Mobiprox: Supporting Dynamic Approximate Computing on Mobiles

Matevž Fabjančič, Octavian Machidon, Hashim Sharif, Yifan Zhao, Saša Misailović, and Veljko Pejović

Abstract—**Runtime-tunable context-dependent network compression would make mobile deep learning (DL) adaptable to often varying resource availability, input "difficulty," or user needs. The existing compression techniques significantly reduce the memory, processing, and energy tax of DL, yet, the resulting models tend to be permanently impaired, sacrificing the inference power for reduced resource usage. The existing tunable compression approaches, on the other hand, require expensive retraining, do not support arbitrary strategies for adapting the compression and do not provide mobile-ready implementations. In this article, we present Mobiprox, a framework enabling mobile DL with flexible precision. Mobiprox implements tunable approximations of tensor operations and enables runtime-adaptable approximation of individual network layers. A profiler and a tuner included with Mobiprox identify the most promising neural network approximation configurations leading to the desired inference quality with the minimal use of resources. Furthermore, we develop control strategies that depending on contextual factors, such as the input data difficulty, dynamically adjust the approximation levels across a mobile DL model's layers. We implement Mobiprox in Android OS and through experiments in diverse mobile domains, including human activity recognition and spoken keyword detection, demonstrate that it can save up to 15% system-wide energy with a minimal impact on the inference accuracy.**

Index Terms—**Approximate computing, context-awareness, mobile deep learning (DL), ubiquitous computing.**

Manuscript received 25 August 2023; revised 2 December 2023; accepted 7 February 2024. Date of publication 14 February 2024; date of current version 25 April 2024. This work was supported in part by the Slovenian Research Agency through the Bringing Resource Efficiency to Smartphones With Approximate Computing Project under Grant N2-0136; in part by the Context-Aware On-Device Approximate Computing project under Grant J2-3047; and in part by the Research Programmes under Grant P2-0098 and Grant P2-0426. *(Corresponding author: Veljko Pejović.)*

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Institutional Review Board of the Faculty of Computer and Information Science, University of Ljubljana.

Matevž Fabjančič is with Cosylab, 1000 Ljubljana, Slovenia (e-mail: matevz.fabjancic@gmail.com).

Octavian Machidon is with the Faculty of Computer and Information Science, University of Ljubljana, 1000 Ljubljana, Slovenia (e-mail: octavian.machidon@fri.uni-lj.si).

Hashim Sharif was with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA. He is now with AMD Research, Santa Clara, CA, USA (e-mail: hsharif3@illinois.edu).

Yifan Zhao and Saša Misailović are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL 61801 USA (e-mail: yifanz16@illinois.edu; misailo@illinois.edu).

Veljko Pejović is with the Faculty of Computer and Information Science, University of Ljubljana, 1000 Ljubljana, Slovenia, and also with the Computer Systems Department, Institute Jozef Stefan, 1000 Ljubljana, Slovenia (e-mail: veljko.pejovic@fri.uni-lj.si).

Digital Object Identifier 10.1109/JIOT.2024.3365957

I. Introduction

POWERFUL services enabled by deep learning (DL), such as real-time camera-based object detection, online translation, and human activity recognition (HAR), are becoming increasingly available on mobile devices. Indeed, DL is an integral part of more than 12% of application downloads from the Android store platform [1]. However, the new affordances do not come for free—large DL models may overload the limited memory of mobile devices, the computational burden may lead to significant delays before the results are available, and the power needed for processing may quickly deplete the mobile's battery.

Reducing the complexity of neural networks (NNs) is the primary means of making DL mobile friendly. Such complexity reduction may be inherent to the network design—MobileNet [2], EfficientNet [3], and ShuffleNet [4] represent some of the architectures that are specifically designed for the mobile's limited memory resources. Yet, the computational burden of these networks may still be overwhelming for a wide range of heterogeneous edge devices [5]. Both memory and computational complexity can be further reduced by a gamut of NN compression techniques. These include parameter quantization [6], weight pruning [7], NN distillation [8], to name a few. The key issue with such complexity reduction is that the network parameters are permanently changed. Thus, in case that the resulting inference accuracy is reduced, that reduction remains permanent.

On mobiles, on the other hand, *DL compression needs to be adaptable to the context of use*: a compressed model that reliably recognizes a user's speech commands when used in a quiet indoor location, might completely fail in noisy outdoor environments; similarly, a user might tolerate a more compressed model that occasionally misclassifies her physical activity during her daily routine, but would require a more accurate model while exercising. A rigid approach to DL compression is against the often dynamic nature of mobile computing, where both a user's requirements with respect to the result accuracy [9], as well as the difficulty of given NN input [10], may vary as the context of use changes.

