

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

Alexandros Kouris, Stylianos I. Venieris, Michail Rizakis and Christos-Savvas Bouganis
Electrical & Electronic Engineering Department, Imperial College London, UK
 {a.kouris16,stylianos.venieris10,michail.rizakis14,christos-savvas.bouganis}@imperial.ac.uk

Abstract—The need to recognise long-term dependencies in sequential data such as video streams has made LSTMs a prominent AI 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 paper we introduce an approximate computing scheme combining model pruning and computation restructuring to obtain a high-accuracy approximation of the result in early stages of the computation. Our experiments demonstrate that using the proposed methodology, mission-critical systems responsible for autonomous navigation and collision avoidance are able to make informed decisions based on approximate calculations within the available time budget, meeting their specifications on safety and robustness.

I. INTRODUCTION

Recurrent neural networks (RNNs) are a family of machine learning models with the ability to recognise 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 are rapidly becoming an ubiquitous scenario. In this setting, AI agents are equipped with LSTM-based mechanisms of sensing, perceiving and, eventually, acting. 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.

Fig. 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 human driver reaction time ranging between 0.7 and 3 seconds (varying with situation and individual person) [2], 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 of deploying LSTMs follow the behaviour depicted in Fig. 2. Conventional implementations [4], [5] require the whole inference computation

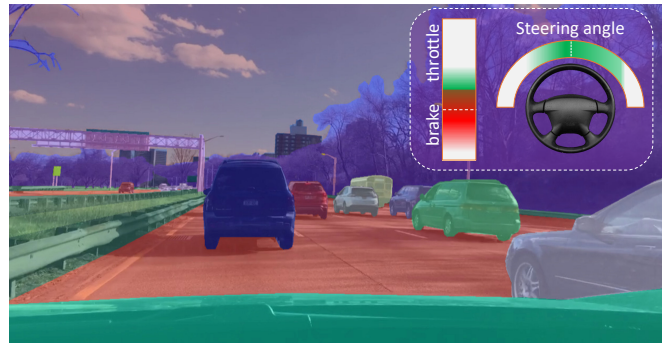


Fig. 1. Throttle/brake and steering angle prediction for autonomous driving with an LSTM model (trained on the dataset of [3]), relying on visual inputs.

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 designs that can provide the best possible estimate of their final output for a given time budget and improve on it as more time budget becomes available. This property would enable the agent to exploit the maximum possible amount of information that is available in the current input and effectively optimise 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. To attack this issue, recent works deviated from general-purpose computing platforms and adopted a *model-hardware co-design* approach for the generation of custom hardware architectures. Each of these works proposes an approximation technique, focusing on model compression [6], quantisation [7] and pruning [8], together with an associated hardware accelerator, 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, 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 [9]–[11].

In this context, we propose a novel methodology for the high-performance deployment of LSTMs in time-constrained applications, which is also complementary to the existing

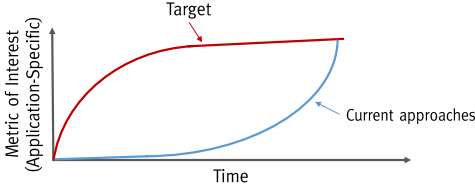


Fig. 2. Conventional and target behaviour of time-constrained AI systems. The y-axis metric reflects the application-level accuracy (higher-is-better).

approaches. The proposed approximate compute scheme is implemented on custom hardware, exploiting the customisation and flexibility of field-programmable gate arrays (FPGAs). The reconfigurable fabric of an FPGA can be customised at the hardware level, allowing the on-chip compute and memory resource allocation to be optimised to match the particular workload and the performance needs of the target application.

The goal is to generate an optimised 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 compute 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 trade-off presented by our approximate scheme that effectively reduces the computational workload of a given LSTM model to meet the desired quality-of-result, compared to a baseline implementation of the same model, while both designs are exploiting the customisation capabilities of an FPGA.

