Accelerating Recurrent Neural Networks for Gravitational Wave Experiments

Zhiqiang Que*, Erwei Wang*, Umar Marikar*, Eric Moreno†, Jennifer Ngadiuba†,
Hamza Javed‡, Bartłomiej Borzyszkowski‡, Thea Aarrestad‡, Vladimir Loncar‡, Sioni Summers‡,
Maurizio Pierini‡, Peter Y Cheung*, Wayne Luk*
† California Institute of Technology, Pasadena, CA, USA
‡ European Organization for Nuclear Research (CERN), Geneva, Switzerland,
{jennifer.ngadiuba, sioni.paris.summers, Maurizio.Pierini}@cern.ch
*Imperial College London, UK, {z.que, w.luk}@imperial.ac.uk

***Abstract*—This paper presents novel reconfigurable architectures for reducing the latency of recurrent neural networks (RNNs) that are used for detecting gravitational waves. Gravitational interferometers such as the LIGO detectors capture cosmic events such as black hole mergers which happen at unknown times and of varying durations, producing time-series data. We have developed a new architecture capable of accelerating RNN inference for analyzing time-series data from LIGO detectors. This architecture is based on optimizing the initiation intervals (II) in a multi-layer LSTM (Long Short-Term Memory) network, by identifying appropriate reuse factors for each layer. A customizable template for this architecture has been designed, which enables the generation of low-latency FPGA designs with efficient resource utilization using high-level synthesis tools. The proposed approach has been evaluated based on two LSTM models, targeting a ZYNQ 7045 FPGA and a U250 FPGA. Experimental results show that with balanced II, the number of DSPs can be reduced up to 42% while achieving the same IIs. When compared to other FPGA-based LSTM designs, our design can achieve about 4.92 to 12.4 times lower latency.**

I. Introduction

Recurrent Neural Networks (RNNs) are a type of architecture specialized for processing ordered data, for example time-series data. These networks have applications in speech recognition [1], DNA sequence analysis, and physics experiments [2, 3]. An exciting physics experiment concerns the detection of gravitational waves, predicted by Albert Einstein a hundred years ago. The first detected wave came from a collision between two black holes, reaching the earth after 1.3 billion years. The detectors at the Laser Interferometer Gravitational-Wave Observatory (LIGO) produce time-series data, as they capture cosmic events such as black hole mergers which happen at unknown times and of varying durations. Accelerating RNN inference using reconfigurable accelerators such as FPGAs would enable sophisticated processing, such as anomaly detection, to run in real time on the data stream from the detector and generate a fast response. Among the many RNN variants, the most popular one is Long Short-Term Memory (LSTM). FPGAs have been used to speed up the inference of RNNs/LSTMs [1, 4, 5, 6, 7], which offer benefits of low latency and low power consumption compared to CPUs or GPUs.

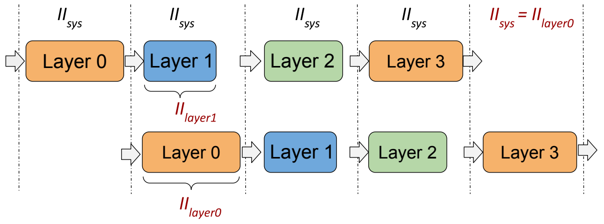

Fig. 1: Unbalanced layer IIs among various cascaded layers in an RNN model

However, existing LSTM accelerators cannot support low-latency and effective multi-layer execution, especially when targeting small LSTM models with requirements of ultra low latency and ultra high throughput for scientific applications. Many existing FPGA-based LSTM accelerators are designed with the same idea as their GPU counterparts, which utilize a single computational engine architecture where the engine is designed to run one block or layer at one time, and the whole network is processed by running the engine repeatedly [5, 6]. Their design consists of arranging computing resources to form a single core with many processing elements, leveraging data level parallelism. For example, Brainwave [5] is a single-threaded neural processing unit (NPU) which has 96,000 processing elements (PEs). However, when the size of the targeted LSTM layer is small, these hardware resources will not be fully utilized, e.g., when targeting a small LSTM layer, the Brainwave hardware utilization is lower than 1% [5], while the utilization of the NPU can be lower than 15% [6]. Moreover, since a single engine is used, the various layers must have the same amount of parallelism which is not flexible to take full advantage of the customizability of FPGAs. Thus, this work applies a layer-wise architecture to map all the LSTM layers on-chip and perform the computation for different layers on their own unit with independent optimization to achieve low latency and high system throughput.

Unlike CNN inference designs [8, 9] which only have forward datapaths and can be fully pipelined, there are feedback datapaths in RNN inference and data dependencies exist between the current timestep and the next timestep. Unrolling the timesteps fully may help, however the sequence length

This work has been accepted at the 2021 32nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP).

(timestep) of an LSTM model is usually larger than the number of layers [10], e.g., 1500 timesteps in an LSTM layer in DeepSpeech [11], which makes the full unrolling of timesteps impractical on FPGAs because of the limited hardware resources.

To accelerate an RNN model with multiple LSTM layers, this work proposes coarse grained pipelining with balanced II (initiation interval) to improve system throughput and reduce latency. This is achieved by identifying appropriate reuse factors for each layer, resulting in fast response and enhanced resolution for processing sensor data. It can achieve the best (smallest) system level II for a neural network with multiple LSTM layers on a given FPGA. The II is the number of clock cycles before a unit can accept new inputs and is generally the most critical performance metric in systems [12]. A perfect pipeline has $II = 1$ cycle, as this is required to keep all pipeline stages busy. However, the II of an LSTM layer is generally larger than one because of the data dependencies. For a model with multiple layers in sequence, the initiation interval of this model is decided by the largest II among all the layers [13], as shown in Fig. 1. The unbalanced IIs in various layers result in hardware inefficiency and low throughput. Accelerating a deep LSTM model is challenging since the computation load varies greatly among layers and data dependency exists both time-wise and layer-wise.

