Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars

Alexandros Kouris
Imperial College London

Stylianos I. Venieris
Samsung AI Center

Michail Rizakis
Imperial College London

Christos-Savvas Bouganis
Imperial College London

***Abstract*—The need to recognize long-term dependencies in sequential data, such as video streams, has made long short-term memory (LSTM) networks a prominent artificial intelligence model for many emerging applications. However, the high computational and memory demands of LSTMs introduce challenges in their deployment on latency-critical systems such as self-driving cars, which are equipped with limited computational resources on-board. In this article, we introduce a progressive inference computing scheme that combines model pruning and computation restructuring leading to the best possible approximation of the result given the available latency budget of the target application. The proposed methodology enables mission-critical systems to make informed decisions even in early stages of the computation, based on approximate LSTM inference, meeting their specifications on safety and robustness. Our experiments on a state-of-the-art driving model for autonomous vehicle navigation demonstrate that the proposed approach can yield outputs with similar quality of result compared to a faithful LSTM baseline, up to 415× faster (198× on average, 76× geo. mean).**

Digital Object Identifier 10.1109/MCE.2020.2969195
Date of current version 8 June 2020.

Recurrent neural networks (RNNs) are a family of machine learning models with the ability to recognize patterns in sequential and temporal data. In the past decade, long short-term memory (LSTM) networks[1] have emerged as the dominant RNN by setting the state-of-the-art record in various AI tasks, such as machine translation and video understanding. Among the various LSTM-enabled applications, time-constrained mission-critical systems[2] are rapidly becoming an ubiquitous scenario. In this setting, AI agents are equipped with LSTM-based mechanisms of sensing, perceiving and, eventually, acting.[3] In such scenarios, making the most informed decision under a limited time budget is of vital importance in order to ensure the robust, safe, and successful operation of the system within complex and uncertain environments.[4]

Figure 1 depicts an example of such a latency-critical system. In this case, a driverless car navigates autonomously in an urban environment under the control of an LSTM that predicts the desired throttle/brake position and steering angle based on the input video sequence. With the human driver reaction time ranging between 0.7 and 3 s (varying with situation and individual person),[5] autonomous driving systems target a relevant low-latency envelope to take action from the moment an event occurs on the road, in order to preserve the ability of achieving comparable reliability with humans. In this respect, extracting the best possible approximation of the desired action to be commanded within the real-time latency constraints is preferred from a more accurate decision later in time.

From a technical viewpoint, performing the most informed action under a time budget reduces to the problem of obtaining the highest quality output from an LSTM given a constraint in computation time. Current methods for deploying LSTMs follow the behavior depicted in Figure 2. Conventional implementations[7,8] require the whole inference computation to finish in order to obtain meaningful information from the LSTM and, thus, prolong the sensing-to-action loop with potentially catastrophic effects. Instead, the stringent latency deadlines of real-life systems call for *progressive inference* designs that can provide the best possible estimate of the final output for a given time budget and improve on it as more time

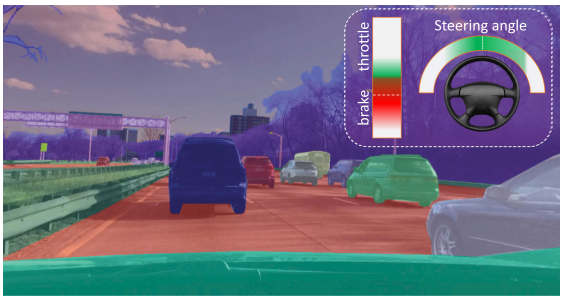

Figure 1. Throttle/brake and steering angle prediction for autonomous driving with an LSTM model (trained on the dataset in the literature[6]) relying on visual inputs. *Video & Webpage:* www.imperial.ac.uk/intelligent-digital-systems/approx-lstms/

budget becomes available (see Figure 2). This property would enable the agent to exploit the maximum possible amount of information that is available in the current input and effectively optimize its overall operation.

From a workload perspective, LSTMs are challenging by being memory-bound. This property means that the performance of brute-force implementations is limited by the available memory bandwidth of the platform, rather than by the available computational power. Furthermore, the excessive memory accesses and the inefficient use of computational resources when executing LSTMs on conventional platforms leads to substantial power inefficiencies, which are critical for battery-operated systems. To attack this issue, recent works deviated from general-purpose computing platforms and adopted a *model-hardware codesign* approach for the generation of custom hardware architectures.[9]

Field-programmable gate arrays (FPGAs) typically consist of one or more processors and a reconfigurable fabric. The processor is responsible for executing noncritical code and coordinates the operation of the overall system. The reconfigurable fabric can be customized at the hardware level, allowing the on-chip computational and memory resource allocation to be optimized to match the particular workload and the performance needs of the target application and its underlying implementation. Enabled by the customization and flexibility of FPGAs, the works below propose different approximation techniques, focusing on model compression,[10] quantization,[11] and pruning,[12] together with an associated

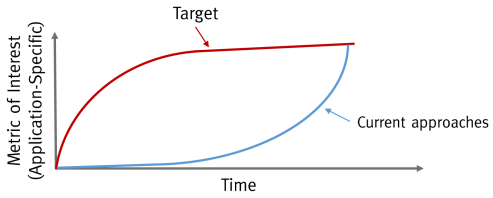

Figure 2. Concept of progressive inference: Conventional and target behavior of time-constrained AI systems. The *y*-axis metric reflects the application-level accuracy (higher-is-better).

FPGA-based hardware accelerator, tailored to the computational needs of the model and its approximate computing scheme, to match the computational demands of LSTMs.

Despite the effectiveness of these methods, their application requires a *retraining step*, which allows the refinement of the model in order to compensate for any approximation losses in the model's accuracy. For the retraining step to be feasible, availability of the training set is required, which is not a realistic assumption in privacy-aware applications,[13] as in the case of large-scale datasets collected by commercial companies that remain proprietary, or medical-oriented institutions that are prevented by confidentiality regulations from sharing their clinical datasets, making privacy-preserving AI techniques increasingly relevant.[14–16]

In this context, we propose a novel methodology for the high-performance (HP) deployment of LSTMs in time-constrained applications, which is also complementary to the existing approaches. The proposed approximate computing scheme is implemented on custom hardware, also exploiting the customization and flexibility of FPGAs. The goal is to generate an optimized hardware mapping of a given LSTM on a target FPGA, tailored to the available time budget and error tolerance. To meet the needs of this task, an iterative scheme is introduced that exploits the resilience of the target application to approximations in order to relax the computational and memory requirements of the given LSTM, and executes the model under time constraints, with increasing accuracy as a function of the time budget.