Recently, proposals have been made to enable dynamic accuracy/complexity adaptation of NNs. Examples include dynamic quantization by AnyPrecision [11], dynamic adjustment of layer width through slimmable NNs (SNNs) [12], or dynamic pruning proposed in [13]. Common for all of the above dynamic adaptation approaches is that they do not support prebuilt networks, but require specialized training that can take days or weeks for large data sets and architectures before real-time adaptation can be used. Furthermore, despite targeting dynamic environments, the above works do not actually provide mobile-ready implementations. Translating the benefits provided by high-level demonstrations (often implemented in PyTorch) to mobile energy savings requires significant engineering effort, as modern mobile DL frameworks, such as TensorFlow Lite, do not support the versatility of high-level frameworks such as PyTorch.

Advances in a different research area—compilers for heterogeneous systems—have recently addressed the issue of ’‘optimal” NN compilation, where individual tensor operations are implemented in accordance with underlying hardware capabilities. Along these lines, ApproxHPVM [14] enables the execution of convolutional NN (CNN) operations with varying degrees of approximation, provided that the hardware/OS supports approximate computation. However, ApproxHPVM targets server environments, generates only CUDA-ready binaries, and does not support compilation for mobile hardware (Android or iOS). With the help of ApproxTuner [15], approximation levels within ApproxHPVM can be dynamically adapted, yet, the provided adaptation method is simple, reactive and context-oblivious.

In this article, we present *Mobiprox*—a novel framework that enables context-adaptable approximation of DL operations executed on the mobile platform. Our guiding vision is that *data scientists are not mobile system experts*. Therefore, DL modeling should be disentangled from system-level performance optimization. Mobiprox aims to support efficient on-device execution of an arbitrary pretrained mobile network architecture. Furthermore, we do not require that a developer knows which optimizations (in our case—execution approximations) are available on the device. Still, we give a developer an option of (dynamically) setting an operational point along the inference accuracy versus resource usage tradeoff curve, yet, in the limit case, the developer need not even set this point, but merely let Mobiprox tune the execution according to its internal approximation adaptation algorithms.

We implement Mobiprox at low levels of the computing stack to support a wide range of NN architectures and embrace various approximation techniques exposed by the underlying hardware and the OS.[1] To support context-sensitive runtime adaptation Mobiprox identifies Pareto-optimal approximation configurations during the off-line tuning stage. The system then enables the network to glide across different speedup/accuracy tradeoff points during the runtime. The key novelty of Mobiprox are also the adaptation algorithms that guide the runtime approximation adaptation according to a given goal, e.g., maximal energy savings.

With Mobiprox, we address multiple challenges that stand in the way toward adaptable approximate mobile DL:

1) The difficulty of implementing approximate operations at an appropriate level of the mobile computing stack; Over the past two decades, a number of approximate computing techniques have been developed—from approximate adders and multipliers to loop perforation and task skipping [16]. Because of their small form factor, however, mobile devices can rarely accommodate both approximate and accurate versions of hardware circuits. Software techniques, on the other hand, often require strong developer involvement, e.g., in marking loops eligible for perforation. Therefore, we focus on software-level approximation of tensor operations. Since mobile DL frameworks (e.g., TensorFlow Lite) and even libraries of specialized functions for mobile DL (e.g., ARM Compute Library) aggregate tensor operations, we implement both approximate and precise tensor operations from basic linear algebra subprogram (BLAS) primitives.
2) The issue of modifying NN operation at runtime on a mobile device; mobile DL frameworks do not support dynamic graph reconfiguration, thus even the existing dynamic approximation schemes (such as SNNs [12]) do not work on mobiles; to overcome this limitation, we implemented our custom approximations at a fine-grained level and exposed the calls for setting the approximation level at runtime through Java Native Interface (JNI).
3) The lack of algorithms and tools for context-aware adaptation of mobile DL; a certain classification accuracy level might be acceptable in some situations, but not in others; in addition, an approximated DL model that works well for certain inputs, might not provide correct classification for some other inputs; finally, gauging model performance at runtime is challenging; we first devise proxies for measuring classification performance (the same-class instance counting-based and the Softmax confidence-based) and then develop algorithms (state-based, and confidence-based) for dynamically adapting the approximation.

Toward this end, we present the following contributions.

