EdgeDRNN: Recurrent Neural Network Accelerator for Edge Inference

Chang Gao (ID), *Student Member, IEEE*, Antonio Rios-Navarro (ID), *Member, IEEE*, Xi Chen (ID), *Student Member, IEEE*, Shih-Chii Liu (ID), *Senior Member, IEEE*, and Tobi Delbruck (ID), *Fellow, IEEE*

Abstract—**Low-latency, low-power portable recurrent neural network (RNN) accelerators offer powerful inference capabilities for real-time applications such as IoT, robotics, and human-machine interaction. We propose a lightweight Gated Recurrent Unit (GRU)-based RNN accelerator called EdgeDRNN that is optimized for low-latency edge RNN inference with batch size of 1. EdgeDRNN adopts the spiking neural network inspired delta network algorithm to exploit temporal sparsity in RNNs. Weights are stored in inexpensive DRAM which enables EdgeDRNN to compute large multi-layer RNNs on the most inexpensive FPGA. The sparse updates reduce DRAM weight memory access by a factor of up to 10x and the delta can be varied dynamically to trade-off between latency and accuracy. EdgeDRNN updates a 5 million parameter 2-layer GRU-RNN in about 0.5 ms. It achieves latency comparable with a 92 W Nvidia 1080 GPU. It outperforms NVIDIA Jetson Nano, Jetson TX2 and Intel Neural Compute Stick 2 in latency by 5X. For a batch size of 1, EdgeDRNN achieves a mean effective throughput of 20.2 GOp/s and a wall plug power efficiency that is over 4X higher than the commercial edge AI platforms.**

Index Terms—**Edge computing, FPGA, embedded system, deep learning, RNN, GRU, delta network.**

I. INTRODUCTION

DEEP neural networks (DNNs) have been widely applied to solve various practical problems with state-of-the-art performance. Recurrent neural networks (**RNN**s) which are a subset architecture of DNNs, are particularly useful in applications involving time series inputs, such as speech recognition [1], [2] and dynamical system control [3], [4]. In contrast to Convolutional Neural Networks (**CNN**s) which use filter kernels, RNNs are fully-connected networks: They take a 1D vector as input and produce a vector of output. The feature vectors generated by CNNs can be fed into an RNN for further processing. In this way, RNNs can connect

Chang Gao, Xi Chen, Shih-Chii Liu, and Tobi Delbruck are with the Institute of Neuroinformatics, University of Zürich and ETH Zürich, 8057 Zürich, Switzerland (e-mail: chang@ini.uzh.ch; xi@ini.uzh.ch; shih@ini.uzh.ch; tobi@ini.uzh.ch).

Antonio Rios-Navarro is with the Robotics and Technology of Computers Laboratory, Universidad de Sevilla, 41012 Seville, Spain (e-mail: arios@us.es).
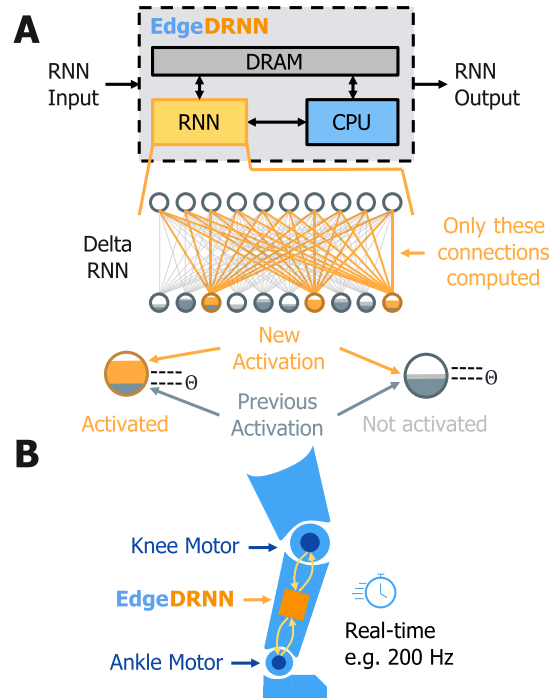
Fig. 1. A: EdgeDRNN accelerator concept. B: Example target application from [7].

the high dimensional input features over time, which is useful for complex sequential classification or regression tasks. Gated RNNs modify a "vanilla" RNN to add nonlinear operations to the units that allow them to memorize and gate their output. Long Short-Term Memory units (**LSTM**) [5] and Gated-Recurrent Units (**GRU**) [6] are used to overcome the vanishing gradient problem frequently encountered during vanilla RNN training with backpropagation through time (**BPTT**), where the sequential operations of the RNN are unrolled to compute the weight updates based on output error. By using BPTT with labeled training data, GRU and LSTM RNNs can be trained to high accuracy for tasks involving time series such as continuous speech recognition.

Edge computing devices that embed some intelligence implemented through trained DNNs are gaining interest in recent years. An advantage of edge computing is that computations are done locally on end-user devices to reduce latency and protect privacy [8]. Most literature reports on the use of CNNs for edge devices. There is less reported on devices that use RNNs particularly on embedded low-latency, high energy-efficient platforms that use FPGAs. RNNs also have larger

memory footprints and memory access of the fully-connected weight matrices dominates power consumption. RNNs are usually computed on the cloud which introduces large and variable latency, thereby making it hard to guarantee real-time performance for edge applications such as human-computer interaction devices and mobile, robotic applications.

Optimization methods have been applied to RNNs for embedded hardware implementations (e.g. weight pruning in ESE [9], and structure pruning in BBS [10]). We also previously reported the DeltaRNN accelerator [11] that uses the delta network algorithm [12]. Our first DeltaRNN implementation [11] stored the large weight matrices in FPGA block RAM and thus needed expensive FPGA boards with greater than 15 W power consumption. However, this work only focused on pushing the limit of high batch-1 throughput without considering the memory and power constraints of extreme edge devices. Typical chips for edge applications, such as microcontrollers and small-footprint FPGA, only have a few hundreds of kilobytes (kB) of on-chip SRAM, but large RNNs usually have megabytes (MB) of parameters, making them difficult to be buffered on-chip even after compression. In this case, storing RNN parameters in off-chip memory such as flash or DDR memory is an inevitable choice for edge devices. Therefore, reported hardware RNN implementations [9], [10] cannot be easily scaled down for edge platforms.

This work is on EdgeDRNN, a hardware accelerator for RNN computation on the edge [13]. Because of our interest in real-time edge applications, our focus is on supporting low-latency batch-1 inference of large RNNs for real-time performance on low-cost and low power but very constrained edge platforms. We show that our implementation can run large-scale multi-layer RNNs using a small number of processing elements with the large weight matrices stored on off-chip DRAM. Besides accelerating RNN inference, it leaves most cycles of the CPU in the system-on-chip (**SoC**) available for other tasks, such as feature extraction and I/O management. EdgeDRNN can be implemented on a small-footprint FPGA with 19X less logic cells and 15X less on-chip memory compared to the one used in DeltaRNN [11]. Thus, EdgeDRNN is suitable for embedded system applications such as robotics (Fig. 1B).

Moreover, in previous work using the current delta network algorithm, a global threshold is applied on both the inputs and hidden unit activations of every layer of the network in sparsifying the activation vector. In this work, we looked at how different threshold values on the inputs and activations of hidden units affect the trade-off between the accuracy of the network on a regression task and the sparsity levels of the change in the activation vectors. A higher sparsity level implies reduced weight memory access and reduced computes.

This article makes the following contributions:

1) We describe a flexible, low-cost, high throughput edge FPGA RNN accelerator that uses a spiking neural network-inspired Delta RNN principle to provide state-of-art latency and power efficiency for a wide range of gated RNN network sizes with batch size of 1.

2) We report the first study of a delta network that uses different delta thresholds for the input and activations of the hidden units. On our tested regression task, this modification increases temporal sparsity of hidden delta states by 16% compared to using a global threshold.

3) We compare the usability and throughput performance of two different EdgeDRNN implementation on the SoC FPGA: Bare-metal and embedded Linux. The latter enables faster development and we report the correct FPGA memory bus port configuration that minimizes the performance loss due to CPU contention for the memory controller.

4) We report benchmark latency and throughput numbers of RNN inference on state-of-the-art commercial chips for edge applications. To our best knowledge, these numbers have never been reported before.

The rest of this article is organized as follows. Section II describes the background of gated recurrent unit (GRU) based RNN and the algorithm of delta network based GRU-RNN, which is called *DeltaGRU*. Section III describes the architectural design of the accelerator and its implementation on MiniZed. Section IV discusses experimental results including the experiments using different delta thresholds for the network. Section V compares the proposed accelerator with prior work and commercial edge devices. Section VI concludes the paper.

II. BACKGROUND

A. DNN Hardware Optimization Methods

Various methods have been proposed to reduce the dominant RNN matrix-vector (**MxV**) operations. Static approximation methods (i.e. constructed during training) include *quantization*, *arithmetic*, and *weight pruning*.

1) Quantization: Quantizing floating-point weights or activations to fixed-point numbers with shorter bit width reduces memory footprint of networks and make it possible to use fixed-point MAC units instead of expensive floating-point MAC units [9], [14]–[16]. Chip area can be further reduced by replacing conventional fixed-point multipliers by look-up table based [17] or multiplexer [18] based multipliers on low bit precision networks with 2-4 bit weights. By including quantization during training (e.g. by using an approach like dual-copy rounding [19]) it is possible to reduce weight precision to 8 bits without accuracy loss.

2) Weight Pruning: Pruning removes unimportant neuron connections that results in sparse weight matrices [20]. Sparse matrices can be encoded into a sparse matrix format such as the Compressed Sparse Column (**CSC**) and Compressed Sparse Row (**CSR**). With an accelerator that can decode the sparse matrix format on-chip, the sparse matrix-vector (**SpMV**) multiplication can be accelerated by executing multiply-and-accumulate (**MAC**) operations only on nonzero weights. This approach was adopted by the Efficient Speech Recognition Engine (**ESE**) [9]. Because unstructured pruning results in computation that is hard to balance across processing elements, structured pruning methods have also been proposed to improve load balancing during the SpMV

computation [10], [21]. This approach was used by the LSTM FPGA accelerator using Bank Balanced Sparsity (**BBS**) [10]. It is also used by the custom digital IC of [21] where it is called Hierarchical Coarse Grain Sparsity (HCGS). Structured pruning is a popular approach for improving RNN hardware performance; both BBS and HCGS use it to increase effective MAC efficiency, but large increases in efficiency result in significantly worse inference accuracy [21]. For example, a 16X compression increases the error rate by a factor of about 1.2X. It allows static (compile-time) optimization, but training is fairly complicated since exploration of the additional structure hyperparameter values is needed to find optimum values that are matched to the particular hardware.