Our approach is to ensure all the layer IIs are balanced to eliminate system stall, so that the system becomes a coarse grained seamless pipeline. It increases pipeline parallelism by performing more computations without increasing latency, and without introducing additional memory traffic or storage. Unbalanced IIs in a pipeline is a common issue, but few studies address balancing IIs in the context of accelerating multi-layer DNNs, especially for RNNs/LSTMs. The proposed coarse-grained pipelining is similar to layer parallelism but the granularity in our approach does not need to cover an entire layer. An LSTM layer can still be divided into multiple blocks with pipeline parallelism. In addition, a customizable template for this architecture has been designed, which enables the generation of low-latency FPGA designs with efficient resource utilization using high-level synthesis (HLS) tools. Moreover, We develop an optimization algorithm such that, given the dimensions of the LSTM layers and a resource budget, computes a partitioning of the FPGA resources for an efficient

To the best of our knowledge, this is the first work to propose balancing IIs for a coarse-grained pipelined architecture to enable fast multi-layer LSTM data analysis in gravitational wave experiments. This work could help improve performance of next generation Gravitational Wave detectors.

We make the following contributions in this paper:

- A novel technique for balancing IIs of multi-layer LSTM inference to increase hardware efficiency and system throughput for data analysis in gravitational wave experiments.
- A scalable and low latency LSTM template which enables the generation of low-latency FPGA designs with efficient resource utilization by HLS tools. We open source the templates with some examples[1].
- A comprehensive evaluation of the proposed method and hardware architecture.

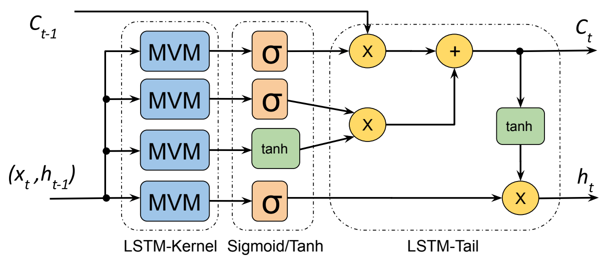

Fig. 2: Structure of an LSTM cell

The specific RNN layered structure and coefficients are LIGO specific, but the need for low latency would benefit many other applications, especially those requiring real-time response, e.g., low latency would benefit the Large Hadron Collider (LHC) physics [14], adaptive radiotherapy [15] and electronic trading [16]. The proposed techniques can be adapted to address these other applications.

II. Background and Preliminaries

RNNs/LSTMs have been shown to have useful properties with many significant applications. This study follows the standard LSTM cell [4, 5, 6]. 2 shows an LSTM cell. It consists of three main parts. At the front, there are four LSTM gates which perform matrix-vector multiplication (MVM), followed by activation functions. While in the tail, there are a few element-wise operations. The hidden state h_t, which will be fed back from the tail to the front, is produced by the following equations:

$$\begin{aligned} i_t &= \sigma(W_i[x_t, h_{t-1}] + b_i), & f_t &= \sigma(W_f[x_t, h_{t-1}] + b_f) \\ g_t &= \tanh(W_g[x_t, h_{t-1}] + b_u), & o_t &= \sigma(W_o[x_t, h_{t-1}] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t, & h_t &= o_t \odot tanh(c_t) \end{aligned}$$

Here, σ, $tanh$ and $\odot$ stand for the sigmoid function, the hyperbolic tangent function and element-wise multiplication respectively. i, f, g and o represent the input, forget, input modulation and output gate respectively. The input modulation gate is often considered as a sub-part of the input gate. The input vector and hidden vector are combined so that W represents the weight matrix for both vectors. Bias term is represented as b. The output c_t is the internal memory cell state and h_t is the output of the cell, also called the hidden vector, which is passed to the next timestep or next layer.

III. Design and Optimization Methodology

This section analyzes unbalanced II issues and introduces several optimizations for multi-layer RNN designs. We define a few parameters, as shown in Table I for later calculations.

[1]https://github.com/walkieq/RNN_HLS

TABLE I: System Parameters

II_{sys}	System initiation interval
TS	Timestep number
ii_N	Timestep loop initiation interval in the LSTM layer N
II_N	Initiation interval for layer N
LT_N	Latency of a single timestep loop for layer N
LT_α	Latency of the unit α; α could be mult / mvm / tail / σ
x_t	The input vector x at timestep t
h_t	The hidden vector h at timestep t
Wx	LSTM gates weight matrix for input vector.
Wh	LSTM gates weight matrix for hidden vector.
Lx	Number of elements in the input vector x
Lh	Number of elements in the hidden vector h
R_x	Reuse factor for MVM involving LSTM input vector x_t
R_h	Reuse factor for MVM involving LSTM hidden vector h_t
R_t	Reuse factor for LSTM tail unit