1) We develop an end-to-end approximate configuration search, selection, and compilation pipeline for mobile devices. Our solution integrates state-of-the-art heterogeneous compilation infrastructure, approximate configuration search framework, and a widely used LLVM compiler into an Android-ready pipeline; furthermore, our solution supports dynamic configuration loading.
2) We devise novel strategies for runtime approximation configuration adaptation; based on the problem properties or the classifier confidence, our solutions ensure that the desired inference accuracy is achieved with the minimal use of a mobile's resources.
3) We implement selected approximate computing primitives at a low-level of the mobile computing stack, supporting both on-CPU and on-GPU approximate execution of different tensor operations for mobile devices.
4) Our evaluation shows that Mobiprox brings substantial energy savings while preserving the classification

[1]The specific implementation presented in this article supports perforated convolution, filter sampling, and half-precision quantization.

accuracy. We perform experiments on both a single-board computer (for precise energy measurements) and on commodity smartphones performing real-time inference, using different NN architectures and multiple application domains, including human activity classification and spoken keyword recognition (SKR). Our evaluation demonstrates that, by adapting to the varying context (i.e., input data difficulty), Mobiprox can achieve energy savings while preserving the inference accuracy.

II. Related Work

Resource-Efficient DL on Mobiles: The expansion of mobile DL applications has been hindered by the high-resource consumption of DL models and the difficulty of the edge computing devices, such as battery-powered smartphones, to meet the resource and energy requirements of such applications [17]. Model representations, may including hundreds of millions of parameters and performing the classification of a single input vector can easily overwhelm the available computing and memory resources of edge computing platforms [18].

Efforts have, thus, focused on reducing the complexity, while preserving the inference power of DL models through weight quantization [6], pruning [7], [19], [20], knowledge distillation [8] and other methods [21]. High-level DL frameworks, such as PyTorch, do not readily support mobile platforms, thus, there are relatively few demonstrations of an on-device DL optimization. On the pruning front, PatDNN enables real-time inference using large-scale DL models (e.g., VGG-16 and ResNet-50) on mobile devices by harnessing pattern-based model pruning [7], while DeepIoT [22] uses reinforcement learning to guide the pruning process. Both solutions lead to significant model size reductions (90% to 98.9% in case of DeepIoT) and speedups (up to 44.5× in case of PatDNN) with no inference accuracy degradation in certain settings, demonstrating vast opportunities for mobile DL optimization. Parameter quantization, on the other hand, despite being actively researched [6], [23], [24], sees only limited implementation in the mobile realm. The main reason is the lack of support for arbitrary bit-width computation in today's mobile hardware.

Dynamic Compression Adaptation: All of the above approaches share a common drawback: once the approximation is applied, the resulting network remains unchanged during runtime. Thus, such approaches enable operation at a single fixed point on the *accuracy-resource usage* tradeoff curve regardless of how the context in which inference is performed changes during runtime. However, this operation is inappropriate for the mobile domain, since the changing context of use is a defining trait of mobile computing and can significantly affect the requirements imposed on the DL inference. For instance, a smartphone may or may not be connected to a charger, calling for more or less energy-efficient operation; sensor data may be more or less noisy, requiring more or less complex DL models; depending on the intended use, a user may require more or less accurate inference results from a mobile app. Recent research therefore focuses on enabling accuracy-resource usage tradeoff by dynamically adjusting the compression level without the need for retraining the network.

The initial solutions enabling dynamic adaptivity, such as MCDNN [25], relied on having several differently compressed candidate DL models in the cloud and downloading the most appropriate model on the device according to the current context. While enabling context-adaptation, this strategy adds substantial overheads of model transfer. Early exit networks can dynamically reduce the computational complexity of a single model by not traversing all network layers and halting the computation at one of intermediate exit points in the network instead [26]. SPINN [27] introduces a scheduler that co-optimizes the early exit policy and DL model splitting at run time, in order to adapt to dynamic conditions and meet user-defined service-level requirements in a cloud-edge environment. The drawbacks of early exit schemes include the need for off-the-shelf models to be restructured and retrained and the complexity of developing exiting policies that will be suitable for a particular operational domain. Unfortunately, despite intended to work in dynamic environments, neither MCDNN nor SPINN have been implemented on mobiles. DeepX compresses network layers using singular value decomposition and enables execution on heterogeneous mobile hardware. However, it supports only fully connected NN layers and the project code is not publicly available.