3) Arithmetic: Bit-serial NN accelerators such as [22], [23] utilize a flexible bit-serial MAC to support various precision of network parameters and are smaller in area compared to conventional fixed-point MAC units. However, since a bit-serial MAC units requires more cycles to finish a multiplication between high bit precision operands, more bit-serial MAC units are required to achieve higher throughput than using conventional MAC units and larger adder trees are need for accumulating partial sums. Thus, the average speedup using this method is only around 2X but it comes with extra overhead area. The C-LSTM accelerator used Toeplitz-like weight matrices in the form of blocked circulant matrices to reduce RNN memory requirements [24] since multiple rows in each circulant matrix block can be generated from a single vector. The method also enables the use of Fast Fourier Transform (**FFT**) to reduce the MxV cost from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log(n))$ [25]. However, forcing weight matrices to be blocked circulant is coarse-grained and leads to higher accuracy degradation compared to weight pruning [9], [10]. Moreover, the method leads to hardware overhead of computing the FFTs of activations and weights.

4) Temporal Sparsity: The delta network algorithm [12] capitalizes on the temporal sparsity of activation state vectors in a network. Setting a finite threshold that is greater than zero has the effect of zeroing-out below-threshold elements of the activation vector, which results in sparse delta vectors. Since zero activations have no downstream influence, these MACs can be skipped. It means that entire columns of the weight matrix can be skipped. Thus delta networks marry the temporal sparsity of spiking networks with the synchronous update and analog state transmission of conventional deep networks. Combining these principles provides the benefits of sparse computing and efficient communication of precise analog information with reduced and predictable memory access of inexpensive DRAM which is crucial for storing the weights.

A set of studies [7], [12], [13] showed in a variety of networks that by applying the delta principle during training, the accuracy loss is minimal even with a 5-10X improvement of RNN throughput and latency. For example, [12] used a 4-layer 320 units per layer GRU RNN for continuous speech recognition on the Wall Street Journal dataset. The Word Error Rate (**WER**) increased by only a factor of 1.08X but with a reduced memory access of 6.2X.

B. Gated Recurrent Unit

The update equations for a GRU layer of H neurons and I-dimensional input, which are as follows:

$$\mathbf{r}_t = \sigma \left(\mathbf{W}_{xr}\mathbf{x}_t + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_r \right)$$
$$\mathbf{u}_t = \sigma \left(\mathbf{W}_{xu}\mathbf{x}_t + \mathbf{W}_{hu}\mathbf{h}_{t-1} + \mathbf{b}_u \right)$$
$$\mathbf{c}_t = \tanh \left(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{r}_t \odot \left(\mathbf{W}_{hc}\mathbf{h}_{t-1} \right) + \mathbf{b}_c \right)$$
$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{c}_t + \mathbf{u}_t \odot \mathbf{h}_{t-1} \tag{1}$$

where $\mathbf{r}, \mathbf{u}, \mathbf{c} \in \mathcal{R}^H$ are the *reset gate*, the *update gate* and the *cell state* respectively. $\mathbf{W}_x \in \mathcal{R}^{H \times I}$, $\mathbf{W}_h \in \mathcal{R}^{H \times H}$ are weight matrices and $\mathbf{b} \in \mathcal{R}^H$ are bias vectors. The σ variable denotes the logistic sigmoid function. Each GRU input is the vector $\mathbf{x}_t$ and its output is the vector $\mathbf{h}_t$.

Fig. 2A illustrates the update of the normal GRU reset gate as a flow diagram.

C. DeltaGRU

The delta network method is applied to the GRU-RNN architecture; we call this *DeltaGRU*. Assume an input vector sequence $\mathbf{X} = \{\mathbf{x}_t \mid t = 1, 2, \ldots, T\}$ with sequence length of T, we first declare the following variables:

$$\hat{x}_{i,t} = \begin{cases} x_{i,t}, & |x_{i,t} - \hat{x}_{i,t-1}| \geq \Theta_x \\ \hat{x}_{i,t-1}, & |x_{i,t} - \hat{x}_{i,t-1}| < \Theta_x \end{cases}$$

$$\hat{h}_{j,t} = \begin{cases} h_{j,t}, & |h_{j,t} - \hat{h}_{j,t-1}| \geq \Theta_h \\ \hat{h}_{j,t-1}, & |h_{j,t} - \hat{h}_{j,t-1}| < \Theta_h \end{cases}$$

$$\Delta x_{i,t} = \begin{cases} x_{i,t} - \hat{x}_{i,t-1}, & |x_{i,t} - \hat{x}_{i,t-1}| \geq \Theta_x \\ 0, & |x_{i,t} - \hat{x}_{i,t-1}| < \Theta_x \end{cases}$$

$$\Delta h_{j,t} = \begin{cases} h_{j,t} - \hat{h}_{j,t-1}, & |h_{j,t} - \hat{h}_{j,t-1}| \geq \Theta_h \\ 0, & |h_{j,t} - \hat{h}_{j,t-1}| < \Theta_h \end{cases} \tag{2}$$

where $\hat{x}_{i,t}$ is the i-th element of input state memory vectors $\hat{\mathbf{x}}_t$ in timestep t. $\hat{h}_{j,t}$ is the j-th element of hidden state memory vectors $\hat{\mathbf{h}}_t$ in timestep t. $\Delta x_{i,t}$ is the i-th element of delta input state vectors $\Delta \mathbf{x}_t$. $\Delta h_{j,t}$ is the j-th element of delta hidden state vectors $\Delta \mathbf{h}_t$. Θ_x and Θ_h are respectively the delta thresholds of inputs and hidden state for each layer. In the initial timestep ($t = 1$), $\hat{x}_{i,0}$, $h_{i,0}$, $\hat{h}_{i,-1}$ are all initialized to zeros.

The update equations for the DeltaGRU are now:

$$\mathbf{M}_{r,t} = \mathbf{W}_{xr}\Delta\mathbf{x}_t + \mathbf{W}_{hr}\Delta\mathbf{h}_{t-1} + \mathbf{M}_{r,t-1}$$
$$\mathbf{M}_{u,t} = \mathbf{W}_{xu}\Delta\mathbf{x}_t + \mathbf{W}_{hu}\Delta\mathbf{h}_{t-1} + \mathbf{M}_{u,t-1}$$
$$\mathbf{M}_{xc,t} = \mathbf{W}_{xc}\Delta\mathbf{x}_t + \mathbf{M}_{xc,t-1}$$
$$\mathbf{M}_{hc,t} = \mathbf{W}_{hc}\Delta\mathbf{h}_{t-1} + \mathbf{M}_{hc,t-1}$$
$$\mathbf{r}_t = \sigma(\mathbf{M}_{r,t})$$
$$\mathbf{u}_t = \sigma(\mathbf{M}_{u,t})$$
$$\mathbf{c}_t = \tanh(\mathbf{M}_{xc,t} + \mathbf{r}_t \odot \mathbf{M}_{hc,t})$$
$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{c}_t + \mathbf{u}_t \odot \mathbf{h}_{t-1} \tag{3}$$

where $\mathbf{M}_{r,t=0} = \mathbf{b}_r$, $\mathbf{M}_{u,t=0} = \mathbf{b}_u$, $\mathbf{M}_{xc,t=0} = \mathbf{b}_c$, $\mathbf{M}_{hc,t=0} = 0$ are delta memory vectors and $\mathbf{M} \in \mathcal{R}^H$. Variables σ and $\odot$ indicate the sigmoid function and element-wise multiplication of vectors respectively.
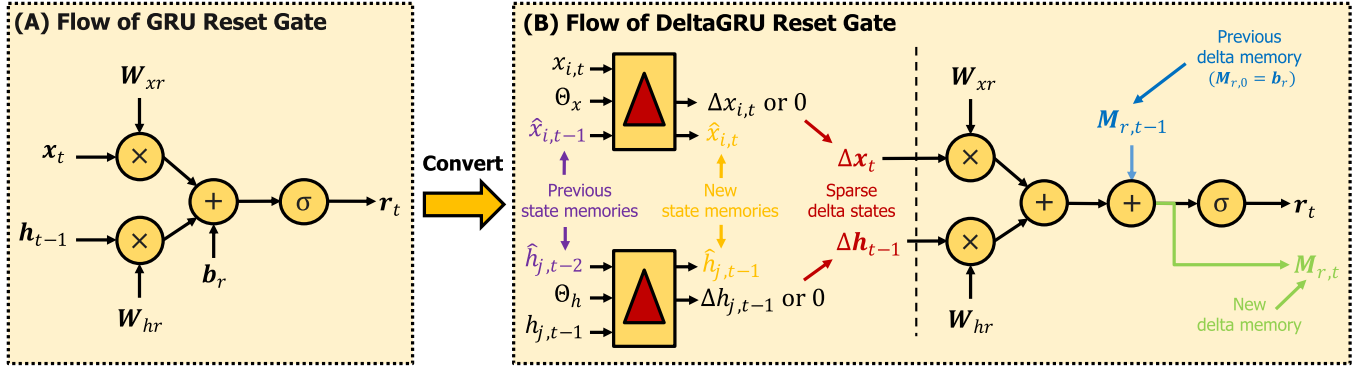
Fig. 2. Computation flow of the reset gate in gated recurrent units (GRUs). A: Normal GRU reset gate update. B: A DeltaGRU reset gate update. See Eqs.1-3.

Fig. 2B illustrates these operations for the DeltaGRU reset gate. The input vector $\mathbf{x}_t$ and the hidden state vector $\mathbf{h}_{t-1}$ are respectively replaced by the delta input state vector $\Delta\mathbf{x}_t$ and the delta hidden state vector $\Delta\mathbf{h}_{t-1}$. Values of the previous state memory vectors $\hat{\mathbf{x}}_{t-1}$, $\hat{\mathbf{h}}_{t-2}$ are updated using Eq. 2 to generate new state memory vectors $\hat{\mathbf{x}}_t$, $\hat{\mathbf{h}}_{t-1}$. The previous delta memory vector $\mathbf{M}_{r,t-1}$ holds the previous step's partial sum-product and the resulting new delta memory vector $\mathbf{M}_{r,t}$ is stored. Otherwise the operations are the same as for the original GRU reset gate, as shown in Fig. 2A. The other gates have similar flow diagrams. The state and delta memories are 1D vectors and can be easily fit into on-chip SRAM buffers.