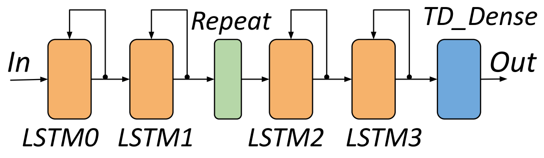

Fig. 3: Overview of the LSTM-based autoencoder

A. LSTM-based autoencoder for gravitational wave detection

Fig. 3 shows an overview of the LSTM-based autoencoder used for gravitational wave detection. The models and the dataset are available on GitHub [17, 18]. The autoencoder consists of two components, an encoder and decoder. The encoder learns to transform data from the input layer into a latent-space representation, which acts as a data "bottleneck". The decoder then reconstructs the output of the reduced latent representation as close as possible to its original input. When the error between input and reconstructed values is high, the input is flagged as anomalous. In this work, an LSTM-based autoencoder is used as an unsupervised prediction model to detect the anomalies for gravitational waves. This works by only training the LSTM-autoencoder to encode and decode normal background conditions at the LIGO interferometers. When an event containing a gravitational wave passes through the autoencoder, the model cannot encode and decode the additional strain provided by the gravitational wave. Both the encoder and decoder have two LSTM layers. A TimeDistributed dense layer is applied before the data output.

B. System II for multi-layer LSTM networks

Accelerating a deep LSTM model which has multiple layers is challenging since the computation varies greatly among layers and data dependencies exist both time-wise and layer-wise. An efficient technique to improve throughput and reuse computational resources is to pipeline hardware units. If each input can overlap with itself, we can achieve simultaneously inference parallelism within a run by coarse grained pipelining as shown in Fig. 1.

However, a naive implementation can result in a large number of idle cycles due to inter-layer dependencies since the pipeline is not seamless; a particular layer might stall until the previous layer finishes. The unbalanced IIs in various layers results in hardware inefficiency and low system throughput. Typically, the particular layer with the largest II should be optimized since it dominates the system II. Generally, the II cycles can be reduced if more hardware resources are allocated to that particular layer by adding more parallelisms. So the targeted layer should be allocated as many hardware resources as possible. However, the hardware resources on a given FPGA is limited, which means that the other layers may occupy less hardware resources. When the resources for a layer decrease, the II of that layer will increase. Then this layer may become the one that has the largest II and dominates the design. Thus, the optimal case is that all the layers have the same II, in which scenario the design utilizes the hardware resources efficiently and achieves the highest system throughput as shown in Fig. 4.

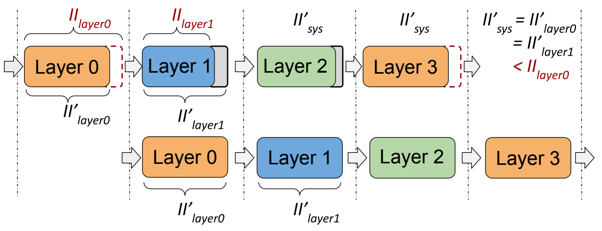

Fig. 4: Overview of the method used to balance IIs

Besides, we find that we do not need to unroll every unit in order to achieve the lowest II. Some hardware resources can be saved from the units which do not require full unrolling. And then these saved hardware resources can be reallocated to the other units which dominate the system to achieve low initiation intervals. As shown in Fig. 4, the hardware resources for layer 1 can be reduced so that the saved resources can be reallocated for layer 0. The II_{layer1} is increased to II'_{layer1} while the II_{layer0} which is the largest can be reduced to II'_{layer0} so that the final system II_{sys} can be reduced.

Partitioning FPGA resources to enhance throughput has been studied for CNNs [8, 9, 19, 20] but they do not touch the RNNs and the recurrent nature as well as the data dependencies in RNN computations, which are absent from CNNs. We develop an optimization algorithm such that, given the dimensions of the LSTM layers and a resource budget, computes a partitioning of the FPGA resources for an efficient and balanced high-performance design. Our algorithm runs in seconds and produces a set of reuse factors [14]. We then use these factors to parameterize an LSTM template design specified using HLS to form a complete multi-layer LSTM implementation. Since all the layers have the same II, we only need to focus on the optimization for a single LSTM layer. The layer II and system II are

$$II_N = ii_N \times TS \tag{1}$$

$$II_{sys} = \max(II_0,\ II_1,\ \ldots,\ II_N) \tag{2}$$

The original II_N should be $II_N = ii_N \times TS + (LT_N - ii_N)$. However, the extra $(LT_N - ii_N)$ cycles can be eliminated after using the rewind for Vivado_HLS *#pragma pipeline*.

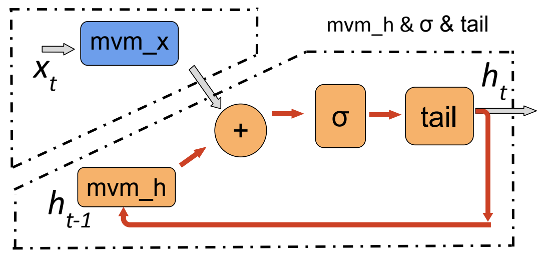

Fig. 5: An LSTM layer after performing the transformation

The rewind is an optional keyword that enables rewinding, or continuous loop pipelining with no pause between the end of one loop iteration and the start of the next iteration. So the proposed balancing method has two benefits. First, it improves throughput due to pipelining. Second, it reduces system latency since if the LSTM loop initiation interval, ii_N, can be reduced by 1 cycle, then the system latency can be reduced by TS cycles in total according to Equation (1).