In this work, we showcase a significantly improved computation-time, accuracy, and power tradeoff presented by our progressive inference scheme that effectively reduces the

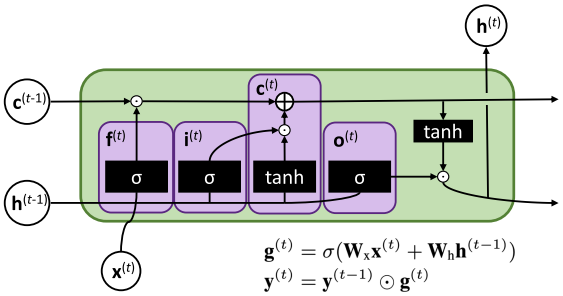

Figure 3. Structure of an LSTM model. $\mathbf{g}^{(t)}$ represents each of the LSTM gates ($\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}$), while $\odot$ denotes the elementwise multiplication between two vectors defined as $(\mathbf{a} \odot \mathbf{b})_i = \mathbf{a}_i\mathbf{b}_i$.

computational workload of a given LSTM model to meet the desired quality of result (QoR), compared to a baseline implementation of the same model, while both designs are exploiting the customization capabilities of an FPGA. The experimental evaluation of the proposed approach is conducted on a state-of-the-art driving model for autonomous vehicles. Self-driving cars, being tightly coupled with the recent developments in consumer electronics,[17,18] form a representative example of a system with tight computation time budget to make mission critical decisions, while being also constrained in a limited computational resource environment. At the same time, autonomous driving is emerging alongside with the revolution of electric vehicles, imposing a low-power envelope for the deployment of increasingly compute-hungry models.[19] This makes special-purpose FPGA-based hardware architectures the most prominent solution, offering high computational efficiency for deployment on resource- and power-constrained environments.

LEARNING LONG-TERM PATTERNS WITH LSTMS

LSTMs are specialized RNNs with enhancements that enable the learning of long-term dependencies. The key feature of an LSTM is a set of units named *gates,* which control its behavior at run time. Figure 3 depicts the structure of an LSTM. The core element of LSTMs is the cell state c, shown along the horizontal line at the top of the diagram. At each time step t, the LSTM removes or adds information to the cell state via its gate modules. Computationally, a gate receives as

inputs the new input sample $\mathbf{x}^{(t)}$ and the previous output $\mathbf{h}^{(t-1)}$ and performs a matrix–vector multiplication with the weight matrices $\mathbf{W}_{\mathrm{x}}$ and $\mathbf{W}_{\mathrm{h}}$, as described in Figure 3. The elements of the weight matrices are learned during the training stage of the target application and remain fixed throughout the inference stage that takes place upon deployment.

Next, the resulted vector of the matrix–vector multiplication is passed through a nonlinear function, such as a sigmoid $\sigma(\cdot)$, to form $\mathbf{g}^{(t)}$. The nonlinear function operates in an element-by-element fashion and outputs a vector with values between 0 and 1, capturing how much of each element should be kept. A value of 0 represents total forgetting of information, 1 represents total propagation, and intermediate values dictate what fraction of the information should be kept. In this manner, by multiplying element-by-element another vector $\mathbf{y}^{(t-1)}$ with the output of the nonlinear function, a new vector $\mathbf{y}^{(t)}$ is produced, which is a filtered version of its previous state (see Figure 3).

An LSTM consists of four gates. Starting from the left of the diagram in Figure 3, the *forget* gate $\mathbf{f}^{(t)}$ determines the amount of information that will be forgotten from the previous cell state $\mathbf{c}^{(t-1)}$. Next, the *input* gate $\mathbf{i}^{(t)}$ and the *cell* gate determine the new information to be stored in the new cell state $\mathbf{c}^{(t)}$. The cell gate employs tanh for its nonlinear function and creates a vector of new candidate values for the new cell state, whereas the input gate controls which values of the current cell state will be updated. At this point, the new cell state $\mathbf{c}^{(t)}$ has been formed. The final step involves the calculation of the new output vector $\mathbf{h}^{(t)}$, which is a filtered version of the cell state. This is generated by passing the cell state through a tanh nonlinearity and multiplying the result with the output of the *output* gate $\mathbf{o}^{(t)}$ in order to update only parts of the cell state.

APPROXIMATE COMPUTING FOR LSTMS

At the core of an LSTM's workload lies the linear algebra operation of matrix–vector multiplication, shown on the first line in Figure 3, which takes place in each of the four gates. Neural networks have been extensively studied to have redundancy in terms of their trained parameters.[20] This property allows the restructuring of the computations of LSTM gates in such a manner that enables us to extract the maximum information at any time instant. In this respect, we propose an approximate computing scheme that enables the tuning of the QoR in exchange for an increase in performance. The proposed approach exploits the statistical redundancy of LSTMs by acting at two levels: 1) approximating weight matrices with a low-rank singular-value decomposition (SVD) and 2) pruning the network by sparsifying the weight matrices based on an importance criterion of their elements. These techniques enable us to restructure the computations of an LSTM and design a computing system that performs the most information-carrying computations, first, in order to obtain the peak QoR given a time budget.

Information-Maximizing Approximation

Each LSTM gate consists of two weight matrices corresponding to the current input and previous output, respectively. In our scheme, we first concatenate the two weight matrices and the input and output vectors to obtain a single augmented matrix and vector, respectively, for each gate as $\mathbf{W} = [\mathbf{W}_{\mathrm{x}}\mathbf{W}_{\mathrm{h}}] \in \mathbb{R}^{R\times C}$ and $\tilde{\mathbf{x}}^{(t)} = [\mathbf{x}^{(t)\top}\mathbf{h}^{(t-1)\top}]^{\top} \in \mathbb{R}^{C\times 1}$. As a next step, we substitute the augmented weight matrix with a low-rank approximation that reduces the computation and memory footprint cost while minimizing the information loss. These properties are satisfied by the rank-1 approximation of each weight matrix based on the SVD. This approach enables us to approximate the weight matrix as the outer product of two vectors (the singular vectors) followed by an elementwise multiplication with a constant number (the singular value). For the ith gate, the rank-1 approximate weight matrix is given by $\widetilde{\mathbf{W}}_i = \sigma_1^i \mathbf{u}_1^i \mathbf{v}_1^{i\top}$. With respect to the computational cost, the original matrix vector multiplication $\widetilde{\mathbf{W}}_i\tilde{\mathbf{x}}^{(t)}$ is replaced by a dot product followed by an elementwise multiplication between a vector and a constant number, i.e., $\sigma_1^i \mathbf{u}_1^i(\mathbf{v}_1^{i\top}\tilde{\mathbf{x}}^{(t)})$, leading to a significant reduction on both the number of operations and the memory footprint of the weight matrix, while retaining the highest amount of information that a rank-1 approximation can have.

Pruning by Means of Network Sparsification

The second level of approximation on the LSTM comprises the structured pruning of the weight matrices at each gate. Pruning can be interpreted as a type of sparsity in which individual weights are masked as zeros. In our structured pruning scheme, we limit sparsity to the structure of rows of the weight matrices. This selection of granularity allows us to always obtain an approximate value for each element of the resulted output vector, instead of having zeroed values at the output vector that carry no information. Individual weight values are set to zero by means of a magnitude-based criterion, which determines the importance of a weight using its absolute value. Overall, the pruning scheme preserves the nonzero (NZ) elements with the highest absolute value on each row of each weight matrix. The value of NZ is tuned to provide the highest possible application-level accuracy, considering the user-specified latency budget.

