LSTM acceleration with FPGA and GPU devices for edge computing applications in B5G MEC

Dimitrios Danopoulos[1], Ioannis Stamoulias[1,2], George Lentaris[1], Dimosthenis Masouros[1], Ioannis Kanaropoulos[1], Andreas Kosmas Kakolyris[1], and Dimitrios Soudris[1]

[1] National Technical University of Athens, Greece
[2] National and Kapodistrian University of Athens, Greece
{dimdano,glentaris,dsoudris}@microlab.ntua.gr

Abstract. The advent of AI/ML in B5G and Multi-Access Edge Computing will rely on the acceleration of neural networks. The current work focuses on the acceleration of Long Short-Term Memory (LSTM) kernels playing a key role in numerous applications. We assume various LSTM sizes while targeting FPGA and GPU hardware for both embedded and server MEC purposes. Systematically, we perform a design space exploration to determine the most efficient acceleration approach and most suitable configuration for each device. We use High-Level-Synthesis to implement our proposed circuit architectures on Xilinx FPGAs, while we use high level tools for NVIDIA GPUs such as PyTorch's JIT compiler or ONNX runtime. Our exploration shows that the full parallelization of an LSTM array multiplication quickly overutilizes the FPGA, while on GPUs LSTM models can be deployed more easily. Instead, the best approach for FPGAs is to find a balance between parallelizing LSTM gates and vector multiplications. Our comparative study shows that FPGAs prevail in light LSTM models, whereas GPUs prevail in larger model topologies. Moreover, we show that far- and near-edge FPGAs achieve similar latency, however, near-edge GPUs can achieve one order of magnitude faster execution than far-edge GPUs. The best results range in 0.3-5msec latency per execution with acceleration factors in $12\times - 174\times$.

Keywords: 5G · forecasting · anomaly detection · LSTM · FPGA · GPU

1 Introduction

Emerging applications in the edge computing era require efficient AI/ML and high performance computing systems to process huge amounts of data at remote locations. Especially in the upcoming B5G/6G networks, AI/ML is expected to play a key role in the MEC domain, both for user applications and for infrastructure zero-touch management. Everyday scenarios, such as autonomous vehicles and smart city/factory operations, will rely on processing & decisions made closer to the location of data generation due to extremely low-latency requirements. Real-time analytics and forecasting will force the adoption of hardware

accelerators across the entire edge–cloud computing continuum with small and large devices, and especially GPUs and FPGAs.

The LSTM type of artificial neural network has achieved state-of-the-art classification accuracy in multiple useful tasks for MEC applications, such as the aforementioned forecasting, network intrusion detection, and anomaly detection [6]. Anomaly detection algorithms identify data/observations deviating from normal behavior in a dataset. Similarly, forecasting algorithms estimate what will happen in the future by using historical data. The LSTM provide solutions at the cost of increased compute and memory requirements, which makes their deployment challenging, especially for resource-constrained platforms, such as embedded FPGAs and GPUs. Furthermore, the internal mechanisms of LSTM networks make it more challenging to parallelize the computations in an acceleration platform as they require state-keeping in between processing steps. This creates data dependencies and often limits the parallelization degrees.

Recently, the research community has started deploying hardware accelerators of LSTM networks to increase the performance and energy efficiency of such computationally intensive tasks for data prediction on the aforementioned scenarios. Also, depending on the scenario, throughput or latency optimized systems need to be developed. Thus, it is important to embrace the heterogeneity paradigm in order to provide high performance systems, such as FPGAs and GPUs, which can cover a plethora of acceleration schemes and offload the general purpose CPUs when needed.

To study the computational aspects and the benefits of accelerating LSTMs in a practical fashion, the current paper presents our exploration of implementing LSTM kernels on FPGA and GPU devices for both far- and near-edge computing scenarios. The main contributions are:

1. Examine various LSTM structures from an acceleration point of view, i.e., assess parallelization and overhead benefits/costs for certain types of HW.
2. We extend our study for multiple LSTM kernels and devices, i.e., anomaly detection and timeseries forecasting, on multiple small and large FPGA/GPUs of distinct underlying architecture and CPU-to-device interfaces.
3. We perform a design space exploration (DSE) to determine the most efficient acceleration approach based on certain performance trade-offs and Quality of Service (QoS) preferences.