II. LEARNING LONG-TERM PATTERNS WITH LSTMS

LSTMs are specialised 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 behaviour at run time. Fig. 3 depicts the structure of an LSTM. The core element of LSTMs is the cell state \mathbf{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}_x and \mathbf{W}_h , as shown on the first line of Eq. (1). 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:

$$\begin{aligned} \mathbf{g}^{(t)} &= \sigma(\mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{W}_h \mathbf{h}^{(t-1)}) \\ \mathbf{y}^{(t)} &= \mathbf{y}^{(t-1)} \odot \mathbf{g}^{(t)} \end{aligned} \quad (1)$$

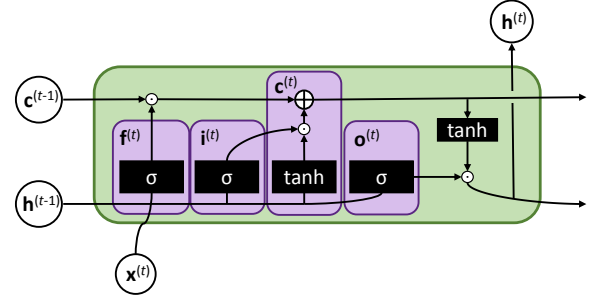


Fig. 3. Structure of an LSTM model.

An LSTM consists of four gates. Starting from the left of the diagram in Fig. 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, while 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.

III. APPROXIMATE COMPUTING FOR LSTMS

At the core of an LSTM's workload lie the linear algebra operation of matrix-vector multiplication, shown on the first line of Eq. (1), which takes place in each of the four gates. Neural networks have been extensively studied to have redundancy in terms of their trained parameters [12]. 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 quality of result (QoR) in exchange for an increase in performance. The proposed approach exploits the statistical redundancy of LSTMs by acting at two levels: (i) approximating weight matrices with a low-rank Singular-Value Decomposition (SVD) and (ii) 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-maximising 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 $\mathbf{W} = [\mathbf{W}_x \mathbf{W}_h] \in \mathbb{R}^{R \times C}$ 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 minimising 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 i -th gate, the approximate weight matrix is given by $\tilde{\mathbf{W}}_i = \sigma_1^i \mathbf{u}_1^i \mathbf{v}_1^{i\top}$. With respect to computational cost, the original matrix vector multiplication $\tilde{\mathbf{W}}_i \tilde{\mathbf{x}}^{(t)}$ 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 NZ elements with the highest absolute value on each row of each weight matrix, leading to NZ non-zero elements per row. This can be expressed as:

$$\tilde{\mathbf{w}}_j^{\text{pruned}} = \text{prune}(\tilde{\mathbf{w}}_j, \text{NZ}), \quad \text{for all rows } j \text{ of matrix } \tilde{\mathbf{W}} \quad (2)$$

where $\tilde{\mathbf{w}}_j$ is the j -th row vector of matrix $\tilde{\mathbf{W}}$ and NZ determines the desired sparsity level of the resulting vector $\tilde{\mathbf{w}}_j^{\text{pruned}}$ and 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 quality of result as a function of time while leveraging the advantages of both aforementioned techniques, we combine them in a hybrid iterative approximation method given by Eq. (3). 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 together with its pruning.

$$\tilde{\mathbf{y}}_i = \sum_{n=1}^{N_{\text{steps}}} \underbrace{\left\{ \sigma_1^{i(n)} \mathbf{u}_1^{i(n)} \underbrace{\left(\text{prune}(\mathbf{v}_1^{i(n)}, \text{NZ})^\top \tilde{\mathbf{x}}^{(t)} \right)}_{\text{refinement step}} \right\}}_{\text{pruning}} \quad (3)$$

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} singular vector pairs. At the n -th refinement iteration, the singular 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 $\mathbf{v}_1^{i(n)}$ singular vector is pruned using our pruning scheme, in order to end up with NZ non-zero elements, and then is multiplied with the current augmented input vector. Hence, the workload for the kernel 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 non-zeros (NZ) and the

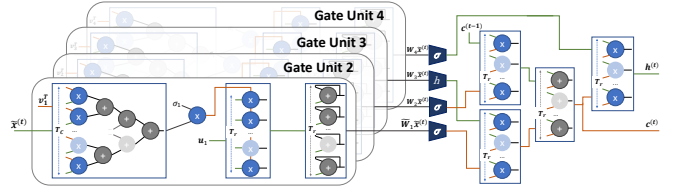


Fig. 4. Custom LSTM hardware architecture (see [13] for more details).

number of refinements (N_{steps}) form tunable parameters that are optimised by the proposed methodology to meet the time constraints and QoR requirements of the target application.

