \odot tanh(g_t) \tag{5}$$

$$h_t = sigmoid(o_t) \odot tanh(c_t) \tag{6}$$

The gates i, f, g, o control the flow of information inside LSTM. Input gate i determines the elements that will enter LSTM's memory cells, while the forget gate f determines the elements which are no longer required to be remembered. The input modulation gate g decides whether the memory cell needs an update, while the output gate o determines the memory elements generated as output. W_i, W_f, W_g and W_o represent the respective weight vectors, while B_i, B_f, B_g and B_o represent the related biases. c_t is the cell's output, while h_t is the hidden state passed on to the next layer.

The fundamental computation in LSTM-based inference is matrix-vector multiplications, followed by vector additions, element-wise multiplication, adder, sigmoid and tanh evaluations.

TABLE I: Comparison of Related Works of LSTM on FPGA

	[8]	[9]	[10]	[11]	[12]	Proposed
Power Reduction*	Yes	Yes	Yes	Yes	Yes	Yes
Latency Reduction*	Yes	Yes	Yes	Yes	Yes	Yes
Unused EMB Utilization	No	No	No	No	No	Yes
Result Accuracy	No	No	No	No	No	Yes

*Detailed report on power and latency is given in Table II.

B. Related Works on FPGA-based LSTM Inference

Several prior works have discussed the FPGA-based implementation of LSTMs. Authors in [8] discussed several compression schemes to reduce LSTM's model size and the necessary matrix multiplications for inference, thus allowing the models to be processed under limited storage and computing resources. They also proposed an efficient hardware architecture to accelerate the compressed LSTM model. A C-LSTM model was proposed by the authors of [9] to minimize the model parameters by approximating typical weight matrices as block circulant matrices (BCM). A pruning mechanism was applied by the authors of [10] to compress large LSTMs while ensuring high prediction accuracy. Authors in [11] proposed an E-RNN model that facilitates training via deriving BCM-based RNN representation to improve the performance and energy efficiency of the system. A bank-balanced pruning approach was proposed by the authors of [12] to implement LSTM on FPGAs, where each weight matrix row was divided into banks to exploit parallelism for low latency inferences. Likewise, authors of [14] proposed a latency-hiding model based on column-wise matrix-vector multiplication.

Table I provides a comparison of the related works. While model pruning has been employed as a primary technique, the rigid physical structure and processing flow of a typical accelerator (such as FPGAs) makes it challenging to exploit weight sparsity and doesn't lead to proportional benefits in energy efficiency and latency. Several other prior works have focused mainly on minimizing data transfer between on and off-chip memories so that the adverse impact of limited read/write bandwidth can be mitigated for LSTM inferences. While these techniques complement our propositions, this paper mainly focuses on methodologies for efficient mapping of LSTM on FPGA and exploiting memory-based inference to minimize the necessary workload.

C. Related Works on Memory-based Computing

A typical neural network processing involves computing the product of high-dimensional input vector and weight matrices and evaluating activation functions such as tanh, sigmoid, logarithm, *etc.* at millions of neurons, which is overwhelming for a low resource FPGA [7]. Traditionally, the coordinate rotation digital computer (CORDIC) approach [15] or Taylor series expansion [16] has been used to map transcendental functions for FPGAs but incurs a significant computing resource. Additionally, such models are associated with high latency and high-power consumption and, thus, are not energy efficient. For computing with memory, a look-up table-based approach is used where a 2D memory array stores all outputs for a functional operation. However, such a technique is unsuitable when the memory size grows exponentially, increasing input operand bit width. To mitigate this, a decomposition technique with multiple memory access can be used [17]. In this, a table of size 2^{a+b} is reduced to two tables, one of size 2^a and another of size 2^b. Such a strategy was adopted in [18]. However, the work considered only application-specific integrated circuit (ASIC) platforms and did not focus on FPGA platforms. Moreover, these works have not concentrated on mapping LSTM or other neural architectures on FPGAs. Nor have these works presented a detailed characterization of how a system's energy efficiency varies with available EMBs.