Hybrid Compression and Pruning

To obtain a refinement mechanism that allows us to increase the QoR as a function of time while leveraging the advantages of both aforementioned techniques, we combine them in a hybrid iterative approximation method given by the following:

$$\tilde{\mathbf{y}}_i = \sum_{n=1}^{N_{\text{steps}}} \{\underbrace{\sigma_1^{i(n)}\mathbf{u}_1^{i(n)}(\overbrace{\text{prune}(\mathbf{v}_1^{i(n)}, \text{NZ})}^{\text{pruning}})^\top \tilde{\mathbf{x}}^{(t)}}_{\text{refinementstep}}\}. \tag{1}$$

The iterative nature of the hybrid method involves the refinement of the computed output over a number of iterations, with each refinement step involving the addition of a low-rank approximation of a correction factor (residual) together with its pruning. With this scheme, the final approximate output vector is formed after applying N_{steps} refinement steps. The weight matrices of each LSTM gate are approximated by N_{steps} vector pairs. At the nth refinement iteration, the value $\sigma_1^{i(n)}$ and vectors $\mathbf{u}_1^{i(n)}$ and $\mathbf{v}_1^{i(n)}$ capture the rank-1 approximation of a correction factor. In this manner, at each refinement step, the current $\mathbf{v}_1^{i(n)}$ vector is pruned using our pruning scheme, in order to end up with NZ elements, and then is multiplied with the current augmented input vector, resulting to an nonfull rank-1 approximation. By utilizing the approximation *residual* at each time step ($\mathbf{R}_i^{(n)} = \mathbf{W}_i\text{-}\widetilde{\mathbf{W}}_i^{(n-1)}$) to extract an SVD-based rank-1 correction factor for the progressive refinement of the augmented weight–matrix approximation, the error due to both the SVD and the pruning are considered in contrast to the case of progressively applying higher-rank approximations of the original weight matrix, minimizing in this way the information loss.[21] Hence, the workload of each gate is reduced to $N_{\text{steps}}(2R + 2NZ + 1)$ operations.

Therefore, in the hybrid method, different combinations of level of pruning and number of refinement steps correspond to different candidate designs with different computation cost and QoR. In this respect, the number of NZs and the number of refinements (N_{steps}) form tunable parameters that are optimized by the proposed methodology to meet the time constraints and QoR requirements of the target application.

DOMAIN-SPECIFIC ARCHITECTURE FOR LSTMS

The philosophy behind the proposed architecture is to overcome the limitations of programmable processors by introducing a set of strategies that exploit the properties of LSTMs. These include the adoption of *dataflow processing* to alleviate the overheads of conventional computing platforms, the exploitation of both the *intergate* and *intragate parallelism* of LSTMs to boost performance and the compile-time *tunable scaling* of the architecture to match the available resources and the response-time demands of the target application.

Dataflow Processing

In contrast with the control-flow paradigm of general-purpose computers where individual instructions are scheduled for execution, we adopt a data-driven dataflow architecture. In this scheme, the availability of input samples triggers the LSTM processing to be performed on them without the need for explicit control and synchronization between computation units. From a hardware perspective, this approach allows us to remove any generic instruction-handling hardware logic and repurpose the resources of the

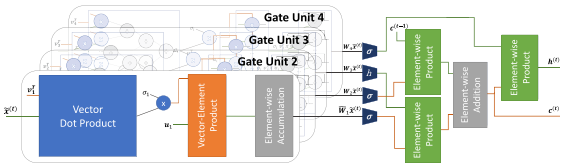

Figure 4. Custom LSTM accelerator architecture (see the paper by Rizakis *et al.*[21]).

FPGA chip specifically for LSTMs. In this way, the architecture avoids the time, resource, and power overhead of off-the-shelf platforms and boosts the attainable performance by dedicating more hardware resources for computation.

Intergate and Intragate Parallelism

Figure 4 shows the block diagram of the architecture. At its core, the architecture is organized as a pipeline of five coarse stages, including four parallel *hardware gate units*, a set of nonlinear operators, and a number of multiplier and adder arrays. Starting on the left-hand side, the four parallel hardware gate units are the heart of the architecture. The proposed design exploits the coarse-grained, intergate parallelism by mapping each LSTM gate to a dedicated hardware gate unit, with all units operating concurrently. At each LSTM time-step t, a hardware gate unit computes its output by performing N_{steps} refinement iterations. As a first step, the current input vector is sent from the off-chip memory into an on-chip buffer, as it will be reused across all refinement iterations. In the nth iteration, the singular vectors $\mathbf{u}_1^{i(n)}$ and $\mathbf{v}_1^{i(n)}$ for the ith gate are streamed in from the off-chip memory in a tiled manner with tile sizes T_r and T_c, respectively, along with the singular values $\sigma_1^{i(n)}$.

Internally, each hardware gate unit contains three processing modules: a *dot-product* unit, a *multiplier* array, and an *adder* array (see Figure 4). By mapping the operations of a gate to parallel circuits, the architecture capitalizes on the fine-grained, intragate parallelism of these operations to obtain performance gains. After the hardware gate units have applied all the necessary refinements, the outputs of the four gates are passed through nonlinear operators. Consequently, the produced outputs are processed using the multiplier and adder arrays to produce the new cell state $\mathbf{c}^{(t)}$ and output vector $\mathbf{h}^{(t)}$.

Configurable Scaling

At compile time, the configuration of the architecture is controlled by means of two parameters: $T_r \in [1, R]$ and $T_c \in [1, \text{NZ}]$. T_r controls the size of all the arrays, while T_c determines the number of multiply–add operators in each hardware gate unit. Different values of T_r and T_c correspond to different scaling of the architecture and provide a tunable performance-resource cost tradeoff, which is used to customize the design based on the available resources and the response-time requirements.

NAVIGATING THE DESIGN SPACE

Given an LSTM and a target FPGA, the parameters of the overall methodology comprise the approximation method parameters NZ and N_{steps}, and the architectural parameters T_r and T_c. Different combinations of these parameters correspond to alternative designs. For a fixed-time constraint, each candidate design is characterized by its 1) QoR, 2) performance in terms of processing speed, and 3) resource consumption. To explore this space, we need to study the effect of the architectural parameters on the performance of the hardware implementation as well as the impact of the approximations on the QoR of the target application.

Performance: Following the Roofline

To investigate the attainable performance of different architectural configurations, we adopt the roofline model[22] from the HP computing community. The roofline model is a visual model for identifying the causes of performance bottlenecks in computing systems. Based on this model, the performance of a design can be limited by either the peak processing rate of the target platform or by the maximum bandwidth that the external memory subsystem can support.

In this context, we built a roofline model for the proposed architecture, which can be used to explore the performance of a large space of alternative designs, without the need for long simulations.[21] The various candidate designs differ in terms of number of refinement iterations (N_{steps}), level of pruning (NZ), and scaling of the hardware (T_r, T_c). Given the pruning level NZ, the number of refinements N_{steps} and a pair of architectural

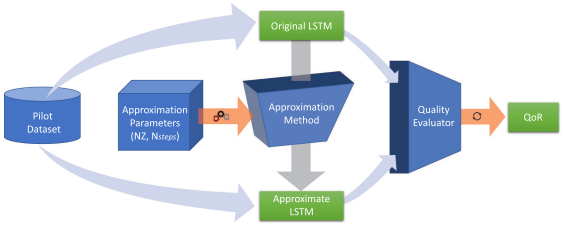

Figure 5. Process of capturing the approximation-QoR tradeoff.

parameters (T_r, T_c), the *attainable performance* of the architecture (in GOp/s) can be modeled as the operation number to the latency ratio for each LSTM inference.