IV. A 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 *inter-gate* and *intra-gate 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 synchronisation between compute units. From a hardware perspective, this approach allows us to remove any generic instruction-handling hardware logic and repurpose the resources of the 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.

Inter- and intra-gate parallelism. Fig. 4 shows the block diagram of the architecture. At its core, the architecture is organised 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, inter-gate 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 n -th iteration, the singular vectors $\mathbf{u}_1^{i(n)}$ and $\mathbf{v}_1^{i(n)}$ for the i -th 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 adder array. By mapping the operations of a gate to parallel circuits, the architecture capitalises on the fine-grained, intra-gate 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. After the nonlinearities stage, 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 trade-off which is used to customise the design based on the available resources and the response-time requirements.

V. 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 characterised by its: 1) quality of result (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.

A. Performance: Following the Roofline

To investigate the attainable performance of different architectural configurations, we adopt the roofline model [14] from the high-performance computing (HPC) 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 [13]. 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 parameters (T_r, T_c), the attainable performance of the LSTM architecture in GOP/s can be modelled as:

$$\text{Performance} = \frac{\text{ops per LSTM inference}(\text{NZ}, N_{\text{steps}})}{\text{latency per sample}(\text{NZ}, N_{\text{steps}}, T_r, T_c)} \quad (4)$$

As the weights of an LSTM do not typically fit in the on-chip memory of an FPGA, we define operational intensity, also referred to as computation-to-communication ratio (CTC), as multiplication and addition operations per byte of weights accessed from the external memory and calculate it as:

$$\text{CTC} = \frac{\text{ops per LSTM inference}(\text{NZ}, N_{\text{steps}})}{\text{bytes accessed}(\text{NZ}, N_{\text{steps}})} \quad (5)$$

where the external memory transfers include the singular vectors and the singular value for each iteration of each gate along with the write-back of the new cell state and the output vector. Utilising the above scheme, a design space exploration is conducted to obtain the highest performing set of parameters for both the approximation method and the architecture.

B. Level of Approximation vs. Quality of Result

Typically, approximation methods exploit the error tolerance of an application together with the perceptual limitations of humans to trade off quality of result (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 [15]. To make principled design decisions that meet the requirements of such applications, it is essential to capture the relationship between application-level QoR and level of approximation and use it to tune the computing system based on the application specifications.

To achieve that, we experimentally measure the error induced by the proposed LSTM approximations on an application as a function of the targeted iterations. Given a (NZ, N_{steps}) pair, the approximate LSTM is generated from the original LSTM. 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. 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.

VI. CASE STUDY: AUTONOMOUS DRIVING

One of the emerging AI-driven applications with the highest potential for societal impact is autonomous driving. Although initial efforts begun in the late 1980s [16], 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: (i) the advancement of deep learning algorithms allowing the extraction of powerful representations, (ii) the availability of real-world training data provided by open-source datasets [3], [17] and (iii) the development of embedded processing platforms with enhanced compute capabilities that allow the deployment of computationally expensive software on-board the vehicle [18], satisfying the low-latency and safety constraints that are imposed.

Vision-based driving assistance and autonomy [19], [20] 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 per-input predictions provided by classical deep learning models that exploit solely spatial information [21].

Self-driving car systems consist of a large set of computationally demanding tasks, including sensor preprocessing, localisation, mapping, path planning and obstacle avoidance,

control and emergency handling [22]. Hard low-latency constraints between perception and action impose the need for high-performance 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 [23], learned from a large-scale crowdsourced driving video dataset (an early version of the BDD100K Dataset [3]), is examined as a case study for evaluating the proposed framework. 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 (LRCNs), typically consisting of a convolutional neural network 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.

Experimental Setup. We focus on the LSTM of this architecture, each gate of which forms an $R \times C$ augmented weight matrix, with $R = 64$ and $C = 8320$. 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 1800 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 of Sec. V-B 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 Kullback-Leibler (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 with a clock frequency of 100 MHz. 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. The core LSTM workload of both the baseline implementation (matrix-vector multiplication) and the proposed approximate computing scheme (dot-product followed by a vector scaling by a constant) is implemented on the FPGA, while the CPU coordinates the operation of the systems by scheduling computations and memory transfers.