C. The II of a single LSTM layer

This work splits one LSTM layer into two sub-layers. The first one is the mvm_x which has no data dependencies and performs MVM operations for the LSTM gates involving the input vectors while the second one includes all the others which form a loop with data dependencies, as shown in Fig. 5. For accelerating LSTM layers used for gravitational wave detection, the system is designed to achieve the average latency (system II) as small as possible. To achieve the lowest system II, fully unrolling the neural network model is an effective method which utilizes a multiplier only once in the computation of a layer. E.g., a fully connected (FC) layer with input size num_in and output size num_out can achieve the lowest latency if there are $num\_in \times num\_out$ multipliers. This is the most parallel and fast way a layer can be computed. It has been demonstrated in the HLS4ML based DNN designs for particle physics [14]. However, unlike forward computation in the FC layers used in the design of [14], there are data dependencies in LSTM computations.

After we have split the LSTM layer into two sub-layers, the two can be pipelined as shown in Fig. 6. According to the discussion in Section III-B, the optimal case is when the two sub-layers have the same II. Since the second sub-layer is complex and its II is usually larger than the one of the first sub-layer, the parallelism for the first sub-layer does not need to be as large as possible, resulting in a reduction of the number of multipliers needed to process the mvm_x unit. The saved multipliers can then be reallocated for other layers to achieve a lower system II. Reducing the parallelism of mvm_x does not hurt the system latency. Normally, each input vector can finish the calculation in the shadow region of processing the h_t because of the pipelining. Besides, the cycles for processing the first mvm_x can be eliminated when calculating the layer II because of the keyword of rewind in Vivado HLS.

While the second sub-layer may seem complex, if the design is split into more sub-layers, these sub-layers cannot be coarse grained pipelined. The reason is that the start of the next

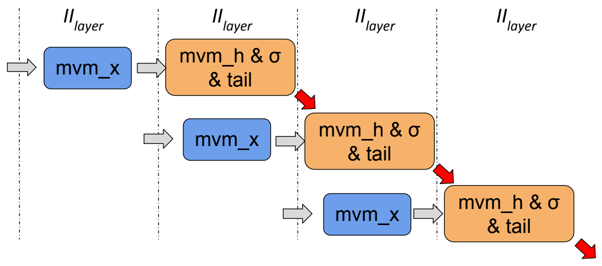

Fig. 6: Coarse grained pipelining in an LSTM layer

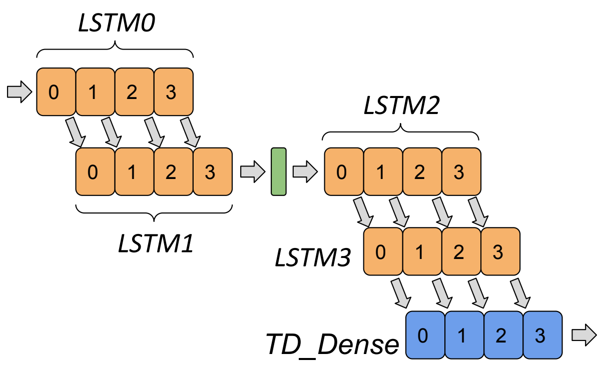

Fig. 7: Timestep overlapping

iteration needs the result from the current iteration, as shown by the red arrows in Fig. 6

D. Overlapping the computations in cascaded LSTM layers

In the proposed coarse grained pipelining, the processing of the cascaded LSTM layers can be overlapped. The second layer does not need to wait for the whole sequence of hidden vectors to be ready. Just one hidden vector from the former LSTM layer is sufficient to start the calculation of the next LSTM layer as shown in Fig. 7. It helps to reduce the overall system latency. It has to be noted that the LSTM2 can only start after the LSTM1 calculation is completed, since only the last timestep hidden vector is returned in LSTM1, which is decided by the structure of the autoencoder.

IV. Implementation

A. HLS implementation

This work maps all the layers on-chip and different layers run in a fashion of coarse grained pipelining to increase the system throughput. Besides, this work always seeks to achieve extremely low latency by utilizing as many hardware resources as possible. However, because of the data dependencies between different timesteps in LSTM calculation, the initiation interval is typically larger than 1. In this case, HLS will automatically increase the initiation interval until it can find a feasible schedule. For complex codes it is common to partition functionality into multiple modules, streaming data between them through explicit interfaces. Smaller components are more modular, making them easier to reuse, debug and verify. The effort required by the HLS tool to schedule code sections increases dramatically with a large number of operations that need to be considered for the dependency and pipelining analysis. Scheduling logic in smaller chunks is thus beneficial for compilation time and sometimes also for system latency. Our experiments show that inlining every function, especially the mvm_x and mvm_h in the LSTM gates , brings large II when the involved matrices are large.

The trade-off between latency, throughput and FPGA resource usage is determined by the parallelization of the inference calculation. This work adopts the reuse factor used in [14] to fine tune the parallelism, which is configured to set the number of times a multiplier is used in the computation of a module. In one extreme, all multiplications can be performed simultaneously using a maximal number of multipliers, while alternatively in the other extreme, one can use only one multiplier and perform the multiplications sequentially; between these extremes the user can fine tune algorithm throughput versus resource usage.With a reuse factor of one, the computation is fully parallel. With a reuse factor of R, $\frac{1}{R}$ of the computation is done at a time with a factor of $\frac{1}{R}$ fewer multipliers.