Comparatively, our methodology pre-computes computationally intensive operations in the LSTM model and accesses the results in runtime. Since underutilized memory units of FPGA are utilized, a significant reduction in latency and power consumption can be achieved by maximizing resource efficiency, as demonstrated in our results. Additionally, unlike in-memory inference approaches [19], [20], our approach doesn't require the custom design of memory cells and peripherals, thus, can be readily deployed on off-the-shelf FPGA hardware for broad utility.

D. Motivation

For typical implementations, significant overhead is associated with *tanh* and *sigmoid* operations in hardware. However, the most critical bottleneck for efficient performance is the matrix-vector multiplications with complex data dependencies in LSTM. Hence, if we can pre-compute these and store them in the MABs, the results can be accessed at runtime. This will reduce the latency and power consumption for performing these computations, make the system more energy efficient, and provide high result accuracy.

In addition to this, getting accuracy in results is an additional objective. For real-time systems, along with deadlines, accuracy is essential. As discussed, prior works in the field adhere to pruning techniques to achieve energy efficiency. But in that course, they lose the accuracy of the system. This work is associated with speed and accuracy, as results are pre-computed. Hence, the speed of operations is ensured, and energy efficiency can be obtained without pruning, which provides an additional accuracy benefit.

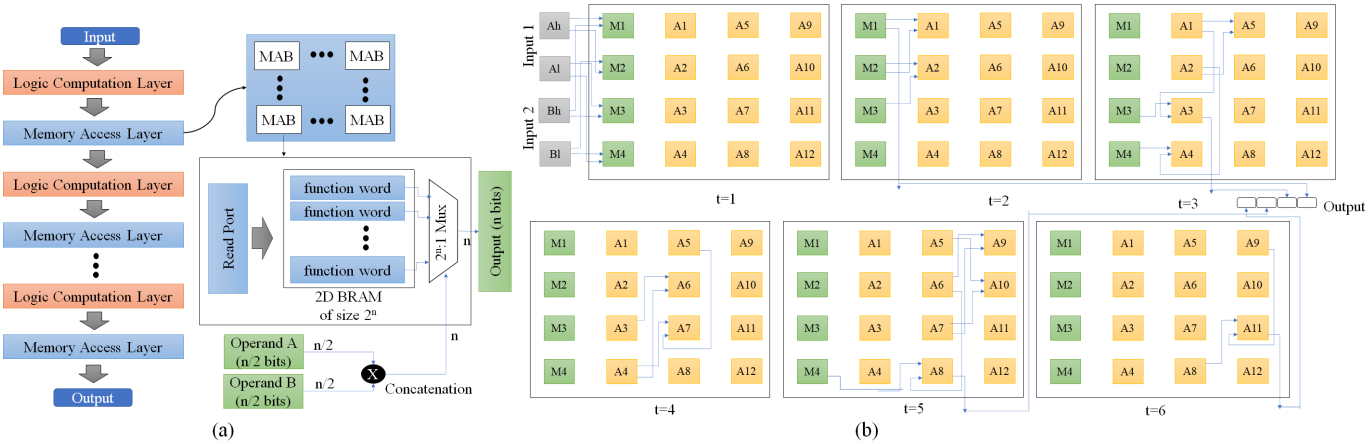

Fig. 2: (a) Architecture of memory-based inference of LSTM. (b) Memory access pattern to multiply two 8-bit operands.

III. Methodology for Memory-based Inference

A. Architecture for Memory-based Inference of LSTM

In Fig. 2(a), our proposed architecture for memory-based inference of LSTM involves several iterations of a logic computation layer (LCL) and a memory access layer (MAL). MAL consists of several memory access blocks (MABs). Each MAB comprises 2^n address spaces, with each address space consisting of a 2^m size function word. m and n vary with the type of FPGA and available memory. For the current work, we consider $m = n = 8$. Pre-computed data is loaded in the MABs, which can be retrieved via the read ports. If another application needs to be loaded, MABs can be re-loaded with a different data file via the read port. The MAB is accessed via an n-bit address line, which can be a combination of a set of operands, as shown in Fig. 2(a), where two operands of size $n/2$ access the MAB with n address spaces.