D. Temporal Sparsity

The temporal sparsity Γ of a DeltaGRU network of L layers with an input sequence length of T is defined as the fraction of zeros in the Δ_x and Δ_h vectors, signified by $\Gamma_{\Delta x}$ and $\Gamma_{\Delta h}$ respectively. The effective temporal sparsity Γ_{Eff} is the weighted average of $\Gamma_{\Delta x}$ and $\Gamma_{\Delta h}$ according to the number of network parameters they correspond to. The definition of temporal sparsity is given by Eq. 4:

$$\Gamma_{\Delta x} = \frac{1}{L \cdot T \cdot I} \sum_{t=1}^{T} n_{x,t}^{1} + \frac{1}{(L-1) \cdot T \cdot H} \sum_{l=2}^{L} \sum_{t=1}^{T} n_{x,t}^{l}$$

$$\Gamma_{\Delta h} = \frac{1}{L \cdot T \cdot H} \sum_{l=1}^{L} \sum_{t=1}^{T} n_{h,t}^{l}$$

$$\Gamma_{\text{Eff}} = \frac{\left(3HI + 3H^2(L-1)\right) \cdot \Gamma_{\Delta x} + 3H^2 L \cdot \Gamma_{\Delta h}}{3HI + 3H^2(L-1) + 3H^2 L}$$

$$= \frac{(I + H(L-1)) \cdot \Gamma_{\Delta x} + HL \cdot \Gamma_{\Delta h}}{I + H(L-1) + HL} \quad (4)$$

where $n_{x,t}^{l}$ and $n_{h,t}^{l}$ are the number of zero elements respectively in the delta vectors Δx and Δh in layer l at timestep t. Because operations on biases are negligible, they are ignored in Eq. 4.

By skipping zero elements in delta vectors, whole columns of matrix-vector MAC operations can be skipped. If the delta network is properly trained (by including the delta operation), [7], [12], [26] showed that the number of operations can be reduced by 5X to 100X with negligible loss of accuracy, depending on the temporal evolution of the states of the input and hidden units.
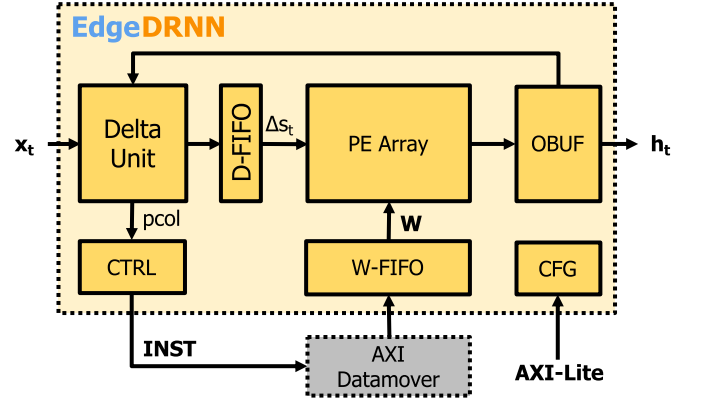

Fig. 3. EdgeDRNN accelerator architecture.

E. Datasets

Two datasets are used in this article: the *TIDIGITS* [27] dataset for the classification task and the *SensorsGas* dataset for the regression task. The *TIDIGITS* speech dataset has more than 25k digit sequences spoken by over 300 men, women, and children. The entire training and test sets are used in our experiments. The *SensorsGas* dataset consists of recordings of metal-oxide sensors in response to various concentrations of carbon monoxide gas over 14 days [28], [29]. This dataset was used in [30] to evaluate the network performance of a gated RNN in predicting the concentration of carbon monoxide. The dataset used here comes from the 70/30 Split variant, that is, 70% of the sequences are randomly selected to form the training set, while the remaining sequences form the test set.

III. EDGEDRNN ACCELERATOR

A. Overview

Due to limited weight reuse, it is difficult to compute RNNs efficiently for real-time applications that usually work best with a batch size of 1. Therefore, a big challenge of RNN inference on the edge is the scarce off-chip memory bandwidth available on portable platforms, and the limited amount of on-chip block RAM on small FPGAs. EdgeDRNN uses cheap off-chip DRAM for weight storage and reduces memory bandwidth by exploiting temporal sparsity in RNN updates.

Fig. 3 shows the architecture of the EdgeDRNN accelerator. The main modules consist of the Delta Unit for encoding delta
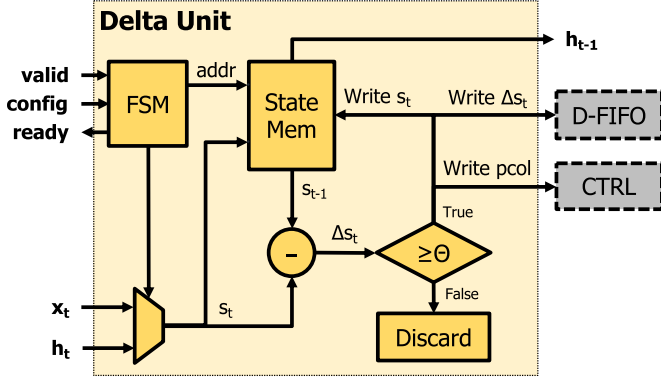
Fig. 4.   Flow chart of the Delta Unit.

Fig. 5.   Architecture of the EdgeDRNN processing element (PE).

vectors and generating weight column pointers (**pcol**); the Processing Element (**PE**) Array for matrix-sparse vector multiplications; the (**CTRL**) control module which contains finite state machines (**FSMs**) and encodes instructions to control the AXI Datamover. Other modules include the configuration module (**CFG**) composed of configuration registers; the output buffer (**OBUF**) for buffering and redirecting outputs back to the Delta Unit and the **W-FIFO** for buffering weights.

B. Delta Unit & CTRL

The Delta Unit stores state memory for delta state vector encoding in a block random access memory (**BRAM**[1]). The FSM addresses the BRAM according to the `valid` signal of input state vectors x_t and h_t, one of which, is selected to be processed as s_t at a time depending on the FSM state. The Delta Unit encodes one element of a delta state vector Δs_t in each clock cycle after the `valid` signal asserted until the whole vector is processed.

The vector sizes are provided by the `config` signal from the CFG module. Delta state vector elements that are greater than or equal to threshold Θ_x or Θ_h; and their corresponding physical weight column address pointer (`pcol`) are respectively dispatched to the D-FIFO and CTRL. The corresponding state element s_t is written into the BRAM to update the state memory. Otherwise, elements are discarded without being written into the D-FIFO. By using only 1 Delta unit, the latency in clock cycles for the Delta Unit to process a vector is exactly the length of that vector. It is possible to reduce the latency by searching for nonzero elements in N subsections of a vector simultaneously. It can be realized by using N Delta Unit blocks in parallel to fill at most one nonzero value into the D-FIFO on every clock cycle. Assuming that nonzero elements are uniformly distributed in a delta state vector and using N Delta Unit blocks running in parallel, the latency τ_{DU} in clock cycles to process a whole vector is

$$\tau_{DU} \approx \max\left(\left\lceil \frac{D}{N \cdot d} \right\rceil, \lceil D \cdot (1 - \Gamma) \rceil \right) \quad (5)$$

where D is the length of the vector, d is the length of the subsection of the vector or the look-ahead window side of the Delta Unit; and Γ is the temporal sparsity defined in Eq. 4.

[1]BRAM is the standard SRAM memory block on FPGAs; on Xilinx Zynq FPGAs a single BRAM block has 18-bit words and a capacity of 18kb.
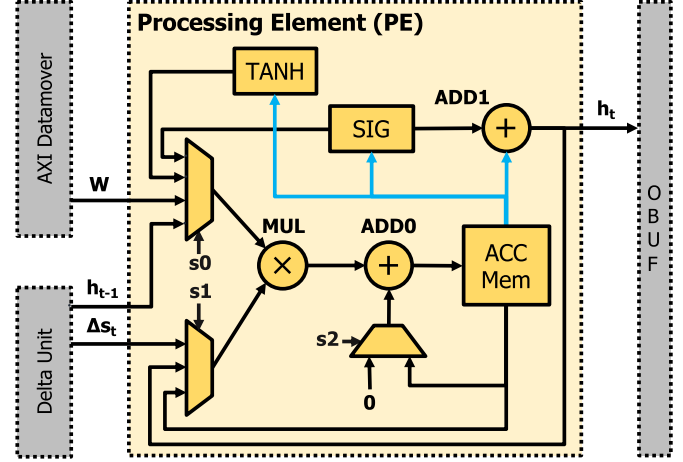
Although τ_{DU} can be hidden under τ_m, the latency of computing MxV, τ_{DU} becomes a bottleneck of total latency when $\tau_{DU} > \tau_m$, which could happen when an accelerator uses a large number of MAC units to compute small networks. However, in this work, we aim to run large network inference with a small number of MAC units for edge applications, making $\tau_{DU} \ll \tau_m$; thus, $\eta = 1$ is used in EdgeDRNN. The MAC utilization results shown in Section IV.D prove that this choice did not lead to latency bottleneck.

The CTRL module contains FSMs that control the PE array. This module generates 80-bit instructions for controlling the Xilinx AXI Datamover IP [31] to fetch RNN parameters. The instruction contains `pcol` and the burst length calculated from the dimensions of the network stored in configuration registers.

C. Processing Element Array

Two-dimensional arithmetic unit arrays such as systolic arrays are difficult to be fully utilized in portable edge devices due to scarce on-chip memory resources, the low external memory bandwidth of the system and the limited weight reuse nature of RNNs. In order to fully utilize every PE, a vector PE array is used in EdgeDRNN. Fig. 5 show the internal structure of a PE.

The PE has a 16-bit multiplier **MUL** and two adders, 32-bit **ADD0** and 16-bit **ADD1**. Multiplexers are placed before operands of MUL so that the PE can be reused for both MxV and vector dot products. The multiplexer below ADD0 selects between '0' and the BRAM data. '0' is chosen when an initialization of BRAM is needed as shown in Fig. 5. ADD1 is responsible for element-wise vector additions. All units are parameterized in the System Verilog RTL and configurable at compile-time to support any fixed-point precision within their designed bit width. The PE supports `tanh` and `sigmoid` functions by using look-up tables (**LUTs**). The input bit width of LUTs is fixed to 16 bits while the output bit width can be set anywhere between 5 (Q1.4) to 9 (Q1.8) bits.

Fig. 6 shows the sparse MxV flow. The weight matrices of the GRU-RNN are concatenated following the arrangement shown on the right half of the figure. Biases are appended to the concatenated weight matrix as the first column and an element 1 is appended to each input state vector $\mathbf{x}_t$ as
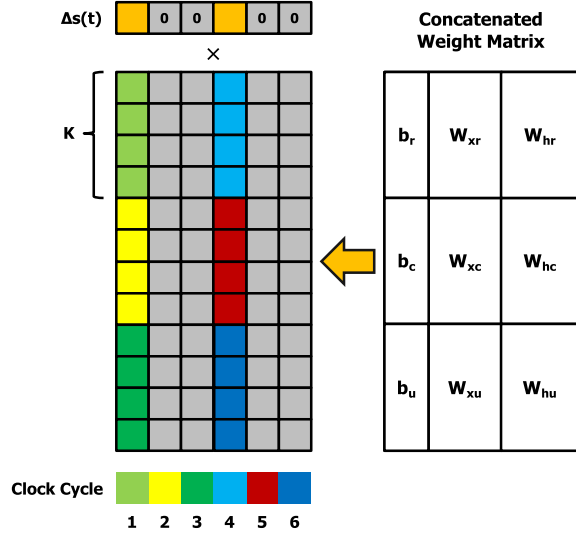
Fig. 6. Flow of sparse matrix-vector multiplication in EdgeDRNN and the arrangement of GRU weights in the concatenated weight matrix.