The total number of multiplications required to infer a given LSTM layer using 16-bit is:

$$DSP_{layer} = \frac{4 \times Lx \times Lh}{R_x} + \frac{4 \times Lh^2}{R_h} + 4 \times Lh \quad (3)$$

$$DSP_{model} = \sum_{layer=1}^{N} DSP_{layer} \leq DSP_{total} \quad (4)$$

Compared with the number of multipliers used in LSTM gates, the one required in the LSTM tail unit is small so the R_t is set to 1. Otherwise, $\frac{4 \times Lh}{R_t}$ should be used in Equation (3). Besides, since the LSTM cell status, c_{t-1}, is represented in 32-bit, the $f_t \times c_{t-1}$ in the LSTM tail needs two Xilinx DSPs to implement one multiplier. Thus, the LSTM tail unit consumes $4 \times Lh$ DSPs. The activation function sigmoid is implemented using BRAM-based lookup tables with a range of precomputed input values. The hyperbolic tangent function is implemented as piecewise linear function [21, 22] to reduce the latency. In the next subsection, we introduce our method for determining R_x and R_h with a given FPGAs.

B. Design space exploration

FPGA multipliers are pipelined; therefore, the latency of one MVM computation, LT_{mvm}, is approximately

$$LT_{mvm} = LT_{mult} + (R - 1) \times II_{mult} \quad (5)$$

where LT_{mult} is the latency of the multiplier, II_{mult} is the initiation interval of the multiplier, which is one cycle in this work. Equation (5) is approximate because, in some cases, additional cycles could be introduced for signal routing. Besides, the Vivado HLS tool will replace a multiplier by an adder when the corresponding weight is simple.

As we discussed in Section III, the optimal case is that the two sub-layers in an LSTM layer have the same II, which results in Equation (6).

$$II_{sublayer} = LT_{mvm\_x} = LT_{mvm\_h} + LT_\sigma + LT_{tail} \quad (6)$$

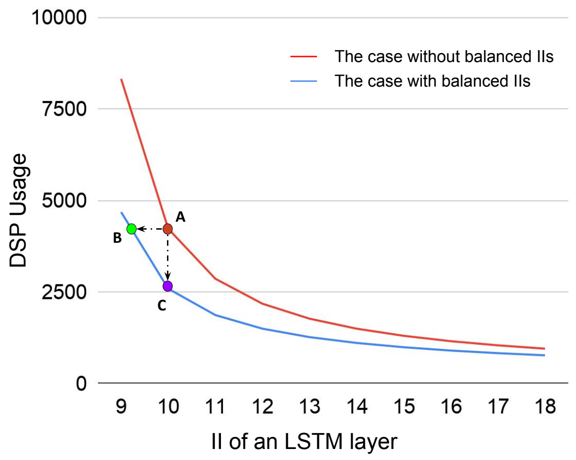

Fig. 8: Pareto frontier

where LT_{mvm_x} and LT_{mvm_h} are the latencies of the MVM units involving input vectors x and hidden vectors h respectively. LT_σ is the latency of the sigmoid function and LT_{tail} is the latency of the LSTM tail unit. These units are shown in Fig. 5. If we substitute the Equation (5) into Equation (6) and then we get

$$R_x = R_h + LT_\sigma + LT_{tail}. \quad (7)$$

The architecture designed in this section serves as a baseline to deploy our methodology, whose goal is to find Pareto-optimal sets of reuse factors of the proposed accelerator to achieve a good trade-off between our design objectives, which are hardware resources, energy, and performance. To achieve low latency, the reuse factors should be as small as possible since when they decrease the parallelism increases, leading to high throughput. However, when reuse factors decrease, the required hardware resources increase and may easily exceed the number of total hardware resources on an FPGA. If we substitute the Equation (7) and Equation (3) into Equation (4), we can get a quadratic inequality of R_h, which gives the minimum R_h for a given number of DSPs.

Fig. 8 illustrates the exploration results of an LSTM layer with $(Lx, Lh) = (32, 32)$ and different values of reuse factors, which are from 1 to 10. The red line represents the cases with the same R_x and R_h. The blue line shows the cases with balanced IIs, where R_x and R_h meet the constraint in Equation (7). For simplicity, LT_σ is set to 3 and the LT_{tail} is 5. Please note that LT_σ and LT_{tail} are both system dependent and can vary depending on clock frequency and FPGA devices. After balancing IIs, the Pareto frontier moves from red line to blue line. With the proposed technique, we can achieve a same II with less DSP usage (from point A to point C) or we can achieve a better II (from point A to point B) as shown in Fig. 8.

V. Evaluation and Analysis

This section presents the performance of the RNN models developed for gravitational wave detection on two generations of Xilinx FPGAs demonstrating the scalability of the proposed optimization.