2 Background and Related Work

2.1 AI@EDGE with acceleration for MEC

The H2020 project *AI@EDGE* [7] aims to develop a "connect-compute" platform that efficiently creates and manages end-to-end network slices. This decentralized HW & SW platform will be placed inside broader MEC domain(s) to support a diverse range of AI-enabled applications and provide security/privacy, flexibility, and acceleration. In particular, *AI@EDGE* key underlying technologies include HW virtualization, multi-tier orchestration, serverless computing,

programmable pipelines to create and use trustworthy AI/ML models upon request, multi-connectivity disaggregated radio access, as well as multiple forms of programmable HW acceleration. The final AI/ML capabilities of the platform will serve both the closed-loop automation for the infrastructure/network (e.g., monitoring, zero-touch management) and also the applications at user level (e.g., download and execute certain AI functions upon user request).

Towards achieving the aforementioned goals, which combine acceleration with virtualization and easy-to-use (even serverless) computing, the selected approach was to exploit ubiquitous SW frameworks and diverse HW devices from dominant GPU and FPGA vendors. That is to say, e.g., instead of low-level FPGA programming with VHDL, we opted for high-level synthesis and TensorFlow-based end-to-end toolflows. Such an approach facilitates quick development and flexible function deployment via Docker containers, i.e., makes the platform attractive to users and facilitates the integration of multiple HW accelerators to an already complex distributed system. Representative devices in the "connect-compute" platform include Xilinx U280 FPGA [16] and NVIDIA V100 [12] for near-edge nodes (server-class computing), as well as Xilinx Zynq MPSoC [19] and NVIDIA Jetson AGX Xavier [10] for far-edge nodes (embedded computing). Representative toolflows include Xilinx Vitis AI and Vitis HLS for FPGAs [17, 18], as well as NVIDIA CUDA-X and ONNX for GPUs [11, 1].

2.2 LSTMs for Time Series Prediction

LSTM networks are a type of Recurrent Neural Network (RNN) that uses more sophisticated units in addition to standard units. The LSTM cell adds long-term memory with its special gates inside in order to solve problems that require learning long-term temporal dependencies. More specifically, a number of "gates" is used to control the information inside the memory, keeping, forgetting or ignoring it when needed. This very important in order to learn the long-term dependencies in sequences which have a long-term trend. Designing an optimal LSTM for Time Series Prediction can be challenging and it requires extensive hyperparameter tuning. For example, the number of LSTM cells used to represent the sequence can be crucial in achieving high accuracy.

Besides predicting future sequences in time-series, LSTMs can successfully detect anomalies in data. Constructed as autoencoders [2], the goal is to minimize reconstruction error based on a loss function, such as the mean squared error. Autoencoders are self-supervised learning models that can learn a compressed representation of input data. When coupled with LSTM layers in a Encoder-Decoder LSTM topology, it allows the model to be used to encode, decode and recreate the input sequence. LSTM autoencoder networks are widely used in many applications for real-time anomaly detection such as manufacturing, network intrusion detection systems and others [9] [14].

2.3 Related Work

To optimize the performance or power efficiency of LSTM networks, which cover a wide range of applications, a plethora of hardware accelerators has been proposed from the academia. These applications span from anomaly detection systems [5] such as network intrusion or timeseries forecasting. The following related work involves designs for similar accelerator platforms for relatively similar problems that cover our domain problem.

A plethora of FPGA implementations have been investigated such as [3] in which the authors accelerated an LSTM model on a Xilinx FPGA achieving 21× speed-up from the ARM Cortex-A9 CPU of the embedded SoC. Also, Chang et al. [4] presented three hardware accelerators for RNN on Xilinx's Zynq SoC FPGA for character level language model achieving up to 23× better performance per power than a Tegra X1 board. Additionally, in FINN-L [15] the authors presented a library extension for accelerating Bidirectional Long Short-Term Memory (BiLSTM) neural networks on FPGA. They performed a thorough DSE in terms of power, performance and accuracy and showed the throughput scalability on a Pynq and MPSoC FPGA board, although no latency metrics were presented. Last, Fowers et al. [8] presented the Brainwave NPU which achieved more than an order of magnitude improvement in latency and throughput over state-of-the-art GPUs on large RNNs at a batch size of 1. Concerning GPU implementations, there is fewer related work as they usually pose a smaller research problem as FPGA implementations. However, there are quiet a few implementations [20, 21] which focus on LSTM training on GPU platforms in order to reduce energy footprint and accelerate the training algorithm.