B. Mapping Methodology and Work Flow

1) Determining Available Memory Space and Formation of MABs: The initial step involves determining available memory space for a particular application and dividing it into a set of MABs, each comprising 2^n function words and each function word of 2^m bits. To perform this step, the initial design (without memory computation lookup) is mapped on the target FPGA. Some memory, which in the present case is FPGA BRAM, is utilized for basic operations. The remainder of memory is partitioned into MABs, with $m = n = 8$. The MABs are loaded with different data files per the target design.

2) Determine Data Flow Graph (DFG): The target application is initially portrayed as a data flow graph (DFG). DFG provides the various computational operations as nodes and the flow of data between the nodes as edges. DFG also provides information about the operations in each layer and related data flow among the nodes of different layers.

For LSTM, various computational nodes in each layer of operation and the data dependencies among the nodes are depicted in Fig. 1(b). DFG of an LSTM comprises seven operation layers, as depicted in Fig. 1(b). Four nodes in layer one are associated with matrix multiplications (MM), while the four nodes in layer two consist of addition operations. The third layer comprises four nodes, three sigmoid functions, and tanh operation. The fourth layer consists of two nodes that are associated with bitwise multiplications. A single node in the fifth layer comprises a tanh functionality, and the final layer is associated with a single node of bitwise multiplication.

3) Node Fusion: In our memory-based inference approach, a parent and a child node can be fused if their total number of operands is less than n bits. Thereby, in LSTM, nodes in the second and third layers can be linked, as no new operands are associated in the third layer. Hence, pre-computations can be performed with input operands for the parent nodes in the second layer, followed by their related child node operations in the third layer. This is depicted in Fig. 1(c).

4) Node Ordering: After all possible node fusion, computations in each node have to be mapped to one or more MABs. However, if the number of MABs is less, it is prudent first to map the computations that dissipate more power. Hence, the nodes need to be ordered as per their expected power dissipation (maximum to minimum). For LSTM, nodes associated with matrix multiplications consume the most power, followed by tanh, sigmoid, addition, and element-wise multiplication.

5) Node Mapping to MABs: A single MAB can be used for nodes whose total input bits are equal to or less than n. However, a series of MAB accesses is required for nodes whose total input bits are more than n. Our formulation for such memory access was presented earlier. For example, the multiplication of two four-bit inputs can be performed with a MAB of $n = 8$, i.e., contains $2^8 = 256$ address spaces.

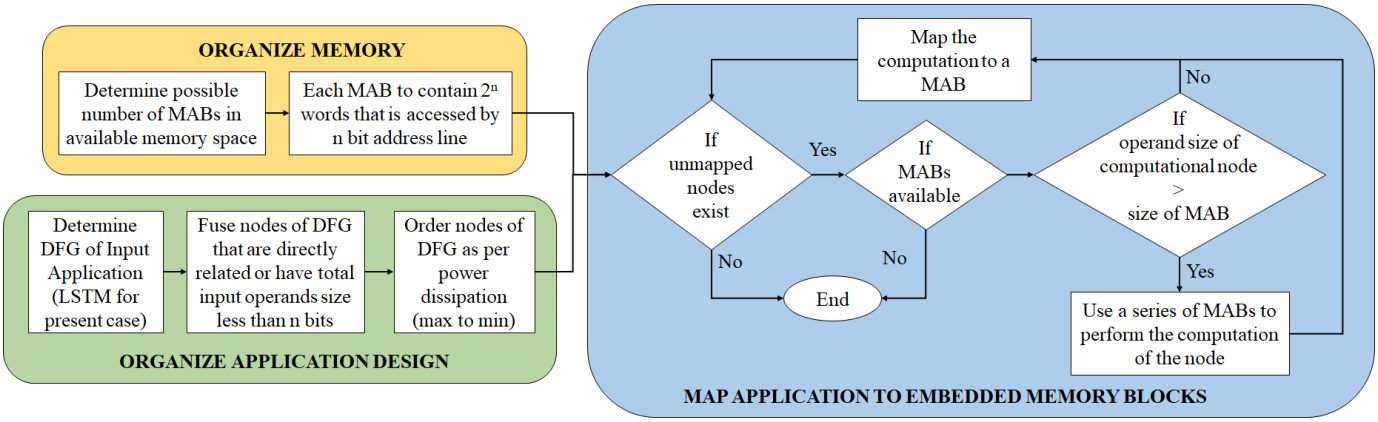

Fig. 3: Mapping and workflow in the proposed memory-based inference methodology.