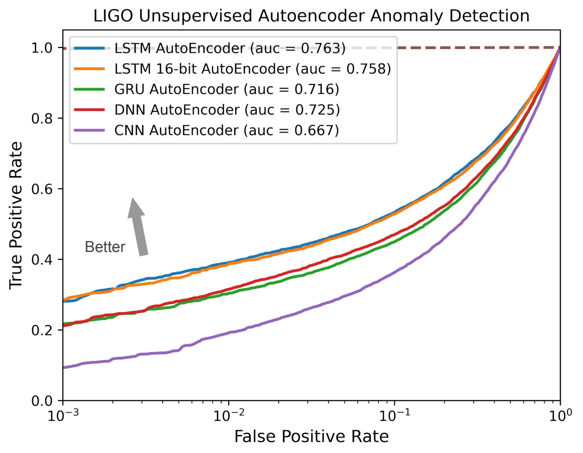

Fig. 9: AUCs and ROC curves for various autoencoders

A. *Experimental setup*

Simulated gravitational waves are generated using the GGWD library [23]. Noise is generated at a specified power spectral density (PSD) to mimic normal detector background conditions using PyCBC [24]. This approach to simulated data generation ignores glitches, blips, and other transient sources of detector noise, though this algorithm can be re-purposed for identifying these detector glitches with unsupervised methods. Signal events are generated simulating GW production from compact binary coalescences using PyCBC [24], which itself uses algorithms from LIGO's LAL Suite [25]. Signal events containing GWs were created overlaying simulated GWs, with the SEOBNRv4 Approximant, on top of detector noise. This provides an analogous situation to a real GW, in which the strain from the incoming wave is recorded in combination with the normal detector noise. Data are then whitened and band-passed, then normalized. The training set has 240K gravitational wave events. The validation set and test set have 60k and 50k events respectively. To study the performance and limitations of the proposed optimizations and hardware architecture, the designs are implemented using Vivado HLS 19.2. Two generations of Xilinx FPGAs, the ZYNQ 7045 and U250, are evaluated and compared with previous work.

B. *Model accuracy*

To quantify the performance of the autoencoders for anomaly detection implemented by various neural networks, we use the AUC metric, or area under the Receiver Operating Characteristic (ROC) curve, as shown in Fig. 9, with higher AUC corresponding to better performance. The default timestep [17] of 100 is used. AUC is a common metric for evaluating models as it is classification-threshold-invariant. The threshold for flagging an anomaly by its loss spike can be calculated by setting a false positive rate (FPR) on noise events. The higher the threshold for detecting an anomaly, the lower the FPR will be. This threshold can be used to calculate the corresponding true positive rate (TPR) on signal events. We observe that the LSTM-based autoencoder has the highest AUC, and hence the best performance, among the unsupervised designs [17] with various NN layers, including GRU, CNN and DNN. Additionally, Qkeras [26] is used to quantize the LSTM-based autoencoder to 16-bit. We find this precision to have a negligible effect on the NN performance.

TABLE II: Performance comparison of the FPGA designs

	Z1	Z2	Z3	U1	U2	U3
FPGA	Zynq 7045			U250		
DSP total	900			12,288		
R_h	1	2	1	1	1	4
R_x	1	2	9	1	9	12
LUT used	45k (21%)	45k (21%)	43k (20%)	449k (26%)	463k (27%)	516k (30%)
DSP used	1,058 (118%)	578 (64%)	744 (83%)	11,123 (91%)	9,021 (73%)	2,713 (22%)
ii_{layer} cycles	9	10	9	12	12	13
II_{layer} cycles	72	80	72	96	96	104

C. *Performance and efficiency comparison*

To illustrate the benefits of our proposed approach, two LSTM-based autoencoders are evaluated. The first one is a small autoencoder which has the same architecture as the one used in gravitational wave detection described in Section III-A but only has two LSTM layers, each having 9 hidden units. The results are shown in Table II. It is running at 100MHz with 8 timesteps. The weights and input are 16 bits. The bias and LSTM cell status are both 32 bits to keep the accuracy. To achieve the lowest latency, the reuse factors should be set to one so that all the operations are unrolled, e.g., the design Z1 in Table II. However the required number of DSPs exceed the one of the total DSPs on this FPGA. One may increase the re-use factor from one to two to fit the design into this FPGA device. However the cost is that now the timestep loop initiation interval, ii_{layer}, increases by one cycle which results in TS cycles increase for the layer II, e.g., the design Z2 in Table II. However, it is not necessary to fully unroll all units in order to achieve the lowest latency. Some hardware resources can be saved from the units which do not require full unrolling and can be allocated to the other units which are dominating to achieve low latency.

With the proposed balancing of IIs, some of the DSPs resources can be rearranged from implementing mvm_x to mvm_h to achieve lower latency, e.g., the design Z3. So this design can still achieve the lowest II like the case with full unrolling, and it is still able to fit in this FPGA device as shown in Table II, showing the benefits of balanced IIs. Besides, with heterogeneous reuse factors, the parallelism of the design can be fine-tuned to make the trade-off between latency, throughput and FPGA hardware resources as shown in Fig. 10. With the balanced II, the number of DSPs can be reduced up to 42% while achieving the same IIs.

Besides, to show the adaptability of our technique, the nominal autoencoder [17] developed for gravitational wave detection is implemented using a larger FPGA, U250, running at 300MHz with 8 timesteps. It has four LSTM layers which have

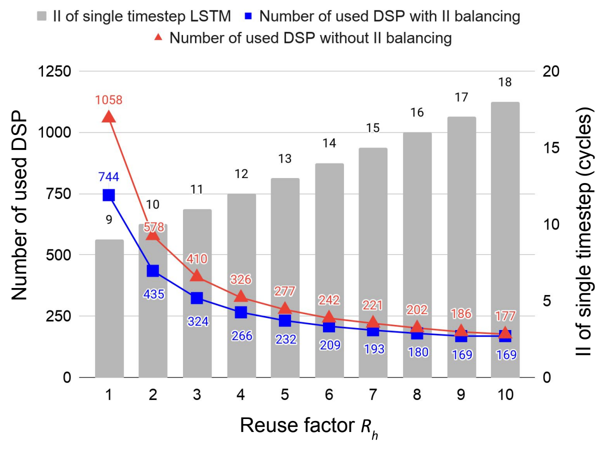

Fig. 10: Initiation intervals and DSP numbers using various reuse factor R_h on Zynq 7045