As the weights of an LSTM do not typically fit in the on-chip memory of an FPGA, we model *operational intensity*, also referred to as the computation-to-communication ratio (CTC), as multiplication and addition operations per byte of weights accessed from the external memory (GOp/byte). Utilizing the abovementioned scheme, a design space exploration is conducted to obtain the highest performing set of parameters for both the approximation method and the architecture given the target platform.

Level of Approximation Versus Quality of Result

Typically, approximation methods exploit the error tolerance of an application together with the perceptual limitations of humans to tradeoff QoR with faster processing. Nevertheless, emerging mission-critical systems, such as driverless cars, place safety and robustness at the forefront and, hence, require guarantees with respect to both QoR and processing latency.[23] To make principled design decisions that meet the requirements of such applications, it is essential to capture the relationship between the application-level QoR and the level of approximation, and use it to tune the computing system based on the application specifications.

To achieve that, we follow the methodology shown in Figure 5. Initially, the error induced by the proposed LSTM approximations on an application is experimentally measured as a function of the targeted iterations. Given a (NZ, N_{steps}) pair, the approximate LSTM is generated from the original LSTM (top to bottom of Figure 5). Next, we run the target application end-to-end over a pilot dataset using both the original and the approximate LSTM. By treating the final output of the original model as the ground truth, an application-specific metric is employed to assess the QoR of the approximate LSTM (left to right of Figure 5). The quality metric measures the similarity between the original and the approximate result and must have a suitable form based on the target domain, such as the relative error between the approximate and reference result or the Kullback–Leibler (KL) divergence that captures the distance between the respective probability distributions. Overall, by varying the values of (NZ, N_{steps}) and observing the associated QoR, the relationship between the level of approximation and the QoR is captured.

CASE STUDY: AUTONOMOUS DRIVING

Overview

One of the emerging AI-driven applications with the highest potential for societal impact is autonomous driving. Although initial efforts began in the late 1980s,[24] the field of autonomous driving has experienced significant progress in the past decade, owing to efforts from both the industrial and academic communities. The main enablers of the emerging technologies being developed are 1) the advancement of *deep learning algorithms* allowing the extraction of powerful representations, 2) the availability of *real-world training data* provided by open-source datasets,[6,25] and 3) the development of *embedded processing platforms* with enhanced computational capabilities that allow the deployment of computationally expensive software on-board the vehicle,[9,26] satisfying the imposed low-latency and safety constraints.

Vision-based driving assistance and autonomy,[19,27,28] is gaining attention due to the low-cost, widely available cameras that can be used independently or accompany other sensors for environmental perception. With such sensors providing a stream of measurements, recurrent models such as LSTMs form a promising learning paradigm that can extract and exploit temporal information from the incoming data to develop a smooth and consistent driving policy, in place of the independent perinput predictions provided by classical deep learning models that exploit solely spatial information.[29]

Self-driving car systems consist of a large set of computationally demanding tasks, including sensor preprocessing, localization, mapping, path planning and obstacle avoidance, and control and emergency handling.[30] Hard low-latency constraints[2] between perception and action impose the need for HP implementations that guarantee the extraction of highly accurate approximations on each individual component, to meet the real-time performance requirements of the overall system with insignificant effect on accuracy. As an example, a coarse but in-time estimation of the vehicle's obstacle avoidance system to take a "sharp" left turn and avoid a collision is preferred to a delayed but rather accurate regression of an exact steering angle response to a visual input.

Target Application

The driving model presented in the paper by Xu *et al.*[31] trained on the Berkeley DeepDrive Video dataset, a large-scale crowdsourced driving video dataset forming an early version of the BDD100K Dataset,[6] is examined as a case study for evaluating the proposed framework. Similar to the work of Kim *et al.*[32] on the vision-based autonomous mobile robot navigation, Xu *et al.*[31] also exploit the end-to-end learning paradigm. Input frames for each video are first processed by a fully convolutional network (FCN) to encode the spatial features, which are then fed to a trained LSTM model that predicts the probability distribution across a discrete set of feasible future actions for the vehicle (go forward, stop, turn left, turn right) taking advantage of the temporal motion information from previous representations. The LSTM input is enhanced with the linear and angular velocities of the vehicle predicted by the system from the previous frame. This FCN-LSTM architecture is a novel version of long-term recurrent convolutional networks, typically consisting of a convolutional neural network (CNN) feeding its output to an LSTM, combining the current state of the art in visual and sequence learning to extract spatio-temporal information for input streams.

Evaluation

In this section, we discuss the extensive experimental evaluation conducted to showcase the effectiveness of the proposed approach in the target application of this case study. The proposed progressive inference methodology is initially compared with an FPGA-based baseline for LSTM inference to demonstrate its efficacy on making informed predictions under computation time constraints (see the "Comparison With FPGA Baseline" section). Then, a comparison of the proposed methodology with faithful off-the-self LSTM implementations targeting other computing platforms (CPU and GPU) considering latency, power consumption, and performance efficiency is discussed (see the "Comparison With CPU and GPU Baselines" section).

Experimental Setup

We focus on the LSTM of the examined driving model for this case study, each gate of which forms an $R \times C$ augmented weight matrix, with $R = 64$ and $C = 8\,320$. We evaluate the method on part of the validation set of the dataset that was used to train the model, by cropping a segment of 100 consequent frames from over 1 800 real videos of diverse driving scenarios. To generate action probability distributions that will act as ground truth for the evaluation of the proposed approximation method, we follow the process in "Level of Approximation Versus Quality of Result" section and execute the original driving model end-to-end over the validation set using TensorFlow. As a metric of the effect of low-rank approximation and pruning on the QoR, we employ KL divergence—a commonly used metric of dissimilarity between distributions—between the reference and predicted probability distribution.

In our experiments, we target the Xilinx's ZC706 board mounting the Zynq 7045 chip. This platform is an industry standard for FPGA-based embedded systems and is based on the Zynq-7000 System-on-Chip, which integrates a dual-core Arm CPU alongside an FPGA fabric on the same chip. For the data format, we use single-precision floating-point representation to comply with the typical precision requirements of LSTMs as used by the deep learning community. All hardware designs are synthesized with Vivado HLS and Vivado Design Suite (v2017.1) achieving a clock frequency of 100 MHz.