3 Methodology

Our primary purpose is to assess the complexity and efficiency of accelerating LSTM-based networks on GPU and FPGA devices in the context of a decreased/viable time-to-market cost (c.f. sec. 2.1). To this end, we devise an evaluation methodology combining a certain degree of optimization and exploration goals, which we summarize in the following steps:

1. Define a representative set of LSTM-based network structures based on numerous real-world applications and spanning a considerable range of computational demands (e.g., input size, units, layers).
2. Define a representative set of tools and devices, i.e., FPGA, GPU, and CPU, which provide today indicative acceleration results in major AI applications and cover a wide range of processing nodes in the edge–cloud continuum.
3. Propose and explore parallelization techniques as well as high level tools for end-to-end AI model deployment.
4. Prune the search space and implement the proposed designs to derive actual results regarding execution time, power and accuracy.
5. Compare the results and define the most promising implementation approach per LSTM and device, as well as assess the overall acceleration potential for LSTMs in this heterogeneous infrastructure.

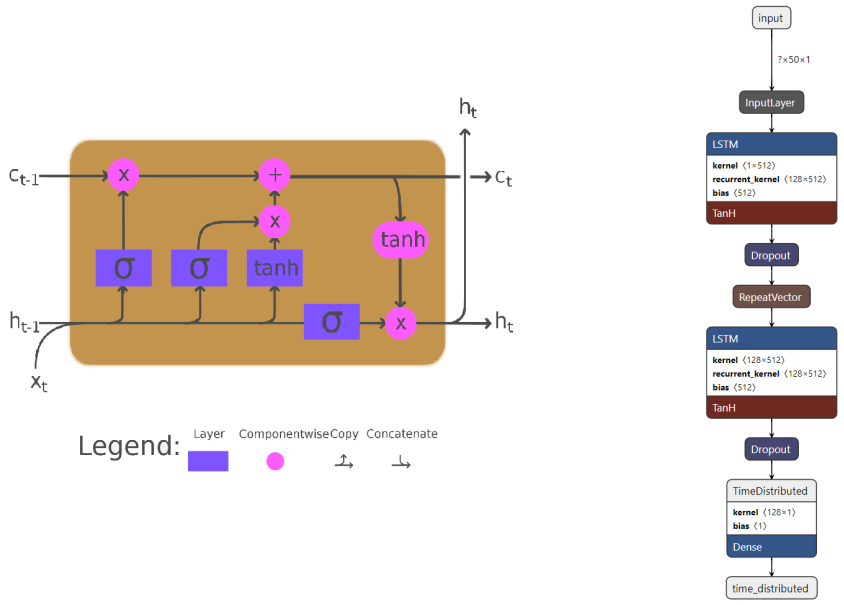

Fig. 1. LSTM illustration. Left: LSTM cell, Right: an LSTM topology used in the experiments

3.1 Representative LSTM networks

As a first step of our methodology, we consider a wide range of LSTM topologies as part of our DSE analysis. Each LSTM, depending on its parameters can have different acceleration potential on a particular hardware device. Several of the models used were built as an encoder-decoder structure targeting network intrusion detection systems. The main parameters of the encoder and decoder layers are the number of *units*, that can differ between the layers of the same LSTM, the number of the input and output *features*, and the number of *timesteps*, which is the same for all the layers of a LSTM. Both encoder and decoder layers have the same core logic usually coupled with a bias vector and an activation function afterwards. Each layer iterates $\times timestep$ times and cannot be parallelized due to data dependencies as the results of each time step are used in the next iteration and are also stored for the next layer. Along with the encoder-decoder LSTMs we investigated the use of encoder-decoder LSTMs with dense (or fully connected) layer at the end and also with single-cell LSTM models (targeting stock or temperature prediction applications). In Figure 1, at the left side we illustrate the core functions inside a typical LSTM cell. On the right of the figure, we show the architecture of one of the LSTMs used in the evaluation. Last, we summarize each LSTM and its characteristics in Table 1. All models were developed with Tensorflow Deep Learning library.