However, for the multiplication of two eight-bit inputs, a series of 16 MABs of $n = 8$ needs to be performed [18]. Memory access for this scenario needs four multiplication and twelve addition MABs, as depicted in Fig. 2(b). While additions require a series of memory access per the size of operands, a single memory access is sufficient for tanh and sigmoid operations. A diagram of the workflow is provided in Fig. 3.

Note: In this work, we consider that the weights are fixed, based on which pre-computations are done and stored in memory. Moreover, if weights are stored outside memory and transferred during deployment, these may get leaked via covert channels during their transmission to the memory. Hence, storing the entire pre-computations with specific weights is better, and when weights are changed, the memory contents are re-loaded. Thus, if a different type of application needs to be loaded, the MABs can be re-loaded with a different data file via the read port. This prevents leakage of secret weights via covert side channels.

IV. Experimentation and Results

We have implemented the proposed mapping methodology as a software framework integrated into commercial FPGA tool flow and evaluated its efficacy for a set of parameters. We have considered LSTM for sequence classification.

A. Experimentation Strategy

We developed an FPGA overlay for LSTM based on the above methodology for experimentation. We adhere to the LSTM design of [21] to generate this overlay. Verilog codes are instantiated in Xilinx Vivado 2020.2 platform. This is targeted to ZynQ XCU104 Evaluation Board, operated at 200MHz clock frequency. For the present work, we consider $m = n = 8$ and divide the available memory space into MABs that contain 256 function words, each of size 256 bits. These MABs can be accessed via an 8-bit address line.

B. Result Analysis

1) Comparison of results with other works: Table II provides a comparison of our work with other existing techniques. The system's performance is evaluated based on the frequency of operation and per sample latency. Frames per second (FPS) are assessed as per Equation 7, where T_k represents the number of execution cycles for stage k. For the present work, per-sample latency and FPS are measured under the condition of sequentially processing a single input sample.

$$FPS = \frac{Frequency}{max\{T_1, T_2, ..., T_k\}} \tag{7}$$

The ratio of FPS estimates energy efficiency and the power consumed, i.e., FPS/W. As evident from the results tabulated in Table II, energy efficiency obtained via the proposed approach is better than all reference works, as accessing memory blocks reduces not only latency but also power consumption, which determines the energy efficiency of the system. In addition, unlike other works that rely on model pruning, our proposed method does not provide any degradation in result accuracy. Hence, the methodology is exceptionally suited for critical decision-making operations. Table III tabulates the resources utilized on ZynQ XCU104 when the referenced LSTM model is implemented in the proposed mechanism. While in the reference design, no BRAM usage is present, the proposed model heavily uses BRAM and reduces the utilization of LUTs. This paves the way for multiple design implementations on the same FPGA, where the other designs that do not use BRAMs can use the remaining LUTs.

2) Energy Efficiency vs. Availability of EMBs: In Fig. 4, we also analyzed how the system's energy efficiency varies with the availability of EMBs. The analysis is motivated since the necessary EMBs for all design functionalities may not always be available. Hence, it is necessary to characterize how energy efficiency under the proposed methodology varies with the availability of EMBs. For this, we analyzed a 16-bit