	LUT	LUTRAM	FF	BRAM	DSP
Available	14400	6000	28800	50	66
EdgeDRNN	30.8%	0.4%	9.3%	32%	13.6%
Total	65.2%	4.4%	34.1%	66%	13.6%

the first element. The PE array multiplies only nonzero delta state elements with corresponding valid columns. Products are accumulated in the Accumulation Memory (**ACC Mem**) to compute delta memory vectors $\mathbf{M}_r$, $\mathbf{M}_u$, $\mathbf{M}_{ic}$, $\mathbf{M}_{hc}$. Products involving $\mathbf{b}_r$, $\mathbf{W}_{ir}$, $\mathbf{W}_{hr}$ are accumulated to $\mathbf{M}_r$; $\mathbf{b}_u$, $\mathbf{W}_{iu}$, $\mathbf{W}_{hu}$ to $\mathbf{M}_u$; $\mathbf{b}_c$, $\mathbf{W}_{ic}$ to $\mathbf{M}_{ic}$; $\mathbf{W}_{hc}$ to $\mathbf{M}_{hc}$. According to the delta update scheme defined by Eq. 2, the appended 1 in the delta state vector $\Delta\mathbf{x}_t$ becomes 0 after the initial timestep, which means that biases $\mathbf{b}_r$, $\mathbf{b}_u$, $\mathbf{b}_c$ are only accumulated to the ACC Mem by once and will be skipped by the Delta Unit after the initial timestep.

The calculation of activation h_t after the MxV is also done by the PE array and stages of this process are shown in Fig. 7. The PE array fetches the delta memory vectors from the ACC Mem to calculate h_t in 8 pipeline stages. Paths without any operator in any stage are buffered for 1 clock cycle using flip-flops. During execution of the activation h_t generation, stages **S0**~**S2** are executed simultaneously with **S5**~**S7** to reuse the arithmetic units using time-division multiplexing.

Finally, assuming that the DRAM interface can deliver W_{DRAM} bits per RNN clock cycle for weight fetch, the optimum number K of PEs in the array is determined by the weight precision bit width W_{Weight}. The definition of K and corresponding theoretical peak throughput, v_{Peak}, is defined below:

$$K = W_{\mathrm{DRAM}}/W_{\mathrm{Weight}},$$
$$v_{Peak} = 2 \cdot f_{pl} \cdot K \qquad (6)$$

where f_{pl} is the clock frequency of the programmable logic. For example, the FPGA used in this article has a 64-bit DRAM interface, so, with 16-bit weights, $K = 8$ is optimal.

D. Implementation on MiniZed

Fig. 8 shows the latest implementation of EdgeDRNN on the $89 MiniZed development board [32] which has a Zynq-7007S SoC. EdgeDRNN is implemented in the programmable logic (**PL**). The SoC also has a programmable CPU which is in a module called the Processing System (**PS**). Data is

transferred between PS and PL through high performance (**HP**) slave ports while control signals generated by the PS is transferred through general purpose (**GP**) master ports. The hard IP block, AXI Datamover, is controlled by the PS to fetch weights to the PL from DDR3L memory. Another hard IP block, AXI DMA is controlled by the PS to transfer inputs and outputs of the accelerator. Compared to our previous work [13], we reduced resource utilization by replacing the AXI SmartConnect IP with the AXI Interconnect IP while preserving the same throughput and latency. To further reduce on-chip power, we used the power optimization strategy during implementation in Xilinx Vivado and lower the ARM CPU clock frequency from 667 MHz to 400 MHz for the bare-metal version.

The peak DRAM read memory bandwidth is 1 GB/s at the 125 MHz clock frequency (64-bits×125 MHz/8-bits/byte). EdgeDRNN can be configured to support 1, 2, 4, 8, 16-bit fixed-point weights and 16-bit fixed-point activations. In this article, EdgeDRNN is configured to support 16-bit activation and 8-bit weights. To fully exploit this HP port bandwidth, we implement $K = 8$ PEs following Eq. 6. Adding more PEs would only leave them idle since weight fetches are limited by the DRAM bandwidth.

The AXI-Lite General Purpose (**GP**) master port is used for the single-core ARM Cortex-A9 CPU to control the AXI-DMA and to write the configuration to the accelerator. Configurations include physical start address of the concatenated weights, delta thresholds, and network dimensions.

The PL is driven by single clock domain of 125 MHz generated by the PS. Table I shows the resource utilization solely for EdgeDRNN (with 5-bit (Q1.4) LUTs) and for the whole PL after synthesis and implementation. BRAMs are used to store previous state memory in the Delta Units and the accumulation memory in PEs and FIFOs. 8 DSPs are used for the MAC units in the 8 PEs while the remaining DSP in CTRL produces weight column addresses. The most consumed resources are LUTs (72%). This entry-level XC7Z007S FPGA has only 14.4k LUTs. By comparison, the top level XC7Z100 has 19X more LUTs and 11X more BRAM.

E. Petalinux OS Integration

Xilinx's Zynq chips are hosted on heterogeneous embedded platforms with a variety of peripherals and communication interfaces. To work with this type of system there are two workflows, bare-metal and embedded OS.

The bare-metal workflow is similar to the workflow of conventional microcontrollers. Bare-metal has a set of libraries that establish a very thin software layer over all the hardware resources available in the system and that helps a little during the elaboration of the software that will be deployed in
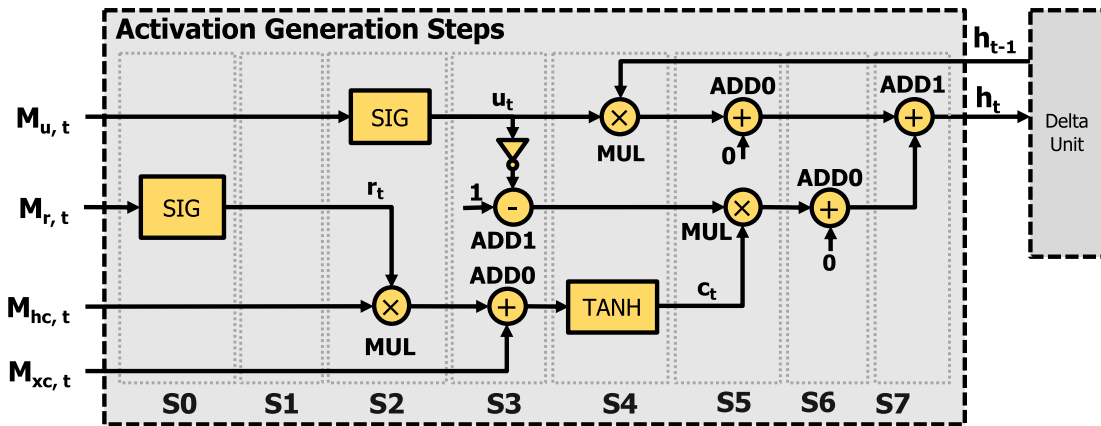
Fig. 7. Stages of GRU activation pipeline in the PE Array.

the system; however, detailed knowledge of the hardware is still necessary to ensure correct functionality. The resulting software runs on the PS processor making use of all its computing power since it is the only software running on the core. Bare-metal allows a more dedicated use of the system resources to achieve high performance execution but it offers little flexibility and versatility.

The second option is to use an embedded Linux OS provided by Xilinx called PetaLinux. This OS establishes several software layers over the system hardware that simplifies its use and the development of applications that make use of the system's peripherals like USB, Bluetooth, and Wi-Fi. The Linux system is a preemptive multitasking operating system that can make application development much faster. Since running Linux slightly slows down inference (Sec. IV), users can decide to pay the throughput price of using Linux for faster development time and easier maintenance. For EdgeDRNN, we implemented both systems to meet our various application requirements.

IV. EXPERIMENTAL RESULTS

We previously developed two EdgeDRNN system-level demonstrations: continuous spoken digit recognition [26] and real-time control of a powered leg prosthetic [7]. Here we report the results of new experiments to measure accuracy, throughput, and power efficiency on the spoken digit task and a new regression task on gas concentration estimation. We also report measurements of embedded Linux implementation of EdgeDRNN.

A. Experimental Setup: Training

We evaluate the accuracy of DeltaGRU and the hardware performance of EdgeDRNN using this DeltaGRU network on both a classification task using the *TIDIGITS* [27] dataset and on a regression task using the *SensorsGas* [30] dataset.

1) Classification: For the classification task, we trained 6 different DeltaGRU network sizes and compared their WER on the *TIDIGITS* audio digit dataset, evaluated using the greedy decoder. Inputs to the networks consist of 40-dimensional log filter bank features extracted from audio sampled at 20 kHz using a frame size of 25 ms and frame stride of 10 ms. We use the Connectionist Temporal Classification (**CTC**) loss [33]
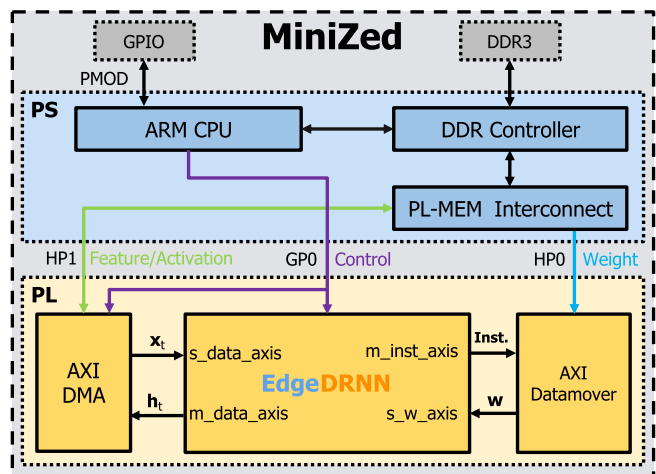

Fig. 8. Top-level diagram of the EdgeDRNN implementation on the MiniZed development board.

to handle variable input sequence lengths. The DeltaGRU networks were trained for 50 epochs using a learning rate of 3e-4 and batch size of 32. Following a similar procedure in [26], a continuous spoken digit recognition demo is built using EdgeDRNN to prove the system functionality.[2]

2) Regression: For the *SensorsGas* regression task, the input dimension of the network is 14 corresponding to data from the 14 sensors. We adopt a 2-step pretrain and retrain scheme we developed for [7]: 1) We pretrain a cuDNN GRU model on the training set for 100 epochs. The learning rate is 5e-4 and the batch size of 64. 2) We load these parameters into a DeltaGRU network with same size as the cuDNN GRU and retrain for another 10 epochs with learning rate of 3e-3 and batch size of 256. In this step we optimize the deltas for the visible and hidden units. Because the cuDNN GRU model is highly optimized for NVIDIA GPUs, the pretrain step helps to train the network to achieve high accuracy with 5X less time.