Table 1. LSTM characteristics

	LSTM characteristics				Model characteristics	
Model	***Layers***	***Timesteps***	***Features***	***Type***	***Params***	***Flops***
LSTM-Autoenc-1	4	2	60	Autoencoder	0.07M	0.3M
LSTM-Autoenc-2	4	2	80	Autoencoder	0.1M	0.54M
LSTM-Autoenc-3	4	2	159	Autoencoder	0.5M	2.10M
LSTM-Autoenc-4	4	2	230	Autoencoder	1.1M	4.4M
LSTM-Autoenc-5	2	50	1	Autoencoder	0.2M	20M
LSTM-Dense	2	30	1	LSTM+Dense	0.2M	12.3M
LSTM-cell	2	50	5	LSTM-cell	0.9M	62.7M

3.2 Acceleration Devices and Programming

We leveraged an heterogeneous hardware architecture as the range of applications especially in the cloud and edge domains is diverse and each device behaves differently depending on the scenario and application. AI@EDGE will employ a development model for AI accelerators based on High Level Synthesis for FPGAs (using Xilinx Vitis) and CUDA programming model for GPUs. It's worth mentioning that the acceleration flow for GPUs was done through ONNX format for model serialization (wherever supported). A similar tool from Xilinx, called Vitis AI, was also tested but the support for LSTMs is limited. FPGAs and GPUs as hardware platforms will be attached directly in servers or shared over the network in edge workloads. In Table 2 we list several popular devices from FPGA and GPU domains both for edge (top) and cloud (bottom) domains but we selected only some representatives from each domain that can cover the spectrum of deployment scenarios.

Table 2. List of FPGA and GPU devices in edge (top) and cloud (bottom)

	Device specifications					
Device	***Type***	***Memory***	***DSP***	***LUT***	***BRAM***	***CUDA cores***
MPSOC ZCU102	FPGA	4Gb	2520	600K	32Mb	-
MPSOC ZCU104 [✓]	FPGA	2Gb	1728	504K	38Mb	-
Jetson Nano [✓]	GPU	4Gb	-	-	-	128
Jetson Xavier NX [✓]	GPU	8Gb	-	-	-	384
Jetson Xavier AGX [✓]	GPU	32Gb	-	-	-	512
Alveo U50	FPGA	8Gb	6840	1182K	47Mb	-
Alveo U200	FPGA	64Gb	5952	872K	35Mb	-
Alveo U280 [✓]	FPGA	32Gb+8Gb	9024	1304K	72Mb	-
Nvidia P40 [✓]	GPU	24Gb	-	-	-	3840
Nvidia A30 [✓]	GPU	24Gb	-	-	-	3584
Nvidia V100 [✓]	GPU	32Gb	-	-	-	5120

4 Proposed Designs

4.1 FPGA acceleration

Our design space exploration, for accelerating a LSTM using FPGAs, started with decreasing the latency of the LSTM kernel for a single input execution. Then, we used multiple kernels to achieve higher throughput. The first decisions during our exploration were based on three characteristics of the LSTM algorithm. Those characteristics are the recursion and the increased number of multiply-accumulate operations required inside each layer (lstm cell) and the way the data are exchanged between the layers of a LSTM. The computationally intensive parts of the LSTM algorithm, in terms of required processing time and number of operations (determined by the number of units and features of the lstm cell), are the calculations of the four activation vectors. Parallelization at those calculations is necessary, especially for a low latency implementation. Next, we present a summary of the considered acceleration techniques.

1. Parallelization at the layer level, where multiple layers would process data in parallel. This parallelization technique could achieve a speed up of maximum the number of layers, but the number of layers of a LSTM is significantly lower than the number of units/features. Disadvantages in this implementation include increased resources, underutilized hardware and extra logic for synchronizing and buffering, thus it was not an ideal choice for acceleration.
2. Parallelization inside each lstm cell at the iteration level, where multiple time steps, for different input data, could be processed in parallel using a systolic array architecture. Some of the disadvantages using this technique were increased resources, less scalable architecture, and an implementation that does not reduces the latency.
3. Parallelization inside each lstm cell at the calculation of the activation vectors. Considering the resources of the targeted FPGA, the developer can choose to process in parallel one or more of the activation vectors and also parallelize each calculation internally.
4. For further decreasing the latency, a fixed-point instead of floating-point architecture can be used. This change reduces the required resources of the kernel, reduce latency and memory transfer overhead.

For our basic architecture (Fig.2) we implemented four parallel engines, one for each activation vector (i_t, f_t, $\widetilde{c}_t$, o_t) and inside each of those engines we parallelized the multiply-accumulate operations for each row of the matrix with the active vector. To achieve the desirable parallelization the active vector and the weight matrices were partitioned in their entirety and second dimension, respectively. The code at Fig.3 presents the high level description for parallelizing the the multiply-accumulate operations. Each time step is calculated sequentially inside the lstm cell. To create the required layers of the LSTM system we execute sequentially the same lstm cell with the necessary changes in the weight and bias values. Last, the weights and biases are read from the DDR memories at the beginning of a new layer execution, before the first iteration, using wide memory interfaces.