Baseline. A hardware architecture implementing a faithful

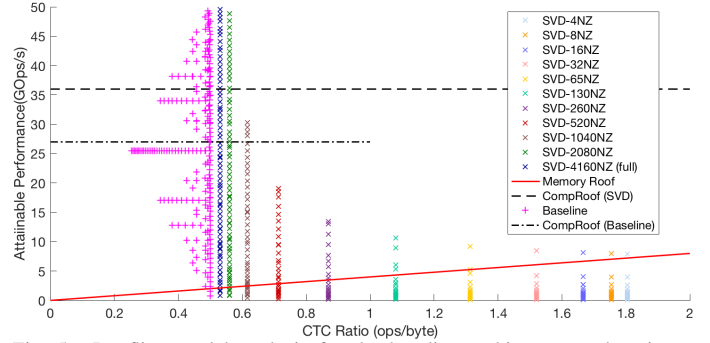


Fig. 5. Roofline model analysis for the baseline architecture and various configurations of the proposed method.

mapping of the original LSTM model described in Sec. II 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 Eq. 1 in a blocked manner. The computational workload for the kernel of each gate is $2RC$ operations. Parametrisation with respect to the tiling along the rows (T_r) and columns (T_c) of the weight matrices is applied and roofline modelling is used to obtain the highest performing configuration (T_r, T_c), similarly to the proposed system’s architecture (Fig. 5). The maximum platform-supported attainable performance was obtained for $T_r = 1$ and $T_c = 4$. As Fig. 5 demonstrates, the designs are mainly memory-bound and as a result a small portion of the FPGA resources are utilised. 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 Fig. 6b.

Comparison. In this section, 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 (i) performance (Fig. 5) and (ii) the relationship between error (described by the KL-divergence between the approximate prediction and ground truth) and computation time (Fig. 6b). Fig. 6a 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% non-zero elements or more, tend to converge more quickly 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 ($\sim 2\%$ non-zero elements) or converge to higher divergence values (as in the case of 0.4% non-zero elements). Additionally, Fig. 7a presents sample instances of probability distributions corresponding to progressive refinement steps for a representative input frame, highlighting the improved QoR as a function of time budget.

As shown in Fig. 6b, since computation time 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

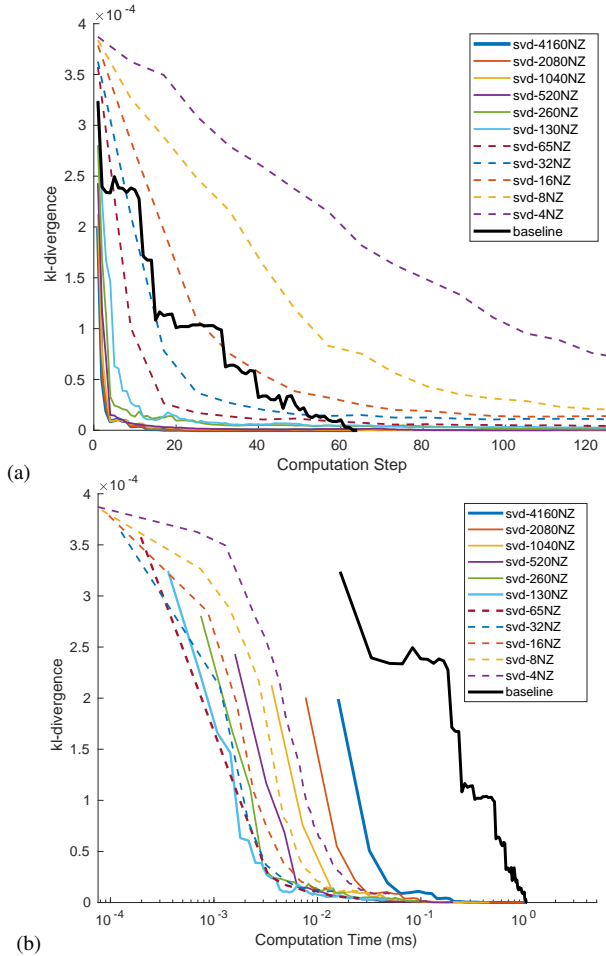


Fig. 6. KL-divergence between approximate prediction and reference model output as a function of: (a) Computation Step, (b) Computation Time.

constraint. This behaviour, 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. Fig. 7b illustrates two representative intermediate probability distributions obtained by an instance of the proposed approach and the baseline, both satisfying the same latency constraint. The distributions indicate that the proposed approach makes a more informed prediction, significantly closer to the ground-truth compared to the baseline.