All networks are trained using the Adam optimizer and quantization-aware training using quantization scheme similar to [19]. To improve accuracy, we use nonlinear functions with the same input and output bit precision as the LUT in the forward phase of the training. In the backward phase, the gradient of the nonlinear function is calculated using

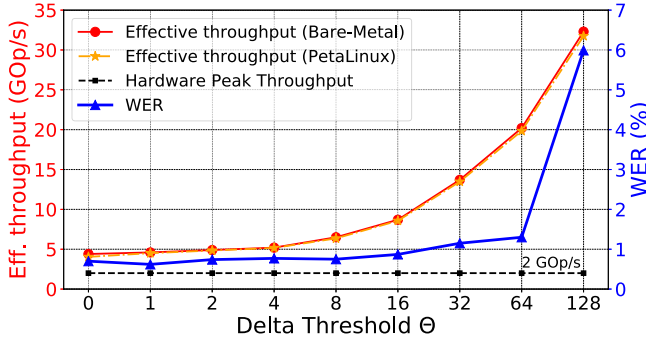[2]https://www.youtube.com/watch?v=XyN-jh5yiMI

Fig. 9. Mean effective throughput and word error rate evaluated on the *TIDIGITS* test set as a function of the delta threshold used in both training and inference of a 2L-768H-DeltaGRU network. $\Theta_x = \Theta_h$ (Θ is shown as Q8.8 integer values corresponding to 0∼0.5 floating point threshold).

the original nonlinear functions in FP32 precision. Training was done with PyTorch 1.2.0 on NVIDIA GPUs running CUDA 10 and cuDNN 7.6.

B. Experimental Setup: Network Implementation

After the quantized DeltaGRU is trained for a particular task, a Python script converts the PyTorch network modules into C/C++ hardware network header files. These files contain the network parameters and configuration register values for EdgeDRNN. By including the header files, bare-metal or PetaLinux applications are compiled using the standard cross compiler. The resulting system image is transferred to the QSPI flash (bare-metal) or eMMC storage (PetaLinux) on the MiniZed. In each timestep of the RNN, a feature vector is transferred from the PS to the accelerator using the AXI DMA. For measuring the performance of the accelerator, features are calculated offline on a computer and stored in a header file. For using the accelerator in real-world applications, features such as log filter bank and spike count features for audio, are calculated online by the ARM core in the PS. A flag connected to a PS hardware register is raised at the end of each timestep.

C. Accuracy & Throughput

1) Classification: Fig. 9 shows the EdgeDRNN throughput and WER on the *TIDIGITS* test set versus the Θ used in training and testing of a 2L-768H-DeltaGRU network. Θ is the same for both Δx and Δh. With $K = 8$ PEs and PL frequency $f_{pl} = 125$ MHz, EdgeDRNN has a theoretical peak throughput of $2Kf_{pl} = 2$ GOp/s. At $\Theta = 0$, there is still a speedup of about 2X from natural sparsity of the delta vectors. Higher Θ leads to better effective throughput, but with gradually increasing WER. The optimal point is at $\Theta = 64$ (0.25), just before a dramatic increase of WER, where EdgeDRNN achieves an effective throughput around 20.2 GOp/s with 1.3% WER. WER and throughput of smaller tested networks are shown in Table II. The 5-bit (Q1.4) LUT was used for this task and did not lead to accuracy loss compared to the network running on CPU with FP32 nonlinear functions.

2) Regression: In this regression task, we evaluate the impact of using different delta thresholds for Δx and Δh on the accuracy results of a 2L-256H-DeltaGRU model evaluated
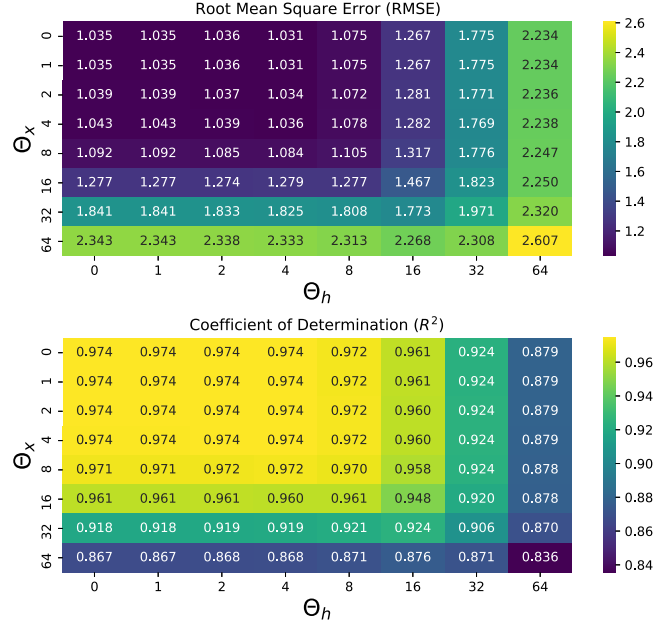

Fig. 10. RMSE (smaller is better) and R^2 (larger is better) versus Θ_x and Θ_h of the 2L-256H-DeltaGRU model evaluated on the *SensorsGas* test set.
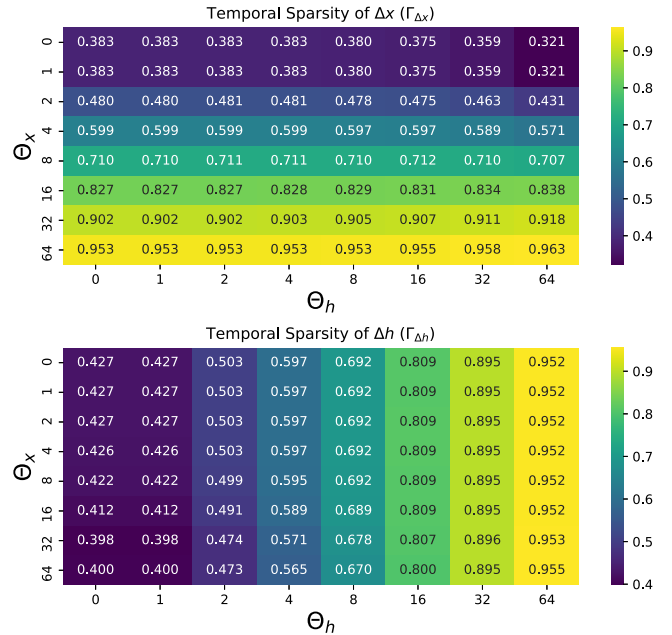

Fig. 11. Temporal sparsity $\Gamma_{\Delta x}$ and $\Gamma_{\Delta h}$ versus Θ_x and Θ_h of the 2L-256H-DeltaGRU model evaluated on the *SensorsGas* test set.

on the *SensorsGas* testset. Fig. 10 and Fig. 11 show respectively the regression accuracy and temporal sparsity versus Θ_x and Θ_h. The pretrained 2L-256H-GRU network without using a delta threshold, achieves a root-mean-square error (**RMSE**) of 0.995 and coefficient of determination (R^2) of 0.976. This accuracy was achieved using 5-bit (Q1.4) LUTs, which gave the lowest RMSE out of all other LUT bit precision values.

Similar to the results for the classification task, Fig. 10 shows that the accuracy degrades when larger delta thresholds are used. Fig. 11 shows that the sparsity levels of $\Gamma_{\Delta x}$ and $\Gamma_{\Delta h}$ are heavily influenced by their corresponding delta thresholds. The accuracy degrades faster with increasing Θ_x for a fixed Θ_h than with increasing Θ_h for a fixed Θ_x. Θ_x has a minor

Network Sizes	Op (Timestep)	WER	WER Degradation vs. GRU	Latency (μs)				Eff. Throughput (GOp/s)		MAC Utilization	$\Gamma_{\Delta x}$	$\Gamma_{\Delta h}$
				Min	Max	Mean	Est.	Mean	Est.			
1L-256H	0.5 M	1.83%	+1.36%	16.7	142.2	46.2	43.3	9.9	10.5	495%	25.6%	90.0%
2L-256H	1.2 M	1.13%	+0.69%	29.1	258.9	90.7	91.6	13.7	13.6	685%	78.9%	89.1%
1L-512H	1.7 M	1.04%	+0.44%	40.6	331.2	130.6	129.8	13.0	13.1	650%	25.6%	89.5%
2L-512H	4.9 M	0.89%	+0.75%	57.3	656.8	252.6	262.9	19.2	18.4	960%	85.5%	91.2%
1L-768H	3.7 M	1.27%	+0.11%	64.1	616.7	224.3	224.8	16.6	16.6	830%	25.6%	91.3%
2L-768H	10.8 M	0.77%	+0.53%	96.5	1344.5	535.6	541.6	20.2	19.9	1010%	87.0%	91.6%

Network Sizes	Op (Timestep)	Latency (μs)			Eff. Throughput (GOp/s)	MAC Utilization
		Min	Max	Mean	Mean	
1L-256H	0.5 M	17.0	311.0	48.2	9.5	475%
2L-256H	1.2 M	30.0	461.0	93.1	13.4	670%
1L-512H	1.7 M	42.0	603.0	133.6	12.7	635%
2L-512H	3.7 M	59.0	923.0	257.5	18.8	940%
1L-768H	4.8 M	66.0	627.0	228.5	16.3	815%
2L-768H	10.8 M	99.0	1366.0	544.9	19.8	990%

impact on $\Gamma_{\Delta h}$ and vice versa. The results from this regression task indicate that propagating changes more often in input states is more important than propagating changes in hidden states. By exploiting this phenomenon, we get the optimal point $(\Theta_x, \Theta_y) = (4, 8)$, where the RMSE and R^2 are 1.078 and 0.972 respectively. With $\Gamma_{\Delta x} = 59.7\%$ and $\Gamma_{\Delta h} = 69.2\%$, the latency of the optimal model is $206\,\mu$s. In comparison, Jetson TX2 runs a 4.8X smaller 1L-200H-GRU network in $271\,\mu$s [30].