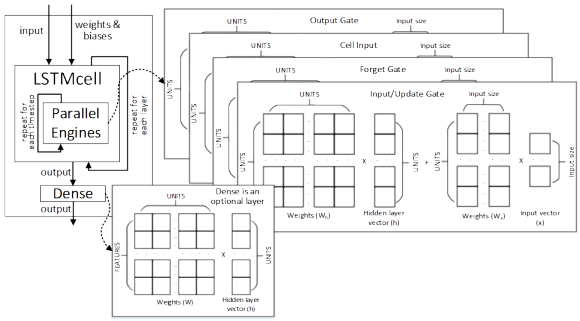

Fig. 2. Proposed LSTM architecture

```
dtype row_vector_mul(const dtype w[MX_COL],dtype h_x[MX_COL],dtype b){
   dtype2 res, first, temp;
   LOOP_MATRIX: for(int j=0; j<MX_COL; j++) {
      #pragma HLS LOOP_TRIPCOUNT min=mn_col max=mx_col
      #pragma HLS PIPELINE II = 1
      first = (j==0) ? (dtype2)b : res;
      temp = (dtype2)w[j] * (dtype2)h_x[j];
      res = first + temp;
   }
   return res;
}
void gate(dtype W[MX_UNITS][MX_COL],dtype b[MX_UNITS],
          dtype h_x[MX_COL],dtype out[MX_UNITS],int type){
   #pragma HLS ARRAY_PARTITION variable=W dim=2 complete
   #pragma HLS ARRAY_PARTITION variable=b complete
   #pragma HLS ARRAY_PARTITION variable=h_x complete
   dtype2 act_fun[MX_UNITS];
   LOOP_ROW: for(int i=0; i<MX_UNITS; ++i) {
      #pragma HLS LOOP_TRIPCOUNT min=mn_un max=mx_un
      #pragma HLS PIPELINE II = 1
      act_fun[i]=row_vector_mul(W[i],h_x,b[i]);
      if(type==0) out[i] = sigmoid(act_fun[i]);
      else        out[i] = tanh(act_fun[i]);
   }
}
```

Fig. 3. Parallelization of the activation vector

4.2 GPU acceleration

In the case of GPUs, we follow a more straightforward approach. We employ the PyTorch [13] open source machine learning framework, as the backbone for developing our DNN models. To leverage the GPU capabilities, we utilize PyTorch's optimized, built-in libraries that allow the effortless execution over the accelerator. Moreover, we utilize PyTorch's Just-In-Time (JIT) compiler and ONNX format for model serialization (wherever possible). With these runtimes on Nvidia GPUs we ensured a more optimal way to deploy our LSTM models into the GPU architecture which provided a significant increase in the speed of the network inference compared with the default Pytorch implementation.

5 Evaluation

5.1 Resources and Accuracy

In general, we focused on resource re-use techniques for the FPGA design where we could do fine grain optimizations. For example, we are re-using the same LSTM cell for each subsequent layer of each LSTM model. During the first call of the kernel all the weights and biases are transmitted to the DDR memories of the FPGA device and are read from the kernel whenever is necessary.

Table 3 presents the resource utilization percentages and latency considering implementations on the Alveo U280 device, for the network intrusion detection system. The floating-point (FP) implementation requires 39.07% of the LUTs, 26.66% of the registers and more than half of the DSP blocks (57.55%) and can process a single input in 3.38ms. Utilizing the same kernel with a floating-point interface but moving to an internal 16bit fixed-point arithmetic we can achieve a reduction of 28% in LUTs, 22% in registers and 45% in DSP blocks and reduce the execution time to 2.14ms. Simply by changing the interface to fixed-point we can further reduce the execution time to 1.20ms, with almost the same resource utilization. Moving to 8, 6, and 4 fractional bits the accuracy drops to 95.17%, 94.19%, and 86.41% respectively. Thus, 16bit arithmetic provides a good trade off between accuracy, resources and performance. Also, 512-bit memory interface was used between kernels and DDR memory, allowing us to reduce even further the latency of the LSTM kernels. Last, towards our DSE we utilized multiple engines (row 4) for the LSTM kernel parallelizing further the process and multiple LSTMs (rows 5-7) for increasing throughput.