a number of hidden units equal to 32, 8, 8, 32 respectively and one TimeDistributed dense layer before the output. Since the U250 has 12,288 DSPs, the whole fully unrolled autoencoder can be fit into this FPGA with both R_x and R_h set to one, shown as the design U1 in Table II. With our technique of balancing IIs, the DSPs of the design U2 can be reduced by 2102 while achieving the same design IIs and same design throughput. After HLS synthesis, the II is slightly larger than the one estimated by the performance model since the DSP usage is very high and some additional cycles are incurred for signal routing. The design U3 is an interesting version with reuse factors (R_h, R_x) as (4, 12). It achieves a slightly worse II, as shown in Table II, however it consumes 3.3 and 4.1 times less DSPs than design U2 and design U1 respectively. Sometimes, the user may only care about the latency of the LSTM running on the FPGAs, then they can just take the point that gives them the lowest latency with most resources. However, if the user can bear with a slightly reduced latency then they can choose a smaller and cheaper FPGA as shown in Table II. One can choose between using less resources but increasing latency and vice versa. Please note because of the data dependence, the ii_{layer} could be hard to optimize to 1. However, it could be further optimized to a smaller value using fast multipliers or fast activation functions. We leave that for future work since it has a limited impact on the conclusions we draw from our study in this paper.

To compare the performance of the proposed design on FPGA with other platforms, we implement the same LSTM-based autoencoder on Intel CPU and NVIDIA GPU. The AVX2 vector instructions are enabled for the CPU while the CuDNN libraries are enabled for the GPU. Compared with the designs running on CPU and GPU, our FPGA design runs much faster, as shown in Table III. We are processing each inference sequentially (batch 1) since requests need to be processed as soon as they arrive. The GPUs provide large throughput by running many parallel inferences but may not perform well when the batch is small, especially there are data dependencies in LSTMs. However, FPGAs work fast on a single inference with a fully unrolled tailor-made design.

TABLE III: Latency comparison of the FPGA design versus CPU and GPU

	CPU	GPU	This work
Platform	Intel E2620	TITAN X	U250
Precision	F32	F32	16 Fixed
Latency	39.7 ms	32.1 ms	0.40 us

TABLE IV: Comparison with previous FPGA-based LSTM designs for anomaly detection and physics

	[28], 2018	[27], 2020	This work	This work
FPGA	Kintex7 K410T	KU115	U250	U250
Model	Single Layer	Single Layer	Single Layers	Four Layers
Application Domain	Anomaly Detection	Physics	-	Anomaly Detection
LSTM hidden units Lh	32	16	32	32,8,8,32
DSPs	1091	2374	2221	9021
Preci. (bits)	16 fixed	16 fixed	16 fixed	16 fixed
Freq. (MHz)	155	200	300	300
Latency (us)	4.27	1.35	0.343	0.867

Some other HLS-based RNN/LSTM accelerators on FPGAs are compared with ours in Table IV. In this table, we focus on latency since the throughput, power or power efficiency of the other designs are not reported. Our design achieves 4.92 to 12.4 times lower latency compared to the state-of-the-art FPGA designs targeting anomaly detection. Our single-layer design, with a similar amount of DSP resources to another design [27], is 3.9 times faster as shown in Table IV. Note that because of the structure of an autoencoder, the processing of the encoder and the decoder cannot be overlapped, which increases the end-to-end latency of the design. Nevertheless, we still achieve better latency than the others which contain only one LSTM layer. Moreover, while the other designs report Vivado HLS synthesis latency, we report the RTL co-simulation latency which is likely to be more accurate.

VI. Related Work

A latency-optimized LSTM-based anomaly detection is proposed in [28] on FPGAs and we achieve 4.9 times faster than it. [14] proposes the HLS4ML tool and introduces a deep FC-layer model for substructure-based jet tagging in LHC physics. [27] introduces HLS LSTMs for the same physics problem.

Partitioning FPGA resources to improve throughput has been studied for CNNs [8, 9, 19, 20], but they do not touch the RNNs and the recurrent nature and data dependency in RNN computations which are absent in CNNs. The FiC-RNN [29] proposes to accelerate multi-layer RNNs using an FPGA cluster, in which each RNN layer occupies a single FPGAs. The authors in [10] put each LSTM layer on each multi-core to achieve coarse grained pipelining. In [30, 31, 32, 33], the batching technique is used to improve the hardware throughput and utilization for LSTM inferences. However, latency can suffer since different inputs may not come at the same time, meaning that a newly arrived request has to wait until the batch is formed, which imposes a significant latency penalty.

Some of the previous studies [1, 34, 35, 36, 37, 38] are focusing on weight pruning and model compression to achieve good performance and efficiency. Some researchers use low bitwidth, even binarized, datapaths [30, 39, 40] and investigate the trade-off between precision and performance. These studies are orthogonal to our proposed approach and hardware architecture. These techniques can be complementary to our approach to achieve even lower latency of RNN inferences on FPGAs.

VII. Conclusions and Future Work

This paper aims to pioneer new data analysis architectures to support next-generation low-latency anomaly detection on time series data, relevant to many fundamental physics experiments including gravitational wave detection. We present a novel approach for minimizing the initiation intervals for the execution of a multi-layer LSTM network by optimizing the reuse factors for each layer. Results show latency reduction of up to 12.4 times over the existing FPGA-based LSTM design. Current and future work includes exploring the use of new FPGA resources such as the AI Engines [41] and the AI Tensor Blocks [42], and incorporating the proposed approach into the design of the data analysis architecture for next-generation gravitational wave detectors.