The core LSTM workload of the proposed approximate computing scheme (dot-product

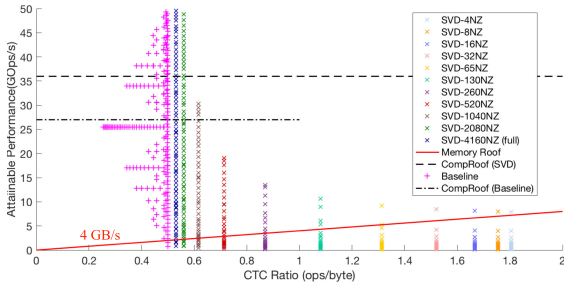

Figure 6. Roofline model analysis for the baseline architecture and various configurations of the proposed method.

followed by a vector scaling by a constant), as well as the baseline LSTM implementation in the "Comparison With FPGA Baseline" section (matrix–vector multiplication), is implemented on the FPGA. At the same time, the CPU coordinates the operation of the system by 1) scheduling the computations between different tiles from all four LSTM gates and mapping them to the available processing elements of the custom hardware accelerator, and 2) setting up the communication interface between the accelerator and the external memory. To this end, we use the four AXI-based HP ports that are available on the target device. For each port, we configure it with a 64-bit width and instantiate a dedicated DMA engine, clocked at 150 MHz, to independently perform the memory transfers. Overall, our memory interface subsystem yields a measured bandwidth of around 4 GB/s, as shown on the slope of the roofline model in Figure 6, with the CPU initializing the DMA engines state prior to execution.

In the comparison of the proposed methodology with a CPU- and GPU-based LSTM implementation (see the "Comparison With CPU and GPU Baselines" section), we used PyTorch (v1.1.0) with CUDA 10, to develop a faithful LSTM baseline and deploy it on the widely used NVIDIA Jetson AGX Xavier board (which was also presented at the 2017 Consumer Electronics Show,[18]) featuring an 8-core Arm 64-bit CPU along with a 512-core Volta GPU. Average performance and power are calculated after completing 1 000 iterations of each experiment across all platforms. The idle power is subtracted from all measurements, leading to a comparison of the actual power consumed by the benchmark execution (including the memory accesses).

Comparison With FPGA Baseline A hardware architecture implementing a faithful mapping of the original LSTM model described in "Learning Long-Term Patterns With LSTMS" section is developed to act as a baseline for the evaluation of the proposed system. This baseline architecture consists of four gate units with a total of 2.1M parameters, implemented in parallel hardware that performs the matrix–vector multiplication operations of LSTM gates (see Figure 3) in a blocked manner. The computational workload for the kernel of each gate is $2RC$ operations. Parametrization with respect to the tiling along the rows (T_r) and columns (T_c) of the weight matrices is applied and roofline modeling is used to obtain the highest performing configuration (T_r, T_c), similarly to the proposed system's architecture (see Figure 6). As Figure 6 demonstrates, the designs are mainly memory-bound, and as a result a small portion of the FPGA resources are utilized. To obtain the application-level QoR of the baseline design under time-constrained scenarios, the KL divergence between the intermediate LSTM output at each tile step of T_r and the predictions of the reference model is examined and illustrated by the black line of Figure 7(b).

The gains of the proposed methodology compared to the baseline design under computation-time constraints are investigated by exploring the design space, defined by (NZ, T_r, T_c), in terms of 1) performance (see Figure 6) and 2) the relationship between error (described by the KL-divergence between the approximate prediction and ground truth) and computation time [see Figure 7(b)]. Figure 7(a) also depicts the relationship between error and computation step for numerous configurations of the proposed system. As illustrated, the QoR of a configuration is inversely proportional to its level of sparsity. Dense configurations, such as those with 50% NZ elements or more, tend to converge to negligible divergence values (below 10^{-6}) in less than 15 computation steps, in contrast with sparser configurations that require more than 75 computations steps to converge to the same divergence level ($\sim$2% NZ elements) or converge to higher divergence values (as in the case of 0.4% NZ elements). Additionally, Figure 8 presents probability distribution instance samples of numerous

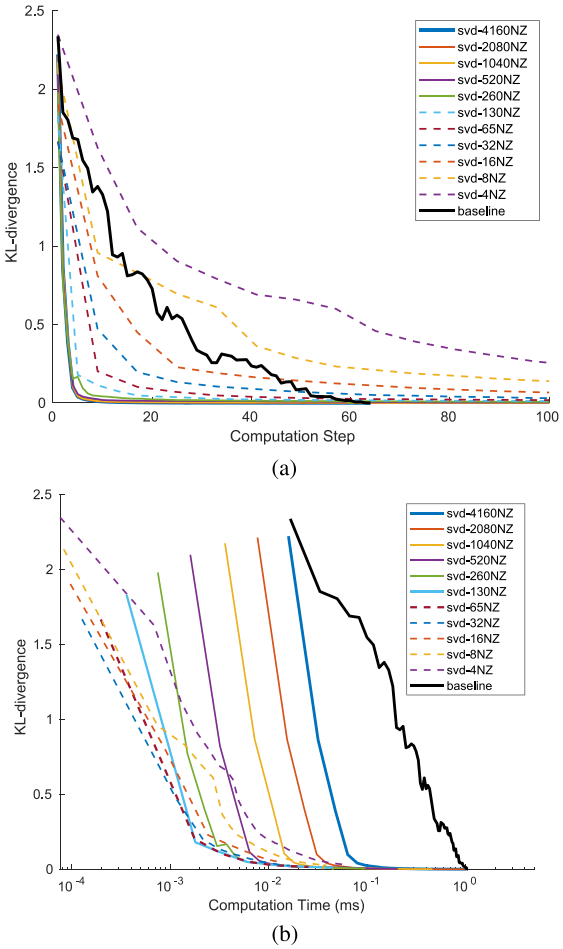

Figure 7. KL-divergence between approximate prediction and reference model output (lower-is-better) as a function of (a) computation step and (b) computation time.

progressive refinement steps for a representative input frame along with their corresponding KL-divergence values. It can be seen that the proposed approach converges to "meaningful results" (application-wise) in much smaller number of computation steps, by exploiting the inherent redundancy of the LSTM model.

As shown in Figure 7(b), since computation time per computation step is also inversely proportional to the level of sparsity of a given configuration, some sparse configurations demonstrate superior accuracy than other denser settings under the same latency constraint. This behavior, however, is not monotonic due to extremely dense configurations requiring a larger number of computation steps to converge. Therefore, the selection of the appropriate level of sparsity is dependent on the latency constraint imposed by the application-level needs. Overall, we notice that the proposed methodology achieves a speed-up of 198× on average (76× geo. mean) across different QoR levels compared to the baseline approach. In particular, when only negligible KL-divergence is allowed between the approximate and reference prediction, the proposed system achieves 2.93× faster inference by exploiting the LSTM model's inherent redundancy. Furthermore, the proposed method demonstrates up to 415× lower inference time to achieve an intermediate QoR prediction exploiting the computation time-accuracy tradeoff. Figure 9 illustrates two representative intermediate probability distributions extracted by an instance of the proposed approach and the baseline, both satisfying the same latency constraint. To obtain these outputs, both methods were fed with the same input, and while calculating their predictions their computation was cut short as the available time budget was hit. The illustrated intermediate output distributions indicate that the proposed approach makes a more informed prediction, significantly closer to the ground truth compared to the baseline. This property is particularly useful in scenarios where tight real-time requirements impose hard latency constraints on the available computation time budget for inference.

Comparison With CPU and GPU Baselines

Targeting the efficient deployment on the embedded space, deep-learning models should abide in a low-power envelope. Power efficiency becomes increasingly prominent in the case of autonomous systems[19] that rely on self-contained power supply resources, and especially in self-driving cars that are emerging alongside with the rise of the electric vehicle era. Power-constrained applications are primarily concerned about 1) the absolute power consumption (watts) and 2) the performance efficiency (performance-per-watt).