Table 3. Resource utilization and latency for LSTM-Autoenc-5 model on U280 FPGA

	LUT	LUT Mem	REG	BRAM	DSP	Time
U280 resources	1304K	590K	2607K	2016	9024	
FP	39.07%	25.25%	26.66%	2.26%	57.55%	3.38ms
16bit fixed(float I/O)	10.63%	11.88%	4.36%	1.10%	12.18%	2.14ms
16bit fixed (fixed I/O, 512bit interface)	15.05%	6.38%	6.49%	2.26%	16.44%	0.72ms
16bit fixed (fixed I/O, 512bit interface, x2 engines)	21.47%	6.43%	9.08%	2.43%	28.60%	0.66ms
16bit fixed(fixed I/O, 512bit interface, 32 batch input, 4x LSTMs)	35.49%	24.66%	12.74%	3.58%	52.99%	7.31/32 = 0.23ms
16bit fixed(fixed I/O, 512bit interface, 64 batch input, 4x LSTMs)	35.49%	24.66%	12.74%	3.58%	52.99%	14.04/64 = 0.22ms
16bit fixed(fixed I/O, 512bit interface, 128 batch input, 4x LSTMs)	35.49%	24.66%	12.74%	3.58%	52.99%	27.37/128 = 0.21ms

Next, Table 4 presents the resource utilization percentages and the execution times considering implementations for the embedded ZCU104 FPGA device. This is a smaller device from the cloud Alveo U280 in terms of resources, hence the parallelization factor is diminished and the design space is a bit narrower. Due to the decreased parallelization the floating point kernel required 443.61ms for processing one input. However, the fixed-point implementations could fit in the device with the same parallelization we had for the alveo U280 board.

Table 4. Resource utilization and latency for LSTM-Autoenc-5 model on MPSoC ZCU104 FPGA device

	LUT	LUT Mem	REG	BRAM	DSP	Time
ZCU104 resources	230K	102K	460K	624	1728	
FP (parallelization/8)	73.26%	70.49%	35.20%	18.97%	28.47%	443.61ms
16bit fixed (float I/O)	37.50%	36.57%	14.63%	17.67%	63.60%	1.41ms
16bit fixed (fixed I/O)	56.21%	69.03%	22.71%	17.67%	63.60%	1.18ms

Our second case study is a group of LSTMs for anomaly detection targeting `LSTM-Autoenc-1` through `LSTM-Autoenc-4`. The LSTMs for the anomaly detection use a 4-layer model with two encoders and two decoders. All models require 2 time steps to complete the process as also presented in Table 1. Table 5 presents the resource utilization percentages and the execution times considering floating-point and fixed-point implementations for the LSTM kernels 1, 2, 3 and 4, targeting the U280 Alveo FPGA device. As expected, the resources and the achievable execution times are increasing moving from models with lower number of units and features to model with higher number of units and features. Moving to a 16bit fixed-point implementation we can see again a significant drop in the resource requirements across all kernels and in latency.

Table 5. Resource utilization and latency for LSTM autoencoders on U280 device

	LUT	LUT Mem	REG	BRAM	DSP	Time
U280 resources	1304K	590K	2607K	2016	9024	
FP LSTM-Autoenc-1	15.20%	4.66%	11.10%	2.09%	24.92%	0.44ms
FP LSTM-Autoenc-2	19.55%	6.16%	14.34%	2.09%	32.90%	0.55ms
FP LSTM-Autoenc-3	40.81%	21.31%	26.03%	0.28%	64.39%	2.31ms
FP LSTM-Autoenc-4*	40.23%	52.21%	9.86%	46.42%	12.47%	4.35ms
16bit fixed LSTM-Autoenc-1	3.89%	2.08%	1.55%	1.10%	5.35%	0.37ms
16bit fixed LSTM-Autoenc-2	5.31%	2.70%	2.21%	2.09%	6.95%	0.51ms
16bit fixed LSTM-Autoenc-3	9.74%	9.68%	1.99%	0.28%	13.12%	1.20ms
16bit fixed LSTM-Autoenc-4	17.69%	18.38%	2.60%	0.28%	18.79%	2.40ms

Table 6 presents the resource utilization percentages and the execution time considering a floating- and a fixed-point implementation of `LSTM-Autoenc-1`, targeting the MPSoC ZCU104 FPGA device. As we can see even in a device

that can be in a far-edge node we can have the same expected execution times as for those in a near-edge node.