Acknowledgement

The support of the United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1, and EP/S030069/1), CERN and Xilinx is gratefully acknowledged. We thank Prof. Zhiru Zhang and Yixiao Du for their help and advice.

References

[1] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[2] A. Schmitt *et al.*, "Investigating Deep Neural Networks for Gravitational Wave Detection in Advanced LIGO Data," in *Proceedings of the 2nd International Conference on Computer Science and Software Engineering*, 2019.

[3] Y.-C. Lin and J.-H. P. Wu, "Detection of gravitational waves using Bayesian neural networks," *Physical Review D*, vol. 103, no. 6, p. 063034, 2021.

[4] Y. Guan *et al.*, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017.

[5] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 1–14.

[6] E. Nurvitadhi, D. Kwon, A. Jafari, A. Boutros, J. Sim, P. Tomson, H. Sumbul, G. Chen, P. Knag, R. Kumar *et al.*, "Why compete when you can work together: Fpga-asic integration for persistent rnns," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 199–207.

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

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

[9] H. Nakahara *et al.*, "High-Throughput Convolutional Neural Network on an FPGA by Customized JPEG Compression," in *28th FCCM*. IEEE, 2020.

[10] L. Peng *et al.*, "Exploiting Model-Level Parallelism in Recurrent Neural Network Accelerators," in *International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2019.

[11] A. Hannun *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[12] Xilinx, "SDSoC Profiling and Optimization Guide."

[13] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of High-Level Synthesis Codes for High-Performance Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1014–1029, 2020.

[14] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[15] D. Thorwarth and D. A. Low, "Technical Challenges of Real-Time Adaptive MR-Guided Radiotherapy," *Frontiers in Oncology*, vol. 11, p. 332, 2021.

[16] S. Denholm *et al.*, "Low latency FPGA acceleration of market data feed arbitration," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2014.

[17] E. Moreno. [Online]. Available: https://github.com/eric-moreno/Anomaly-Detection-Autoencoder

[18] E. Moreno *et al.* in preparation.

[19] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 535–547.

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

[21] B. Moons *et al.*, "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017.

[22] E. Azari and S. Vrudhula, "An energy-efficient reconfigurable lstm accelerator for natural language processing," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4450–4459.

[23] "Generate Gravitational-Wave Data (GGWD)," https://github.com/timothygebhard/ggwd, 2019.

[24] A. Nitz *et al.*, "gwastro/pycbc: Pycbc release v1.16.9," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3993665

[25] LIGO Scientific Collaboration, "LIGO Algorithm Library - LALSuite," free software (GPL), 2018.

[26] C. Coelho, A. Kuusela, S. Li, H. Zhuang, T. Aarrestad, V. Loncar, J. Ngadiuba, M. Pierini, A. Pol, and S. Summers, "Automatic deep heterogeneous quantization of deep neural networks for ultra low-area, low-latency inference on the edge at particle colliders," *arXiv preprint arXiv:2006.10159*.

[27] R. Rao, "Implementation of Long Short-Term Memory Neural Networks in High-Level Synthesis Targeting FPGAs," Ph.D. dissertation, 2020.

[28] Y. H. Lee, D. J. Moss, J. Faraone, P. Blackmore, D. Salmond, D. Boland, P. H. Leong *et al.*, "Long Short-Term Memory for Radio Frequency Spectral Prediction and its Real-Time FPGA Implementation," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 1–9.

[29] Y. Sun and H. Amano, "Fic-rnn: A multi-fpga acceleration framework for deep recurrent neural networks," *IEICE Transactions on Information and Systems*, vol. 103, no. 12, pp. 2457–2462, 2020.

[30] V. Rybalkin *et al.*, "FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs," in *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018.

[31] Z. Que *et al.*, "Efficient Weight Reuse for Large LSTMs," in *30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2019.

[32] A. Boutros *et al.*, "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs," in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2020.

[33] Z. Que *et al.*, "Mapping Large LSTMs to FPGAs with Weight Reuse," *Journal of Signal Processing Systems*, 2020.

[34] Z. Chen, A. Howe, H. T. Blair, and J. Cong, "Clink: Compact lstm inference kernel for energy efficient neurofeedback devices," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[35] S. Cao *et al.*, "Efficient and Effective Sparse LSTM on FPGA with Bank-Balanced Sparsity," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019.

[36] R. Shi *et al.*, "E-LSTM: Efficient inference of sparse LSTM on embedded heterogeneous system," in *56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019.

[37] G. Nan *et al.*, "DC-LSTM: Deep Compressed LSTM with Low Bit-Width and Structured Matrices," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020.

[38] Z. Chen *et al.*, "BLINK: bit-sparse LSTM inference kernel enabling efficient calcium trace extraction for neurofeedback devices," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020.

[39] E. Nurvitadhi *et al.*, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016.

[40] V. Rybalkin and N. Wehn, "When Massive GPU Parallelism Ain't Enough: A Novel Hardware Architecture of 2D-LSTM Neural Network," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 111–121.

[41] "Xilinx AI Engines and Their Applications," in *WP506(v1.1)*, July 10, 2020.

[42] M. Langhammer *et al.*, "Stratix 10 NX Architecture and Applications," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021.