D. Theoretical & Measured Performance

Eq. 7 gives the estimated mean effective throughput ν_{Eff} of EdgeDRNN running a DeltaGRU layer:

$$
\begin{aligned}
\nu_{Eff} &= \frac{\text{Op}}{\tau_m + \tau_a} \\
&\approx \frac{2\left(3HI + 3H^2(L-1) + 3H^2L\right)}{\frac{(3HI+3H^2(L-1))(1-\Gamma_{\Delta x})+3H^2L(1-\Gamma_{\Delta h})}{Kf_{pl}} + \frac{3H}{Kf_{pl}}}
\end{aligned}
\tag{7}
$$

where Op is the number of operations in a DeltaGRU layer per timestep, τ_m the latency of MxV, τ_a the latency of remaining operations to produce the activation, and the other variables are defined as in Eqs. 4 and 6.

Table II compares the Eq. 7 predictions with benchmark results of different DeltaGRU network sizes running on Edge-DRNN. Estimated results calculated from Eq. 7 are close to measured results and the maximum relative error between them is smaller than 7.1%. Thus Eq. 7 can be used to estimate EdgeDRNN performance for a particular RNN network size. On average, EdgeDRNN can run all tested networks under 0.54 ms latency corresponding to 20.2 GOp/s effective throughput for the 2L-768H-DeltaGRU.

E. Performance in PetaLinux

For performance measurements on the PetaLinux-based system, we implemented an application that performs the same operations as the software implemented for bare-metal but use the AXI DMA driver included in the OS.

Table III shows the latency and performance results for the 6 networks used in this work. The minimum and mean latency numbers in the PetaLinux version are respectively up to 3.4% and 11.3% higher than the numbers obtained for the bare-metal version. Because the minimum PetaLinux latency is nearly the same as the bare-metal latency, the big difference in maximum latency numbers between the PetaLinux and the bare-metal version is due to CPU contention for other tasks running on the PS that lock the single PS DDR controller. EdgeDRNN fetches weights from HP ports (Fig. 8) that are routed through the PS DDR controller. (The FPGA's ACP interface should not be used to access DRAM memory under PetaLinux because it is connected directly to the L2 cache on the ARM core where the OS runs. This configuration creates conflicts and the performance of the system is seriously compromised.) Under PetaLinux, the HP interface should be used to connect any module placed on the PL that requires direct access to the DRAM memory.

To understand the impact of CPU load and CPU DRAM access on the RNN inference time, we wrote a small program that loops over a memory array and is designed to trigger L2 cache misses. We used two different memory array sizes to study the effect of cache misses since the large memory array causes more L2 cache misses. Table IV shows that the impact on RNN latency is minor: a small network takes about 50% longer to run with either memory array size, and a large RNN is only slowed down by 10%.

The RNN inference time varies between 50 us to 0.5 ms across the different network sizes. During this inference time, the PS is free for other tasks (e.g. computing features) and only needs to check if the RNN update is finished when these tasks are completed.

F. Power Measurement

Table V shows the power breakdown of the MiniZed system. The total power is measured by a USB power meter; the PS, PL and static power is estimated by the Xilinx Power Analyzer. The whole system burns at most 2.3 W but the EdgeDRNN

	Mean Latency (us)
1L-256H-TH64 (little network with few parameters)	
Only EdgeDRNN	48
EdgeDRNN + Small memory workload	72
EdgeDRNN + Large memory workload	82
2L-768H-TH64 (big network with many parameters)	
Only EdgeDRNN	545
EdgeDRNN + Small memory workload	613
EdgeDRNN + Large memory workload	595

	Part	Wall Power (mW)	Percentage
PS	DDR3L	534	23.3%
	PLLs	388	16.9%
	ARM Cortex-A9	166	7.2%
	Peripherals	21	0.9%
Regulator/etc.		942	41.1%
Static		119	5.2%
EdgeDRNN		**66**	**2.9%**
DMA/Interconnect		54	2.4%
Total		2290	

only consumes 66 mW. It is interesting to note that the DRAM power is about 8X more than the RNN logic. This result clearly shows that the RNN computation is memory dominated.

V. COMPARISON

A. Comparison With FPGA RNN Accelerators

Table VI compares EdgeDRNN with other state-of-the-art FPGA RNN accelerators. Both BBS [10] and DeltaRNN were optimized for batch-1 inference by using all MACs for a single input sample. BBS can use DRAM to support large networks and has the highest batch-1 throughput among all accelerators; however the reported throughput number was obtained by buffering the whole network by using expensive on-chip memory. After compression, the network has around 0.8 MB parameters, which can be buffered on large FPGAs like the GX1150 used by BBS, but it is still too expensive for edge hardware platforms (e.g. MiniZed has only 0.2 MB on-chip memory). ESE [9] reuses weights fetched from off-chip memory to feed 1024 MACs for batch inference and achieved 2520 GOp/s total throughput; however only 32 out of 1024 MACs were used for each input sample limiting its batch-1 throughput. Except for EdgeDRNN and DeepRnn [14], other platforms are not designed for edge applications. BBS, DeltaRNN and ESE provide much higher throughput but their power consumption is around 3X-18X larger than EdgeDRNN and they require expensive FPGA development systems that are not very portable. By contrast, the small number of processing elements in EdgeDRNN is intentionally chosen to match the available memory bandwidth of the DRAM interface, since there is no point in having idle PEs.

To fairly compare architectures without the influence of different specifications of FPGA platforms, it makes sense to normalize the batch-1 throughput and other corresponding

numbers of accelerators to the same number of PEs ($K = 8$),[3] clock frequency ($f_{pl} = 125$ MHz), DRAM interface bit width for weight fetch (64-bit) and bit precision of weights (INT8) & activations (INT16) as used by EdgeDRNN. We also assume that the normalized platforms are implemented on MiniZed having the same power consumption of EdgeDRNN. The normalized batch-1 throughput $\nu_{\text{Eff,Norm}}$ is defined below:

$$\nu_{\text{Peak,Mem}} = 2 \cdot f_{pl} \cdot \frac{W_{\text{DRAM}}}{W_{\text{Weight}} + W_{\text{Index}}}$$

$$\nu_{\text{Eff,Norm}} = \nu_{\text{Peak,Mem}} \cdot \frac{1}{1 - \Gamma_{\text{Eff}}} \qquad (8)$$

where $\nu_{\text{Peak,Mem}}$ is the memory-bounded peak throughput and W_{Index} is the bit width of the nonzero element index. To exploit weight sparsity by skipping zero elements in the weights, indices of nonzero weight elements have to be used and introduces off-chip memory overhead. Both BBS and ESE use $W_{\text{Index}} = 4$ for their tested networks. EdgeDRNN and DeltaRNN only need indices of valid columns corresponding to nonzero delta state vector elements, and they are calculated on-chip without introducing off-chip memory overhead; thus, $W_{\text{Index}} = 0$ for EdgeDRNN and DeltaRNN. In this normalization process, we assume the ideal case, in which normalized platforms reach the memory-bounded peak throughput and can fully utilize sparsity. Thus, Eq. 8 gives the upper bound throughput value of the normalized platform.

Table VI shows that EdgeDRNN achieves the highest normalized throughput, and an even higher normalized throughput than our previous BRAM-based DeltaRNN because of the improved pipeline and higher sparsity achieved. Compared with BBS, EdgeDRNN achieves only a small fraction of the total batch-1 throughput, but the normalization makes it clear that BBS achieves its high throughput by using on-chip BRAM, a huge number of MACs, and a higher clock frequency. Among all the accelerators, EdgeDRNN also shows the highest effective MAC utilization and the lowest wall plug power. Finally, the EdgeDRNN FPGA development kit is a factor of at least 25X cheaper than other FPGA RNNs, and the cost is comparable to the cheapest edge AI accelerators.

B. Architectural Comparison

Fig. 12 compares the architecture and MxV flow of Edge-DRNN with BBS and ESE. We compare EdgeDRNN with ESE and BBS because they are also FPGA RNN accelerators using DRAM with high reported throughput. Both ESE and BBS exploit weight sparsity with load balancing techniques.

1) ESE: In ESE, interleaving rows of matrix elements are assigned to MAC units in the PE array and the MxV is computed column by column. To balance the workload and exploit weight sparsity better, the network weight matrix is pruned so that the number of nonzero elements assigned to each MAC unit is the same for the whole matrix. K activation buffers (**BUF**) are required for K MAC units, which immediately execute operations when nonzero activations and weights are available.

---

[3]Each PE has a single MAC unit.

TABLE VI

COMPARISON WITH STATE-OF-THE-ART FPGA RNN ACCELERATORS

Platform	This Work	BBS [10]	DeltaRNN [11]	ESE [9]	DeepRnn [14]
FPGA	XC7Z007S	Arria 10 GX1150	XC7Z100	XCKU060	XC7Z045
Dev. Kit Cost	$89	$4,495	$2,295	$3,295	$2,495
Weight Storage	Off-chip	On-chip & off-chip	On-chip	Off-chip	On-chip & off-chip
Bit Precision (Activation/Weight/Index)	INT 16/8/0	INT 16/16/4	INT 16/16/0	INT 16/12/4	INT 16/16/0
Sparsity Type	Temporal	Weight	Temporal	Weight	-
[1]Effective Sparsity	90.0%	87.5%	88.2%	88.7%	-
Frequency (MHz)	125	200	125	200	142
DRAM Interface Bit Width for Weight Fetch	64	-	-	512	-
Number of MACs (Batch-1)	8	4096	768	32	256
[2]Peak Throughput (GOp/s)	2	**1638.4**	192	12.8	4.5
Effective Batch-1 Throughput (GOp/s)	20.2	**2432.8**	1198.3	78.8	0.7
[3]MAC Utilization	**1010%**	150%	630%	620%	15%
[4]Memory-bounded Peak Throughput (GOp/s)	**2.0**	1.3	2.0	1.3	2.0
[5]Normalized Effective Batch-1 Throughput (GOp/s)	20.2	≤ 10.7	≤ 17.0	≤ 11.5	≤ 2
Wall Plug Power (W)	**2.3**	19.1	7.3	41.0+PC	-
Batch-1 Wall Plug Power Efficiency (GOp/s/W)	8.8	127.4	**164.2**	1.9	-
[5]Normalized Batch-1 Wall Plug Power Efficiency (GOp/s/W)	**8.8**	≤ 4.7	≤ 7.4	≤ 5.0	-

[1] The effective sparsity of EdgeDRNN & DeltaRNN is calculated by Eq. 4.
[2] Peak throughput is calculated by Eq. 6.
[3] MAC utilization is the ratio between batch-1 throughput and the peak throughput of the accelerator.
[4] Memory-bounded peak throughput is calculated by Eq. 8.
[5] Normalized to the same frequency, DRAM interface bit width for weight fetch, number of MACs and activation & weight bit precision with EdgeDRNN. We assume the normalized numbers are obtained on the same MiniZed board and assume they have the same wall plug power consumption with EdgeDRNN. Detailed discussion is in Section V.A.
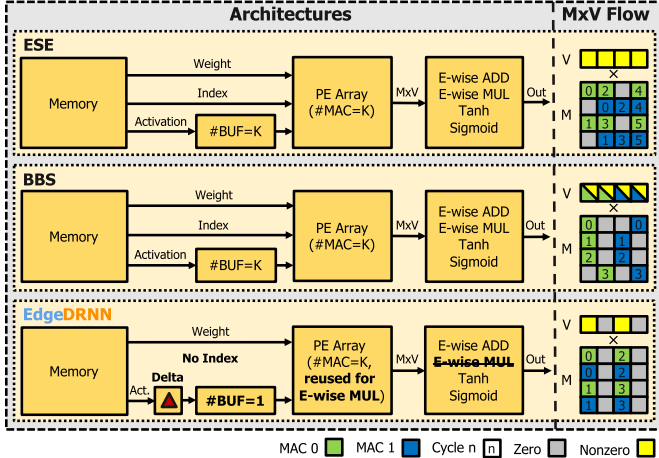

Fig. 12. Architecture comparison and MxV Flow of ESE, BBS and EdgeDRNN. Bold text indicates differences. Colors in the MxV Flow part indicate MAC unit, and zero and nonzero activation values. The text indicates the clock cycle.