TABLE II: Comparison of Latency, Power and Energy Efficiency

	[10]	[9]	[12]	[8]	[11]	Implementation of [21]			Proposed		
Platform	XCKU060	Virtex 7	Arria 10 GX1150	Arria 10 SX660	Virtex 7	ZynQ XCU104			ZynQ XCU104		
Model Size	12 bit	16 bit	16 bit	8 bit	12 bit	4 bit	8 bit	16 bit	4 bit	8 bit	16 bit
Frequency (MHz)	200	200	200	200	200	200	200	200	200	200	200
Per Sample Latency (μs)	82.7	16.7	2.4	23.9	8.3	7.59	12.6	15.4	4.303	4.5	5.044
Accuracy Degradation(%)	0.3	0.32	0.25	0.02	0.31	0.0	0.0	0.0	0.0	0.0	0.0
Power (W)	41	22	19.1	15.9	25	5.02	17.18	21.16	0.606	1.33	4.32
Frames Per Second (FPS)	386941	179687	416666	334728	382510	263504	158730	129870	464792	425531	396510
Energy Efficiency (FPS/W)	9438	8167	21814	21052	15300	52490	9239	6138	766983	319948	91784

TABLE III: Comparison of Resource Utilization

	[21]			Proposed			LUT Reduction (%)		
	4 bit	8 bit	16 bit	4 bit	8 bit	16 bit	4 bit	8 bit	16 bit
Platform	ZynQ XCU104			ZynQ XCU104			ZynQ XCU104		
LUT	109	162	234	4	8	16	96.3	95.1	92.3
BRAM	0	0	0	10	56	136	-	-	-

LSTM and illustrated results graphically in Fig. 4.

As evident from the illustration, energy efficiency increases steeply in the initial phases, gradually slows down in the middle, and steadies at the end. This is because the high-power-consuming functionalities are mapped first; hence, a steep decrease in power increases energy efficiency. This is followed by mapping the low power-consuming functionalities and, in the end, the least power-consuming functionalities. Hence, the steep rise in energy efficiency gradually decreases and becomes steady at the end, where very low power-consuming functionalities are mapped.

Notably, our memory-based mapping methodology prioritizes high-power-consuming functionalities for mapping; therefore, under EMB-scarce mapping, judicious deployment of computationally intensive functions results in significant energy efficiency improvement with more EMB resources. As evident in the figure, our methodology can enhance the energy efficiency by 70× of a 16-bit LSTM model even when only 40% of necessary EMBs are available.

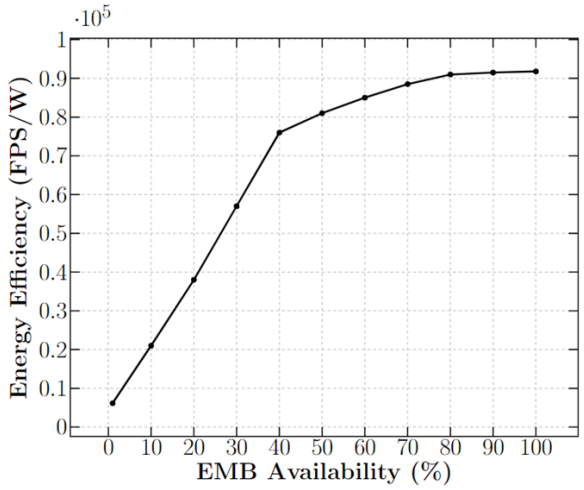

Fig. 4: Graphical analysis of energy efficiency for EMB availability for 16-bit LSTM.

V. Conclusion

Edge platforms' strict energy and resource budget has forced system designers to adopt model pruning to reduce power consumption and latency; however, the technique invariably compromises predictive accuracy. Comparatively, in this work, we have proposed an alternative *memory-based inference methodology* where unutilized memory units on FPGA can be exploited to store pre-computed results of fundamental computational primitives in a complex neural network model. The pre-computed results are accessed during inference as needed, thus minimizing the necessary workload and allowing higher performance inference without compromising the model's accuracy. Our methodology was demonstrated on LSTM. When implemented on the ZynQ XCU104 evaluation board, a 3× reduction in latency and 5× reduction in power was obtained, then the reference 16-bit LSTM model. The proposed methodology can be successfully applied in other complex networks. Complex networks are associated with several power-consuming operations like matrix multiplication, sigmoid, tanh, etc. For those cases, pre-computing the results and storing them in memory aids in minimizing high power dissipation and facilitates energy efficiency.