In this respect, we also compare multiple instances of the proposed methodology and its underlying FPGA-based hardware implementation with highly optimized off-the-shelf CPU- and GPU-based traditional implementations of LSTM inference, commonly used by the deep learning community, in terms of raw performance, absolute power consumption, and performance efficiency.

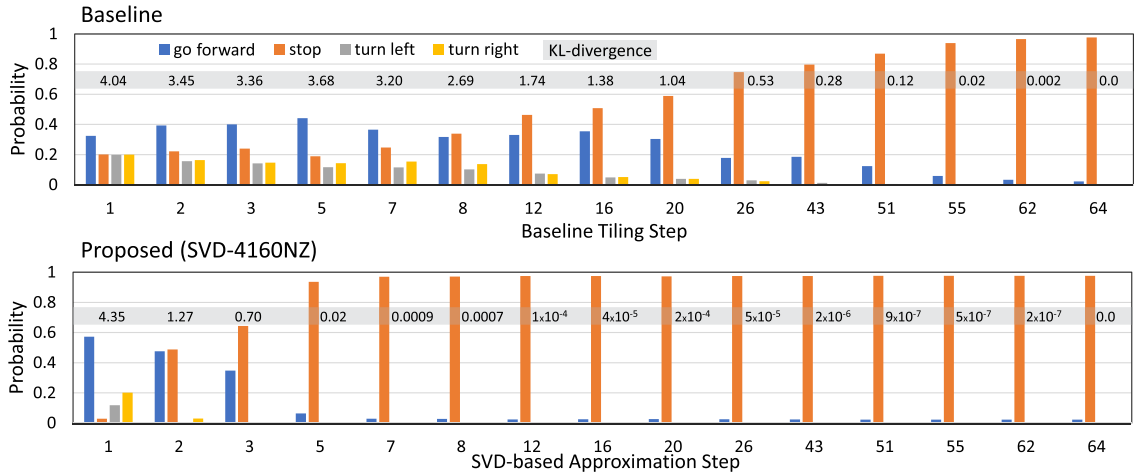

Figure 8. Intermediate prediction instances obtained by the progressive inference baseline (achieving 16.6^{-2} ms/step) and a dense instance of the proposed SVD-based approach (achieving 15.9^{-2} ms/step) on the same data sample, as a function of computation steps. KL-divergence values with respect to the final result are also shown (grey row).

Although raw performance and power consumption are also reported, the most equitable metric for cross-platform comparison is power efficiency, as it effectively normalizes the results with respect to the available computational resources of each target platform.

Table 1 summarizes the results of this comparison. The employed FPGA baseline achieves a 2.28× speed-up compared to the CPU implementation, while suffering a 0.75× slow-down with respect to the GPU, in terms of absolute latency. However, when power consumption is also considered, these results are translated into a 4.31× and 1.96× improvement on performance efficiency compared to the CPU and GPU baseline, respectively. These demonstrated gains in power efficiency achieved by the use of a custom FPGA-based solution render FPGAs as the cardinal platform for LSTM deployment in many power-constrained applications, especially in the embedded space of autonomous systems.

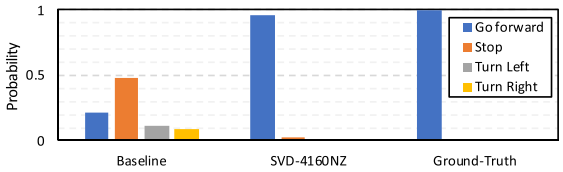

Figure 9. Intermediate prediction instances obtained by the baseline and the proposed approach with NZ = 4 160 on the same data sample, under the same latency constraint ($t = 10^{-1}$ ms).

Multiple instances of the proposed approximate computing scheme are also listed in Table 1. It can be seen that by utilizing solely the proposed computation-restructuring methodology, a speed-up of 6.7×, 2.2×, and 2.93× is achieved in the latency required to yield (almost) identical outputs (KL-divergence ≤ 0.001) with the reference design, compared to the CPU, GPU, and FPGA baselines accordingly, also translated into an improvement of 12.46×, 5.67×, and 2.89× in performance efficiency. These significant gains arise by the proposed methodology exploiting the inherent redundancy of LSTM models in order to maximize the achievable accuracy at every stage of the computation. By performing the most information-carrying computations first, the workload (and computation time) required to reach similar accuracy with the baseline is effectively reduced.

By relaxing the error tolerance into slightly higher KL-divergence values (≤ 0.1), the proposed hybrid compression-and-pruning methodology provides informed approximations of the inference outputs, while demonstrating remarkable performance gains of up to 724×, 238×, and 317× in latency (1 161×, 529×, and 269× improved performance efficiency) compared to the same CPU, GPU, and FPGA baselines, respectively. These gains are amplified remarkably by further relaxing the error tolerance into higher KL-divergence (≤ 1.0), which, however, still yield "meaningful" results.

Table 1. Comparison with other computing platforms.

Platform	Benchmark	Latency	Power	Perf. Efficiency
CPU	Baseline	2.4266 ms	5.13 W	0.342 GOp/s/W
GPU	Baseline	0.7974 ms	7.11 W	0.752 GOp/s/W
FPGA	Baseline	1.0620 ms	2.72 W	1.476 GOp/s/W
FPGA	†* (KL $\leq$ 0.001)	0.36190 ms	2.76 W	4.267 GOp/s/W
FPGA	†** (KL $\leq$ 0.01)	0.03924 ms	3.20 W	33.943 GOp/s/W
FPGA	†** (KL $\leq$ 0.1)	0.00335 ms	3.20 W	397.713 GOp/s/W
FPGA	†*** (KL $\leq$ 1.0)	0.00072 ms	3.41 W	1 735.992 GOp/s/W

† This work, *SVD-4160NZ (no pruning), **SVD-130NZ, ***SVD-32NZ

Since all the configurations of the proposed (and baseline) approach are memory bounded (see Figure 6), the attainable parallelism and performance (GOp/s) of the underlying hardware architecture increase proportionally to the selected sparsity level. As it can be noticed in Table 1, the absolute power consumption of sparser configurations increases. This is due to the fact that since sparser configurations lead to the higher CTC ratio (see Figure 6), more parallel processing can be exploited in this case, by instantiating more on-chip computational resources on the FPGA. Consequently, although processing becomes faster, the absolute power of the accelerator also increases as a result of the on-chip power consumption.

Exploiting the computation time-accuracy tradeoff, the proposed progressive inference methodology can provide high-quality approximations of the final result at early stages of the computation, which are iteratively refined as more time budget becomes available. This scheme is particularly useful for systems with hard computation-time constraints (e.g., in mission-critical real-time applications), enabling them to maximize the attainable QoR within the given latency envelope. Furthermore, the introduced highly parametrized custom hardware architecture for the proposed methodology demonstrates remarkable power efficiency by exploiting the enhanced customization capabilities and flexibility of FPGAs. In this manner, highly optimized hardware mappings of different configuration instances of the proposed approximation scheme are generated, while being tailored to the needs of the target application.