2) BBS: BBS balances the workload using structured pruning. Rows of a weight matrix are split into banks of equal length. Their pruning method forces the numbers of nonzero values to be the same across all banks. By assigning the same number of row banks to each MAC unit in the PE array, the workload is balanced. As shown on the right side of Fig. 12, each row of the matrix is equally divided into two banks respectively for MAC 0 and MAC 1 and the computation is done row by row. In this case, each MAC receives different activation elements and K BUFs are required for K MAC units. BBS also supports the buffering of a partial weight

matrix on-chip to enhance throughput, which is useful for large FPGA platforms. The reported batch-1 throughput of BBS in Table VI is obtained with all network parameters on-chip, which is not practical on a small FPGA platform like MiniZed that has only 0.2 MB on-chip memory.

3) EdgeDRNN: Unlike ESE and BBS, EdgeDRNN includes an extra unit to compute delta state vectors. Similar to ESE, EdgeDRNN also assigns interleaving rows to MAC units and computes MxV column by column; however, all MAC units share the same delta state vector elements; thus, only 1 BUF (D-FIFO) is required.

Both ESE and BBS require indices of nonzero weight elements to realize zero-skipping. The indices cause overhead on memory access, reducing effective memory bandwidth. EdgeDRNN skips whole columns of computation and indices of valid columns are calculated on-the-fly to avoid memory overhead.

Moreover, ESE and BBS require extra Element-wise (**E-wise**) multiplication units for the RNN activation h_t generation after MxV. EdgeDRNN reuses multipliers in the PE array by time-division multiplexing to save DSP and LUT resources. Element-wise addition is done by reusing adders in the PE array and also using a single 16-bit adder per PE, as shown in Fig. 5.

Our previous work, DeltaRNN [11], achieved high batch-1 throughput and MAC utilization with temporal sparsity, but it stored all network parameters on chip, making it unscalable. Meanwhile, EdgeDRNN is designed to match the external memory bandwidth available on any FPGA platform with external DRAM. The small number of MAC units along tall

Platform	This Work			NCS2 [34]	Jetson Nano [35]		Jetson TX2 [36]		GTX 1080 [37]	
Chip	XC7Z007S			Myriad X	Tegra X1		Tegra X2		GP104	
Dev. Kit Cost	$89			$69	$99		$411		$500+PC	
DRAM Type (Bus Width)	DDR3L (16-bit)			-	LPDDR4 (64-bit)		LPDDR4 (128-bit)		GDDR5X (256-bit)	
DRAM Bandwidth (GB/s)	1.0			-	25.6		59.7		320	
Test Network	2L-768H-DeltaGRU			2L-664H-LSTM	2L-768H-GRU					
#Parameters	5.4 M			5.4 M	5.4 M					
Bit Precision (A/W)	INT16/8			FP16	FP32	FP16	FP32	FP16	FP32	FP16
WER on TIDIGITS	$\Theta = 0x00$ 0.7%	$\Theta = 0x08$ 0.8%	$\Theta = 0x40$ 1.3%	1.1%	0.8%					
Latency (μs)	2633	1673	**536**	3,588	5,757	4,356	3,124	2,693	**527**	484
Effective Batch-1 Throughput (GOp/s)	4.1	6.5	**20.2**	3.0	1.9	2.5	3.5	4.0	**20.5**	22.3
[1]Wall Plug Power (W)	2.3			**1.7**	7.2	7.1	8.2	8.1	96.6+PC	82.2+PC
Batch-1 Wall Plug Power Efficiency (GOp/s/W)	1.8	2.8	**8.8**	1.8	0.3	0.4	0.4	0.5	0.2	0.3

[1] EdgeDRNN power was measured by a USB power meter. Power numbers of Jetson Nano and Jetson TX2 boards are measured by a Voltcraft 4500ADVANCED Energy Monitor. Power of GTX 1080 was measured by the *nvidia-smi* utility.
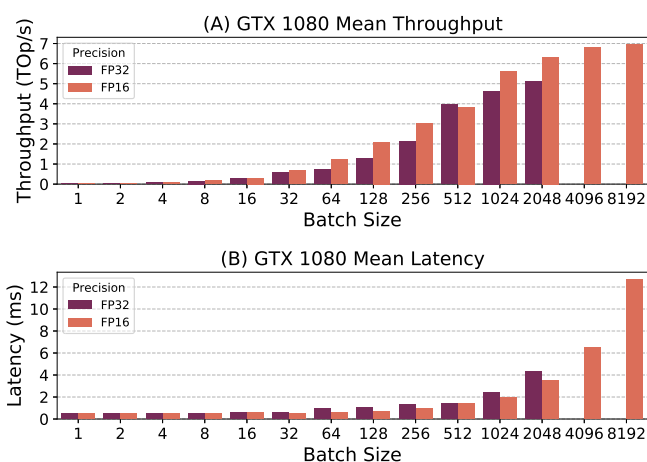

Fig. 13. Measured mean throughput (A) & latency (B) of GTX 1080 versus different batch sizes up to the maximum batch size that fits into the available DRAM.

Fig. 14. Measured batch-1 latency per frame of a sample (25896O4A.WAV) from the *TIDIGITS* test set benchmarked on EdgeDRNN (INT16 & INT8) and GPUs (FP16) with the first 50 timesteps excluded.

concatenated weight matrix columns, as shown in Fig. 6, makes the burst length long enough to maintain high DRAM controller efficiency for large networks.

C. Comparison With an SNN Processor

We compare the performance metrics of TrueNorth [38], an application-specific integrated circuit (**ASIC**) SNN processor, on the TIDIGITS dataset. The system is reported to dissipate 38.6 mW power by using a feature extraction method that can be implemented directly on TrueNorth [39]. We cannot easily compare the power numbers of this ASIC processor with the power dissipated by an FPGA which is a more general-purpose platform. To run TrueNorth, an interfacing FPGA that burns several Watts is needed so the system power would much higher. The reported accuracy from their work is only 95% which is lower than the 99% accuracy achieved by the quantized delta network reported in our previous work [12].

D. Comparison With Commercial Platforms

Table VII compares EdgeDRNN with popular commercial platforms, including their cost and memory system bandwidth. All platforms are 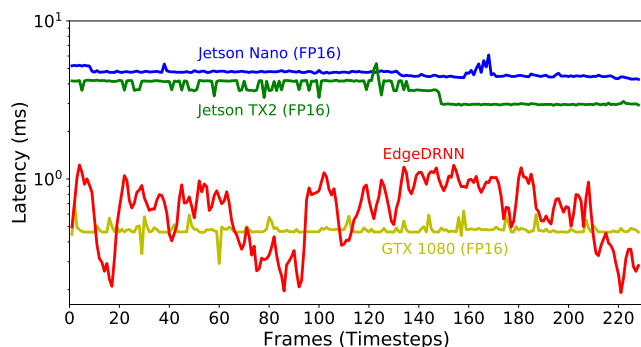benchmarked on the same spoken digit recognition task (first 10,000 timesteps of the *TIDIGITS* test set) using networks of the same size, except that the Intel Compute Stick 2 (NCS2) does not support GRU and was benchmarked with an LSTM network with a similar parameter count and trained with the same hyperparameters. The latency requirement of the recognition task is 10 ms which is determined by the frame stride. To meet this requirement, frames cannot be concatenated into a single tensor. The computation of the RNN is executed when there is a new frame.

For benchmark of GPUs, we used GRUs because we found that latency numbers of both FP32 and FP16 cuDNN GRU implementations are 3X lower than that of running the DeltaGRU algorithm using the NVIDIA cuSPARSE library. In addition, we removed peripheral devices from the Jetson board with the exclusion of the needed Ethernet cable to the PC. Because GPUs also need time to boost their clock frequency and to allocate memory, the first 50 timesteps of the test sequence are excluded. The power efficiency results show that EdgeDRNN still achieves over 5X higher system power efficiency compared to commercial ASIC and GPU products.

GPUs are throughput-oriented architectures suitable for neural network training with large batch sizes; however, it is not optimal for edge inference where batch-1 throughput is critical for achieving low latency. The claimed peak FP32 throughput of Jetson Nano [35], Jetson TX2 [36] and GTX 1080 [37] are respectively 0.5 TOp/s, 0.8 TOp/s and 9 TOp/s
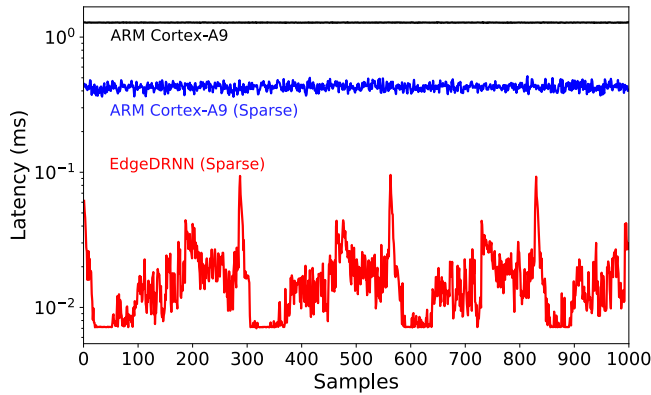
Fig. 15. Measured hardware latency per sample on EdgeDRNN and embedded processor using the 2L-128H-DeltaGRU network used in the real-time control demonstration of AMPRO [7].

while the measured batch-1 throughput are only 1.9 GOp/s, 3.5 GOp/s and 20.5 GOp/s. The low batch-1 throughput of GPUs is because weights fetched from off-chip DRAM cannot be reused to fully utilize GPU cores. Fig. 13A shows the throughput of GTX 1080 approaches the claimed peak throughput with large batch sizes due to more weight data reuse; however, increasing batch size also causes worse latency numbers as shown in Fig. 13B. FP16 outperforms FP32 because of the smaller memory bottleneck.