VII. CONCLUSION

The deployment of LSTMs in latency-critical applications is a challenging task due to their high computational requirements. In this paper, 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.

REFERENCES

[1] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

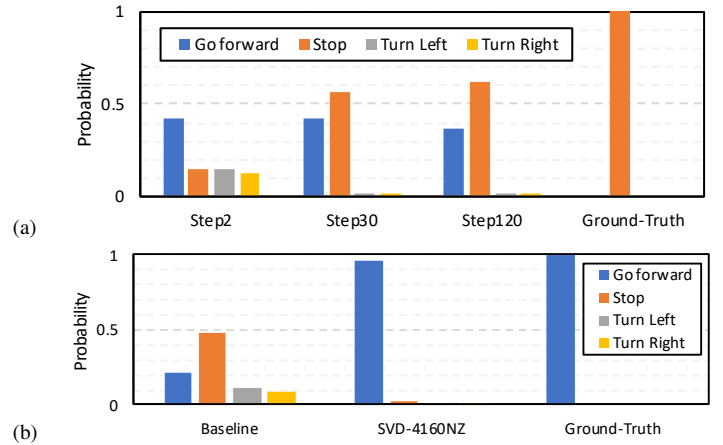


Fig. 7. Intermediate prediction instances obtained by: (a) the proposed approximation scheme with $NZ=8$ in various time-steps of the computation, (b) the baseline and the proposed approach with $NZ=4160$, under the same latency constraint ($t=10^{-1}$ ms).

- [2] 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 *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 44, no. 20, 2000.
- [3] F. Yu *et al.*, “BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling,” *arXiv preprint arXiv:1805.04687*, 2018.
- [4] Y. Guan *et al.*, “FPGA-based Accelerator for Long Short-Term Memory Recurrent Neural Networks,” in *ASP-DAC*, 2017, pp. 629–634.
- [5] A. X. M. Chang and E. Culurciello, “Hardware Accelerators for Recurrent Neural Networks on FPGA,” in *ISCAS*, 2017.
- [6] S. Han *et al.*, “ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA,” in *FPGA*, 2017.
- [7] Z. Wang, J. Lin, and Z. Wang, “Accelerating Recurrent Neural Networks: A Memory-Efficient Approach,” *TVLSI*, vol. 25, no. 10, 2017.
- [8] X. Zhang *et al.*, “High-Performance Video Content Recognition with Long-Term Recurrent Convolutional Network for FPGA,” in *FPL*, 2017.
- [9] R. Shokri and V. Shmatikov, “Privacy-Preserving Deep Learning,” in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015.
- [10] A. Kouris, S. I. Venieris, and C. Bouganis, “CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks,” in *FPL*, 2018.
- [11] M. J. Wainwright, M. I. Jordan, and J. C. Duchi, “Privacy aware learning,” in *Advances in Neural Information Processing Systems*, 2012.
- [12] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, “Predicting Parameters in Deep Learning,” in *NIPS*, 2013.
- [13] M. Rizakis, S. I. Venieris, A. Kouris, and C. Bouganis, “Approximate FPGA-Based LSTMs Under Computation Time Constraints,” in *ARC*, 2018.
- [14] S. Williams, A. Waterman, and D. Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures,” *Communications of the ACM*, vol. 52, no. 4, 2009.
- [15] R. McAllister *et al.*, “Concrete Problems for Autonomous Vehicle Safety: Advantages of Bayesian Deep Learning,” in *IJCAI 2017*.
- [16] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in *Advances in neural information processing systems*, 1989.
- [17] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *CVPR 2012*.
- [18] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, “Computer Architectures for Autonomous Driving,” *Computer*, vol. 50, no. 8, pp. 18–25, 2017.
- [19] M. Bojarski *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
- [20] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [21] L. Chi and Y. Mu, “Deep steering: Learning end-to-end driving model from spatial and temporal visual cues,” *arXiv:1708.03798*, 2017.
- [22] S. Thrun, “Toward robotic cars,” *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.
- [23] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-To-End Learning of Driving Models From Large-Scale Video Datasets,” in *CVPR*, 2017.