Table 6. Resource utilization and latency for LSTM Autoencoder on ZCU104 FPGA

	LUT	LUT Mem	REG	BRAM	DSP	Time
ZCU104 resources	230K	102K	460K	624	1728	
FP LSTM-Autoenc-1	60.28%	24.76%	35.10%	16.38%	68.23%	0.61ms
fixed LSTM-Autoenc-1	20.30%	12.04%	7.41%	7.41%	27.95%	0.44ms

5.2 Inference performance on Edge & Cloud GPUs

We evaluate the inference time of a single-element batch with respect to the examined LSTM models defined in Table 1 and the various GPU accelerators considered, as presented in Table 2. Table 7 shows the respective results, which also reveal three major insights.

Table 7. Inference time for various LSTM models in each GPU device

	GPU Devices					
Model	Nano	NX	AGX	V100	P40	A30
LSTM-Autoenc-1	1.16ms	0.74ms	0.80ms	0.22ms	0.48ms	0.25ms
LSTM-Autoenc-2	1.18ms	0.69ms	0.80ms	0.23ms	0.49ms	0.26ms
LSTM-Autoenc-3	1.20ms	0.72ms	0.92ms	0.27ms	0.49ms	0.29ms
LSTM-Autoenc-4	1.21ms	0.75ms	0.93ms	0.31ms	0.49ms	0.33ms
LSTM-Autoenc-5	8.30ms	2.61ms	3.16ms	0.84ms	1.59ms	1.04ms
LSTM-Dense	5.42ms	1.69ms	1.89ms	0.57ms	1.05ms	0.64ms
LSTM-Cell	8.37ms	2.42ms	2.41ms	0.88ms	1.62ms	0.98ms

First, we observe that scaling the number of input features has minimal impact on performance. Indeed, in the case of the first four (`LSTM-Autoenc-1` - `LSTM-Autoenc-4`) examined models, where the number of features scale from 60 up to 230, we notice a negligible degradation in performance, both for Edge and Cloud devices. Second, we see that the number of timesteps is the major performance bottleneck of LSTM-based models. This observation was expected, since as shown in Table 1, models with higher amount of timesteps also reveal a total higher number of FLOPs. However, the imposed performance degradation is not proportional over all the devices. We observe that for less powerful devices, i.e., Jetson Nano and Xavier NX the overhead of adding more timesteps is greater, reaching up to 8× and 4× slower execution respecitvely (`LSTM-Autoenc-5` vs `LSTM-Autoenc-4`). On the other hand, AGX and Cloud devices are less affected, with an average slowdown of 3×. Last, as expected, there is a significant performance boost between edge and cloud devices, with up to 10× speed-up in certain cases (`LSTM-Autoenc-5`). This showcases the potential of model offloading from the edge to the cloud for faster execution.

5.3 Acceleration performance

Towards evaluating the aforementioned LSTM models across several FPGA and GPU devices we summarize the potential of each device in this paragraph. Depending on the scenario of the application and the LSTM model topology the most efficient accelerated kernels will be placed inside broader MEC domain(s) of the "connect-compute" platform of *AI@EDGE*. For example, for the `LSTM-Autoencoder-5` the latency on the cloud V100 GPU (0.84ms) is close to the cloud U280 FPGA (0.66ms without batch). The latency in a typical server CPU, specifically Intel Silver 4210, it is 18.8ms thus the achievable acceleration is $22.4\times - 28.5\times$. For a larger model (`LSTM-Autoencoder-4`) the typical latency in this server CPU is 54ms thus the acceleration becomes $12.4\times - 174\times$ with the upper value coming from the V100 device. In the same scenarios but for the edge domain, we have the NX GPU and ZCU104 FPGA compared with a typical embedded CPU such as the ARM Cortex-A53. The first aforementioned model has a latency of 30ms while the second larger model has a latency of 100ms in ARM A53. The acceleration from the edge hardware platforms becomes $11.5\times - 25.4\times$ for the first model, with the upper limit achieved from the ZCU104 FPGA, while in the second model we achieved a speed-up $15\times - 133\times$. Overall, in small model architectures the FPGA implementation seems more promising while, usually in larger models, GPUs become more efficient.