References

[1] G. K. Durbhaka and B. Selvaraj, "Predictive maintenance for wind turbine diagnostics using vibration signal analysis based on collaborative recommendation approach," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 1839–1842.

[2] A. Shylendra, P. Shukla, S. Bhunia, and A. R. Trivedi, "Analog-domain time-series moment extraction for low power predictive maintenance analytics," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022, pp. 9–12.

[3] S. Bharti and A. Mcgibney, "Privacy-aware resource sharing in cross-device federated model training for collaborative predictive maintenance," *IEEE Access*, vol. 9, pp. 120 367–120 379, 2021.

[4] X. Liu, J. Yang, C. Zou, Q. Chen, X. Yan, Y. Chen, and C. Cai, "Collaborative edge computing with fpga-based cnn accelerators for energy-efficient and time-aware face tracking system," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 1, pp. 252–266, 2022.

[5] T. Groléat, M. Arzel, and S. Vaton, "Stretching the edges of svm traffic classification with fpga acceleration," *IEEE Transactions on Network and Service Management*, vol. 11, no. 3, pp. 278–291, 2014.

[6] P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Review of hardware architectures for advanced encryption standard implementations considering wireless sensor networks," in *Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2007. Lecture Notes in Computer Science, vol 4599.*, 2007, pp. 443–453.

[7] A. Ghosh, S. Paul, J. Park, and S. Bhunia, "Improving energy efficiency in fpga through judicious mapping of computation to embedded memory blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 6, pp. 1314–1327, 2014.

[8] M. Wang, Z. Wang, J. Lu, J. Lin, and Z. Wang, "E-lstm: An efficient hardware architecture for long short-term memory," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 280–291, 2019.

[9] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-lstm: Enabling efficient lstm using structured compression techniques on fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18, 2018, p. 11–20.

[10] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[11] Z. Li, C. Ding, S. Wang, W. Wen, Y. Zhuo, C. Liu, Q. Qiu, W. Xu, X. Lin, X. Qian, and Y. Wang, "E-rnn: Design optimization for efficient recurrent neural networks in fpgas," 2018. [Online]. Available: https://arxiv.org/abs/1812.07106

[12] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 63–72. [Online]. Available: https://doi.org/10.1145/3289602.3293898

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997.

[14] Z. Que, H. Nakahara, E. Nurvitadhi, H. Fan, C. Zeng, J. Meng, X. Niu, and W. Luk, "Optimizing reconfigurable recurrent neural networks," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 10–18.

[15] W. Ligon, G. Monn, D. Stanzione, F. Stivers, and K. Underwood, "Implementation and analysis of numerical components for reconfigurable computing," in *1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403)*, vol. 2, 1999, pp. 325–335 vol.2.

[16] C. Brunelli, H. Berg, and D. Guevorkian, "Approximating sine functions using variable-precision taylor polynomials," in *2009 IEEE Workshop on Signal Processing Systems*, 2009, pp. 057–062.

[17] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Transactions on Computers*, vol. 54, pp. 319–330, 2005.

[18] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. Prajapati, M. A. Indovina, S. M. P. Dinakarrao, and A. Ganguly, "Look-up-table based processing-in-memory architecture with programmable precision-scaling for deep learning applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 263–275, 2022.

[19] S. Nasrin, D. Badawi, A. E. Cetin, W. Gomes, and A. R. Trivedi, "Mf-net: Compute-in-memory sram for multibit precision inference using memory-immersed data conversion and multiplication-free operators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1966–1978, 2021.

[20] P. Shukla, A. Shylendra, T. Tulabandhula, and A. R. Trivedi, "Mc2ram: Markov chain monte carlo sampling in sram for fast bayesian inference," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[21] G. Ahir, "Long short term memory (lstm) : Verilog implementation," 2020. [Online]. Available: https://github.com/ahirsharan/LSTMlong-short-term-memory-lstm—verilog-implementation