Fig. 14 compares latency per frame on a test set sample. EdgeDRNN latency is lower during the silent or quieter periods (e.g. between 60 s and 80 s) when the input is changing slowly. EdgeDRNN is as quick as the desktop 1080 GPU and 5X quicker than the other platforms, despite having a DRAM bandwidth that is orders of magnitude slower.

In [7], we reported that EdgeDRNN ran the RNN for robotic control about 51X faster than the embedded BeagleBone Black platform with a ARM Cortex-A8 CPU, while burning about the same total power of 2W. Moreover, to compare the performance of EdgeDRNN and the ARM Cortex-A9 CPU on the PS side of MiniZed, we took the same 2L-128H-DeltaGRU network used in our previous real-time control demonstration [7] and measured the latency per frame on 1 minute test data (sample rate = 200 Hz), which are 1000 frames of motor encoder readings. Fig. 15 shows the latency of the ARM CPU and EdgeDRNN. The mean latency of the ARM CPU is 1281 μs without sparsity and 428 μs with sparsity. The mean latency of EdgeDRNN with sparsity is 16 μs, therefore EdgeDRNN is 27X faster than the ARM CPU which exploits temporal sparsity in the same network. In the case of the robotic task, EdgeDRNN runs the network 300X faster than the required maximum latency of 5ms.

VI. CONCLUSION

The 2 W EdgeDRNN runs batch-1 RNNs as fast as a 200 W GPU+PC, and its power efficiency is at least a factor of 4X higher than any of the commercial edge AI platforms in the benchmark. We found that the batch-1 RNN throughput numbers of commercial GPUs are a factor of over 100X less than their claimed peak throughput. Using the delta network to exploit temporal sparsity allows a modest number of 8 PEs to achieve an effective 162 Op per clock cycle,
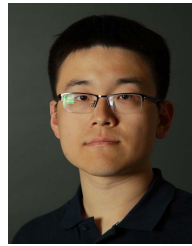
equivalent to an MAC utilization efficiency of over 1000%. EdgeDRNN uses a standard AXI4 interface for weight fetches; thus it can be scaled up to larger FPGA platforms by simply increasing the number of PEs to match the available memory bandwidth provided by on-chip BRAM or off-chip DRAM. Thus using temporal sparsity in delta activation vectors allows the arithmetic units on this task to effectively compute 10X more operations with the same amount of memory access.

The delta threshold Θ allows instantaneous trade-off of accuracy versus latency. Future work could exploit a dynamic trade-off of accuracy versus latency to quickly converge onto optimal values in a guided search.

REFERENCES

[1] A. Graves, "Sequence transduction with recurrent neural networks," in *Proc. ICML Represent. Learn. Workshop*, 2012. [Online]. Available: https://sites.google.com/site/representationworkshopicml2012/schedule/alex-graves

[2] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[3] K.-I. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Netw.*, vol. 6, no. 6, pp. 801–806, Jan. 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S089360800580125X

[4] T. W. S. Chow and Y. Fang, "A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics," *IEEE Trans. Ind. Electron.*, vol. 45, no. 1, pp. 151–161, Feb. 1998.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[6] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734. [Online]. Available: http://www.aclweb.org/anthology/D14-1179

[7] C. Gao, R. Gehlhar, A. D. Ames, S.-C. Liu, and T. Delbruck, "Recurrent neural network control of a hybrid dynamical transfemoral prosthesis with EdgeDRNN accelerator," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 5460–5466.

[8] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[9] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2017, pp. 75–84.

[10] S. Cao *et al.*, "Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*. New York, NY, USA: Association Computing Machinery, Feb. 2019, pp. 63–72, doi: 10.1145/3289602.3293898.

[11] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "DeltaRNN: A power-efficient recurrent neural network accelerator," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*. New York, NY, USA: Association Computing Machinery, Feb. 2018, pp. 21–30, doi: 10.1145/3174243.3174261.

[12] D. Neil, J. Lee, T. Delbrück, and S.-C. Liu, "Delta networks for optimized recurrent network computation," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Sydney, NSW, Australia, Aug. 2017, pp. 2584–2593. [Online]. Available: http://proceedings.mlr.press/v70/neil17a.html

[13] C. Gao, A. Rios-Navarro, X. Chen, T. Delbruck, and S.-C. Liu, "Edge-DRNN: Enabling low-latency recurrent neural network edge inference," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Sep. 2020, pp. 41–45.

[14] A. X. M. Chang and E. Culurciello, "Hardware accelerators for recurrent neural networks on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[15] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "FPGA-based low-power speech recognition with recurrent neural networks," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2016, pp. 230–235.

[16] Y. Guan *et al.*, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2017, pp. 152–159.

[17] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[18] M. Yang, C.-H. Yeh, Y. Zhou, J. P. Cerqueira, A. A. Lazar, and M. Seok, "A 1 μW voice activity detector using analog feature extraction and digital deep neural network," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2018, pp. 346–348.

[19] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, "Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms," *Frontiers Neurosci.*, vol. 9, p. 222, Jul. 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00222

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[21] D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, and J.-S. Seo, "An 8.93 TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity for on-device speech recognition," *IEEE J. Solid-State Circuits*, vol. 55, no. 7, pp. 1877–1887, Jul. 2020.

[22] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[23] O. Bilaniuk, S. Wagner, Y. Savaria, and J.-P. David, "Bit-slicing FPGA accelerator for quantized neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[24] Z. Wang, J. Lin, and Z. Wang, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2763–2775, Oct. 2017.

[25] S. Wang *et al.*, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*. New York, NY, USA: ACM, Feb. 2018, pp. 11–20, doi: 10.1145/3174243.3174253.

[26] C. Gao, S. Braun, I. Kiselev, J. Anumula, T. Delbruck, and S.-C. Liu, "Real-time speech recognition for IoT purpose using a delta recurrent neural network accelerator," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[27] R. G. Leonard and G. Doddington, *Tidigits Speech Corpus*. Dallas, TX, USA: Texas Instruments, 1993. [Online]. Available: https://catalog.ldc.upenn.edu/LDC93S10

[28] J. Burgués, J. M. Jiménez-Soto, and S. Marco, "Estimation of the limit of detection in semiconductor gas sensors through linearized calibration models," *Analytica Chim. Acta*, vol. 1013, pp. 13–25, Jul. 2018.

[29] J. Burgués and S. Marco, "Multivariate estimation of the limit of detection by orthogonal partial least squares in temperature-modulated MOX sensors," *Analytica Chim. Acta*, vol. 1019, pp. 49–64, Aug. 2018.

[30] S. Wang, Y. Hu, J. Burgues, S. Marco, and S.-C. Liu, "Prediction of gas concentration using gated recurrent neural networks," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 178–182.

[31] Xilinx. *AXI Datamover*. Accessed: Jun. 30, 2020. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_datamover.html

[32] AVNET. *MiniZed*. Accessed: Jun. 30, 2020. [Online]. Available: http://zedboard.org/product/minized

[33] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*. New York, NY, USA: ACM, 2006, pp. 369–376, doi: 10.1145/1143844.1143891.

[34] *Intel Neural Compute Stick 2 Product Specifications*. Accessed: Jun. 30, 2020. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html

[35] (Mar. 2019). *Jetson Nano Developer Kit*. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[36] (Aug. 2019). *Harness AI at the Edge With the Jetson TX2 Developer Kit*. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2-developer-kit

[37] *GeForce GTX 1080 Graphics Cards From NVIDIA GeForce*. Accessed: Jun. 30, 2020. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/

[38] F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[39] W.-Y. Tsai *et al.*, "Always-on speech recognition using TrueNorth, a reconfigurable, neurosynaptic processor," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 996–1007, Jun. 2017.
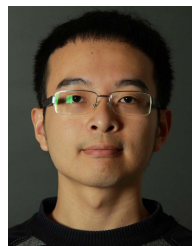
Chang Gao (Student Member, IEEE) received the B.Eng. degree in electronics from the University of Liverpool, Liverpool, U.K., and Xi'an Jiaotong-Liverpool University, Suzhou, China, and the master's degree in analogue and digital integrated circuit design from the Imperial College London, London, U.K. He is currently pursuing the Ph.D. degree with the Institute of Neuroinformatics, University of Zurich and ETH Zürich, Zürich, Switzerland. His current research interest includes computer architectures for deep learning with an emphasis on recurrent neural networks.

Antonio Rios-Navarro (Member, IEEE) received the B.S. degree in computer science engineering, the M.S. degree in computer engineering, and the Ph.D. degree in neuromorphic engineering from the University of Seville, Seville, Spain, in 2010, 2011, and 2017, respectively. He currently holds a post-doctoral position at the Computer Architecture and Technology Department, University of Seville. His current research interests include neuromorphic systems, real-time spikes signal processing, field-programmable gate array design, and deep learning.

Xi Chen (Student Member, IEEE) received the B.Eng. degree in electrical engineering and automation from Tianjin University, Tianjin, China, the B.Eng. degree in electronic and computer systems from the University of Kent, Canterbury, U.K., and the M.Sc. degree in analogue and digital integrated circuit design from the Imperial College London, U.K. He is currently pursuing the Ph.D. degree with the Institute of Neuroinformatics, University of Zurich and ETH Zürich, Zürich, Switzerland. His current research interests include hardware acceleration for deep learning and hardware architectures for on-chip training of artificial neural networks.

Shih-Chii Liu (Senior Member, IEEE) received the bachelor's degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, and the Ph.D. degree in the computation and neural systems program from the California Institute of Technology, Pasadena, CA, USA, in 1997. She worked with Silicon Valley before returning for her Ph.D. degree. She is currently a Professor with the University of Zurich, Zürich, Switzerland. Her group focuses on audio sensors particular the spiking cochlea and bio-inspired deep neural network algorithms and hardware.

Tobi Delbruck (Fellow, IEEE) received the B.Sc. degree in physics from the University of California at San Diego, San Diego, CA, USA, in 1986, and the Ph.D. degree from the California Institute of Technology, Pasadena, CA, USA, in 1993. Since 1998, he has been with the Institute of Neuroinformatics, University of Zurich and ETH Zürich, Zürich, Switzerland, where he is currently a Professor of physics and electrical engineering. His group focuses on neuromorphic sensory processing and efficient deep learning.