6 Conclusion

In this work we investigated the acceleration of various LSTM models on multiple hardware platforms. The LSTM models represented real world applications such as network intrusion detection, anomaly detection, stock prediction or temperature forecasting. We covered a wide range of devices spanning from the edge to cloud assuming a flexible deployment method that will become part of the "connect-compute" platform of *AI@EDGE*. Through diverse experimentation and tuning of possible design space tradeoffs we showed that mid- to large-size LSTMs can benefit greately from HW acceleration, especially at the far-edge deployment (embedded devices). The best results range in 0.3-5msec latency per execution with acceleration factors in $12\times - 174\times$.

Acknowledgements Work partially supported by H2020 project "AI@EDGE" (g.a. 101015922). The authors would like to thank Stefano Secci et al., from CNAM Paris, for providing LSTM example kernels for anomaly detection apps.

References

1. Bai, J., Lu, F., Zhang, K., et al.: Onnx: Open neural network exchange. https://github.com/onnx/onnx (2019)
2. Bank, D., Koenigstein, N., Giryes, R.: Autoencoders. CoRR **abs/2003.05991** (2020), https://arxiv.org/abs/2003.05991
3. Chang, A., Martini, B., Culurciello, E.: Recurrent neural networks hardware implementation on fpga (11 2015)

4. Chang, A.X.M., Culurciello, E.: Hardware accelerators for recurrent neural networks on fpga. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–4 (2017). https://doi.org/10.1109/ISCAS.2017.8050816
5. Diamanti, A., Vilchez, J.M.S., Secci, S.: Lstm-based radiography for anomaly detection in softwarized infrastructures. In: 2020 32nd International Teletraffic Congress (ITC 32). pp. 28–36 (2020). https://doi.org/10.1109/ITC3249928.2020.00012
6. Ergen, T., Kozat, S.S.: Unsupervised anomaly detection with lstm neural networks. IEEE Transactions on Neural Networks and Learning Systems **31**(8), 3127–3141 (2020). https://doi.org/10.1109/TNNLS.2019.2935975
7. EU: H2020 project AI@EDGE, https://aiatedge.eu/ (2022)
8. Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., Heil, S., Patel, P., Sapek, A., Weisz, G., Woods, L., Lanka, S., Reinhardt, S.K., Caulfield, A.M., Chung, E.S., Burger, D.: A configurable cloud-scale dnn processor for real-time ai. In: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). pp. 1–14 (2018). https://doi.org/10.1109/ISCA.2018.00012
9. Homayouni, H., Ghosh, S., Ray, I., Gondalia, S., Duggan, J., Kahn, M.: An autocorrelation-based lstm-autoencoder for anomaly detection on time-series data. pp. 5068–5077 (12 2020). https://doi.org/10.1109/BigData50022.2020.9378192
10. Nvidia: Jetson AGX Xavier Developer Kit (2022), https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
11. Nvidia: NVIDIA CUDA-X (2022)
12. Nvidia: NVIDIA V100 TENSOR CORE GPU (2022), https://www.nvidia.com/en-us/data-center/v100/
13. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
14. Provotar, A., Linder, Y., Veres, M.: Unsupervised anomaly detection in time series using lstm-based autoencoders. pp. 513–517 (12 2019). https://doi.org/10.1109/ATIT49449.2019.9030505
15. Rybalkin, V., Pappalardo, A., Ghaffar, M., Gambardella, G., Wehn, N., Blott, M.: Finn-l: Library extensions and design trade-off analysis for variable precision lstm networks on fpgas. pp. 89–897 (08 2018). https://doi.org/10.1109/FPL.2018.00024
16. Xilinx: Alveo U280 Data Center Accelerator Card (2022), https://www.xilinx.com/products/boards-and-kits/alveo/u280.html
17. Xilinx: Vitis (2022), https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html
18. Xilinx: Vitis AI (2022), https://www.xilinx.com/developer/products/vitis-ai.html
19. Xilinx: Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit (2022), https://www.xilinx.com/products/boards-and-kits/zcu104.html
20. Zhang, X., Xia, H., Zhuang, D., Sun, H., Fu, X., Taylor, M.B., Leon Song, S.: h-lstm: Co-designing highly-efficient large lstm training via exploiting memory-saving and architectural design opportunities. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). pp. 567–580 (2021)
21. Zheng, B., Vijaykumar, N., Pekhimenko, G.: Echo: Compiler-based gpu memory footprint reduction for lstm rnn training. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). pp. 1089–1102 (2020)