RELATED WORK

The rapid advances in deep learning have led to significant research effort invested in optimizing the execution of deep neural networks. The majority of existing work has focused on compute-intensive CNNs for computer vision tasks. The substantial redundancy of modern deep CNNs together with the inherent parallelism and data-reuse of CNN workloads have made them amenable to various compression and acceleration techniques. At the algorithmic level, methods such as knowledge distillation,[33] efficient convolutions,[34] and neural architecture search[35] have been successfully applied to significantly compress CNN models by leveraging their high inherent redundancy. At the same time, techniques such as reduced precision[15,36] and custom hardware designs[37] have been employed for acceleration by exploiting the high levels of parallelism and data reuse of CNNs. Nevertheless, with memory-bound LSTMs having substantially different computational patterns, the CNN-centric methods and accelerator designs either provide minimal gains or are not directly applicable to LSTMs.[38,39]

Closer to the progressive inference philosophy of our approach lie CNNs that employ early-exit classifiers. CNNs with early exits[40–42] provide a run-time accuracy-latency tradeoff and are able to produce an increasingly refined output as a function of time, which casts them suitable for time-constrained inference scenarios. However, as the early-exit classifiers have to be trained, access to the training set is necessary and complex hyperparameter tuning is required.[41,42] Furthermore, although early exiting has been applied to CNN-based classifiers with promising results, this mechanism is not directly applicable to the substantially different topology of LSTMs. Alternatively, our method enables us to perform progressive inference using LSTM models without the need to access the training set and the excessive time overhead of tuning the associated hyperparameters.

With a focus on LSTM workloads, several works have proposed optimizations for executing LSTMs on conventional programmable platforms such as CPUs[43] and GPUs.[38,44,45] By employing tailor-made caching and data-locality strategies, this line of work has demonstrated significant performance gains and has approached the

performance limits of commodity programmable hardware architectures. To push further the attainable performance of LSTMs, another line of work has exploited the characteristics of FPGAs to propose custom accelerator designs. Based on the stage where optimizations are applied, FPGA-based LSTM designs can be categorized into 1) posttraining with fine-tuning, 2) training-stage, and 3) run-time methods.

Posttraining Methods With Fine-Tuning By putting emphasis on minimizing the effect of memory boundedness of LSTM workloads, ESE[10] proposes to sparsify LSTMs via a pruning scheme and map it on an FPGA-based accelerator tailored for sparse workloads. Given a pretrained model, its weights are pruned in an iterative manner using a load-balance-aware strategy that aims to sustain the utilization of the accelerator high. Furthermore, to avoid excessive accuracy drop, at each iteration, the unpruned weights are fine-tuned using the training set. To overcome the inefficiencies of CPUs and GPUs when executing the sparse, pruned model, ESE exploits the customizability of FPGAs to propose an accelerator optimized for sparse computations. As a result, the load-balance-aware pruning leads to 6.2× faster execution over dense LSTMs on ESE's accelerator. To further improve the load balancing, Park *et al.*[46] proposed an alternative encoding format for storing sparse matrices and managed to achieve higher sustained utilization of the PEs on the same accelerator.

Overall, despite the fact that the pruning method used by both ESE and Part *et al.*[46] is applied *posttraining* on pretrained LSTMs, access to the training set is required in order to iteratively prune and fine-tune the model's weights and, thus, not significantly degrade the accuracy. In contrast, our method is also applied posttraining on pretrained models, but it does not require access to the dataset and, hence, is suitable for privacy-aware cases.

Training-Stage Methods By modifying the model design process, Wang *et al.*[11] proposed a compression technique that modifies the LSTM model *before* the training stage. By applying a circulant structure to the matrices within each LSTM gate, this approach allows the same weights to be shared across several neurons and substantially reduces the model size and storage requirements. Further parametrizing this technique, the E-RNN[47] framework introduces a blocking version of circulant matrices and treats the block size as a tunable parameter to balance the processing speed and accuracy. The block-circulant matrix operations were executed in the frequency domain to leverage the computational efficiency of FFT. At the hardware level, to bypass the limitations of conventional platforms when executing irregular computations, E-RNN proposed a highly parametrized custom hardware architecture mapped on the flexible FPGA fabrics, leading to a 7.7× speed-up over ESE.[10]

In contrast with our posttraining approach, both of these methods are applied at the LSTM model level and intervene substantially with the LSTM model design and training. Nevertheless, since the SVD-based decomposition of our work is applicable to circulant matrices, our scheme is orthogonal to these works and can be applied in a complementary manner to yield further performance improvements.

Run-Time Methods This class of methods exploits techniques to dynamically skip unnecessary computations during the execution of an LSTM. In this context, DeltaRNN[48] employs a strategy to dynamically avoid computations based on the estimated impact on the output of the network. The skipping criterion is based on the degree of change of each input activation. To effectively implement this technique without significantly dropping the accuracy, the target LSTM has to be trained using the Delta Network scheme.[49] From a hardware perspective, to overcome the inefficiency of GPUs due to the conditional execution strategy, the DeltaRNN-trained LSTM is mapped on a custom accelerator design, which exploits the reconfigurability of FPGAs to efficiently perform the dynamic computations. Nevertheless, despite the run-time computation-skipping, DeltaRNN requires the target model to be trained using the Delta Network algorithm and, hence, is limited to settings where the training set is available, while requiring substantial modification of the training scheme and tuning of the hyperparameters. In contrast to this, our method avoids the time overhead and engineering effort

of training and parameter tuning and can be directly applied to pretrained LSTMs.

Among the existing designs, reduced arithmetic precision schemes have also been used to obtain gains in terms of performance and power efficiency. ESE[10] and E-RNN[47] employ 12-bit fixed-point precision for both weights and activations. However, to avoid the severe degradation of accuracy due to the limited numerical precision, a fine-tuning step is required by means of additional training iterations. Alternatively, DeltaRNN[48] avoids fine-tuning and employs a 16-bit fixed-point representation. Nonetheless, DeltaRNN's quantization is not network-agnostic but hand-tuned to minimize the accuracy losses of the target network. In our work, 32-bit single-precision floating-point format is used to avoid the need for fine-tuning and limit the sources of QoR degradation to our approximate computing techniques. Nevertheless, our method is orthogonal and independent of employed numerical precision and, thus, can be combined with existing quantization schemes to further boost both performance and power efficiency.

CONCLUSION

The deployment of LSTMs in latency-critical applications is a challenging task due to their high computational requirements. In this article, an iterative approximate computing method together with an FPGA-based architecture are introduced combining model pruning with computation restructuring to make approximate but informed LSTM predictions in time-constrained environments. In a self-driving car scenario, the proposed system demonstrates significant improvements in accuracy for every given computation time budget compared to a baseline that follows conventional implementations.

It is noteworthy that the proposed approximation methodology effectively reduces the workload required to achieve a desired QoR for a given model, and therefore, it can be decoupled from the proposed custom-hardware implementation and adapted for deployment on other computing platforms with variable performance gains. Future work encompasses an investigation of ways to adapt the proposed methodology for efficient deployment on other platforms.

ACKNOWLEDGMENTS

This work was supported in part by the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/S030069/1) and in part by the Engineering and Physical Sciences Research Council under Grant 1507723.

REFERENCES

1. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
2. P. Corcoran and S. K. Datta, "Mobile-Edge computing and the internet of things for consumers: Extending cloud computing and services to the edge of the network," *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 73–74, Oct. 2016.
3. Q. Wang, Y. Liu, J. Liu, Y. Gu, and S. Kamijo, "Critical areas detection and vehicle speed estimation system towards intersection-related driving behavior analysis," in *Proc. IEEE Int. Conf. Consum. Electron.*, Jan. 2018.
4. N. Kumar, D. Puthal, T. Theocharides, and S. P. Mohanty, "Unmanned aerial vehicles in consumer applications: New applications in current and future smart environments," *IEEE Consum. Electron. Mag.*, vol. 8, no. 3, pp. 66–67, May 2019.
5. D. V. McGehee, E. N. Mazzae, and G. S. Baldwin, "Driver reaction time in crash avoidance research: Validation of a driving simulator study on a test track," in *Proc. Human Factors Ergonom. Soc. Annu. Meeting*, vol. 44, no. 20, pp. 3–320, 2000.
6. F. Yu *et al.*, "BDD100K: A diverse driving video database with scalable annotation tooling," 2018, *arXiv:1805.04687*.
7. Y. Guan, Z. Yuan, G. Sun, and J. Cong, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *Proc. 22nd Asia South Pacific Des. Autom. Conf.*, Jan. 2017, pp. 629–634.
8. A. X. M. Chang and E. Culurciello, "Hardware accelerators for recurrent neural networks on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2017.
9. S. Ray, "Safety, security, and reliability: The automotive robustness problem and an architectural solution," in *Proc. IEEE Int. Conf. Consum. Electron.*, Jan. 2019.
10. S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 75–84.

11. Z. Wang, J. Lin, and Z. Wang, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 25, no. 10, pp. 2763–2775, Oct. 2017.
12. X. Zhang *et al.*, "High-performance video content recognition with long-term recurrent convolutional network for FPGA," in *Proc. 27th Int. Conf. Field Programmable Logic Appl.*, Sep. 2017.
13. S. Wang, S. S. Cheung, and H. Sajid, "Visual bubble: Protecting privacy in wearable cameras," *IEEE Consum. Electron. Mag.*, vol. 7, no. 1, pp. 95–105, Jan. 2018.
14. R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.
15. A. Kouris, S. I. Venieris, and C. Bouganis, "CascadeCNN: Pushing the performance limits of quantisation in convolutional neural networks," in *Proc. 28th Int. Conf. Field Programmable Logic Appl.*, Aug. 2018, pp. 155–1557.
16. M. J. Wainwright, M. I. Jordan, and J. C. Duchi, "Privacy aware learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1430–1438.
17. B. Markwalter, "The path to driverless cars [CTA insights]," *IEEE Consum. Electron. Mag.*, vol. 6, no. 2, pp. 125–126, Apr. 2017.
18. F. Pieri, C. Zambelli, A. Nannini, P. Olivo, and S. Saponara, "Is consumer electronics redesigning our cars?: Challenges of integrated technologies for sensing, computing, and storage," *IEEE Consum. Electron. Mag.*, vol. 7, no. 5, pp. 8–17, Sep. 2018.
19. V. K. Kukkala, J. Tunnell, S. Pasricha, and T. Bradley, "Advanced driver-assistance systems: A path toward autonomous vehicles," *IEEE Consum. Electron. Mag.*, vol. 7, no. 5, pp. 18–25, Sep. 2018.
20. M. Denil, B. Shakibi, L. Dinh, and N. De Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.
21. M. Rizakis, S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Approximate FPGA-Based LSTMs under computation time constraints," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2018, pp. 3–15.
22. S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.
23. R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 4745–4753.
24. D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 305–313.
25. A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3354–3361.
26. S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.
27. M. Bojarski *et al.*, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*.
28. C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 2722–2730.
29. L. Chi and Y. Mu, "Deep steering: Learning end-to-end driving model from spatial and temporal visual cues," 2017, *arXiv:1708.03798*.
30. S. Thrun, "Toward robotic cars," *Commun. ACM*, vol. 53, no. 4, pp. 99–106, 2010.
31. H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 3530–3538.
32. Y. Kim, J. Jang, and S. Yun, "End-to-end deep learning for autonomous navigation of mobile robot," in *Proc. IEEE Int. Conf. Consum. Electron.*, Jan. 2018.
33. G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NIPS Deep Learn. Representation Learn. Workshop*, 2015.
34. E. J. Crowley, G. Gray, and A. J. Storkey, "Moonshine: Distilling with cheap convolutions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2888–2898.
35. Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 784–800.
36. K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: hardware-aware automated quantization with mixed precision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 8612–8620.
37. S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," *ACM Comput. Surv.*, vol. 51, no. 3, 2018, Art. no. 56.
38. X. Zhang, C. Xie, J. Wang, W. Zhang, and X. Fu, "Towards memory friendly long-short term memory networks (LSTMs) on mobile GPUs," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2018, pp. 162–174.

39. J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Architecture*, Jun. 2018.
40. S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.
41. G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Representations*, 2018.
42. Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2019, pp. 3301–3310.
43. M. Zhang, S. Rajbhandari, W. Wang, and Y. He, "DeepCPU: Serving RNN-based deep learning models 10x faster," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 951–965.
44. G. Diamos *et al.*, "Persistent RNNs: Stashing recurrent weights on-chip," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 2024–2033.
45. F. Zhu, J. Pool, M. Andersch, J. Appleyard, and F. Xie, "Sparse persistent RNNs: Squeezing large recurrent networks on-chip," in *Proc. Int. Conf. Learn. Representations*, 2018.
46. J. Park, W. Yi, D. Ahn, J. Kung, and J. Kim, "Balancing computation loads and optimizing input vector loading in LSTM accelerators," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, to be published, doi: 10.1109/TCAD.2019.2926482.
47. Z. Li *et al.*, "E-RNN: Design optimization for efficient recurrent neural networks in FPGAs," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, Feb. 2019, pp. 69–80.
48. 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-Programmable Gate Arrays*, 2018, pp. 21–30.
49. D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu, "Delta networks for optimized recurrent network computation," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2584–2593.

Alexandros Kouris is currently working toward the Ph.D. degree with the Electrical and Electronic Engineering Department, Imperial College London, London, U.K. Contact him at a.kouris16@imperial.ac.uk.

Stylianos I. Venieris is currently a Researcher with the Samsung AI Center, Cambridge, U.K. He received the Ph.D. degree from Imperial College London, London, U.K. Contact him at s.venieris@samsung.com.

Michail Rizakis is currently working toward the graduate degree with the Electrical and Electronic Engineering Department, Imperial College London, London, U.K. Contact him at michail.rizakis14@imperial.ac.uk.

Christos-Savvas Bouganis is currently a Reader in Intelligent Digital Systems with the Electrical and Electronic Engineering Department, Imperial College London, London, U.K. He is a Senior Member of IEEE. Contact him at ccb98@ic.ac.uk.