Finally, pruning and quantization have also been revised to support dynamic adaptation. Runtime neural pruning (RNP) framework [13] enables bottom-up, layer-by-layer pruning guided by a Markov decision process and reinforcement learning. The importance of each convolutional kernel is assessed and based on it channelwise pruning is performed, where the network is pruned more when classifying an "easy-to-classify" input instance. A different approach for dynamic compression adaptation is the SNN [12]. The method trains a single CNN and then executes it at different widths (number of channels in a layer), permitting runtime adaptive accuracy-efficiency tradeoffs at runtime. There is no need for retraining or loading different models: the network adjusts its width on the fly, based on the resource constraints, input difficulty, or other contextual factors. Any-Precision approach [11] proposes a CNN training method that allows the network to adapt its numerical precision during inference to support dynamic speed and accuracy tradeoff. Yet, neither RNP, SNN, nor Any-Precision apply to already trained networks, nor have these techniques been implemented in the mobile realm. The reason for this is that, unlike our approach, the above methods were not originally planned with mobile platform restrictions in mind. SNNs, for instance, rely on dynamic NN graph reconfiguration, something that none of the mobile DL frameworks (e.g., TensorFlow Lite, Pytorch Mobile, etc.) supports at the moment.

III. Preliminaries

Mobiprox builds upon the existing work on heterogeneous and approximate computing compilers.

Heterogeneous parallel virtual machine (HPVM) [28] is a compiler infrastructure targeting heterogeneous hardware. It introduces HPVM-C, a programming language for defining *data flow graphs* (DFGs), directed graphs in which nodes represent a computation task and edges represent inputs and outputs of a computation task. Computation workloads are defined using HPVM's intrinsic functions used to specify the target device the node will be executed on, node inputs, node outputs, and any compute operations (e.g., addition). HPVM compiler achieves parallel execution of produced binaries by identifying dependencies among the nodes in a DFG and generating compute code for specified target devices (CPU, GPU) for each node.

ApproxHPVM [14] expands HPVM by introducing support for NN tensor operations: multiplication, convolution, addition, pooling, and activation functions. Additionally, ApproxHPVM enables transforming high-level descriptions of CNNs (in frameworks, such as Keras and PyTorch) into DFGs in the form of generated HPVM-C source files. However, while HPVM generates code for computation nodes in a DFG, ApproxHPVM's tensor operations are mapped to functions defined in the *HPVM Tensor Runtime* library. In ApproxHPVM individual tensor operations can be marked with the maximum allowed level of approximation, and the compiler then ensures that these are mapped to the appropriate underlying approximate computing techniques (either software or hardware-based). Yet, ApproxHPVM's tensor operations are supported for Nvidia CUDA-enabled devices only. In this article, we introduce a novel OpenCL implementation that enables approximate tensor operation execution on Android devices (Section IV-B2).

ApproxTuner [15] delivers heuristic-based search of the space of possible approximations of each individual network layer, so that a comprehensive *speedup-inference accuracy* tradeoff curve is charted and the list of the most promising sets of approximations is identified. Yet, ApproxTuner does not take into account the peculiarities of the mobile platform and the predicted tradeoff curves it draws do not reflect the actual performance observed on the mobiles. Consequently, in this article, we build a new cross-platform approximation profiler based on ApproxTuner (Section IV-B3).

A. Approximation Techniques

We identified the following generally applicable approximation techniques that can be employed at a level of a single NN operation and are supported by commodity mobile hardware, and we implemented them in Mobiprox.

Convolution perforation [29] is an approximation that skips certain input matrix coordinates when calculating convolution, as shown in Fig. 1. Due to the nature of convolutions, this does not necessarily mean that the inputs at skipped coordinates are never used—indeed, the inputs are used in neighboring convolutions. This, in turn, makes it feasible to interpolate convolution results at skipped coordinates by computing the average of computed neighboring cells. We support two types of convolution perforation—*row perforation* and *column perforation*. The parameter `offset` defines the index of the first omitted row or column, while parameter `stride` defines the interval between the skipped rows/columns. In Fig. 1, parameters `stride`=2 and `offset`=1 were used.

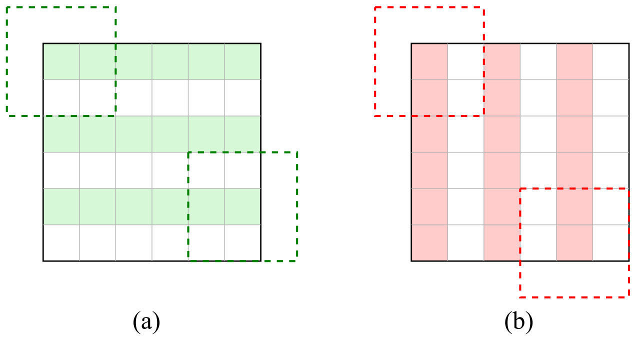


Fig. 1. Perforated convolution. Colored sections indicate convolution coordinates. Dashed squares indicate the area of the first and the final convolution. (a) Row perforation. (b) Column perforation.

Filter sampling approximates the filters that the convolutions are performed with. In CNNs filters are 4-D tensors with dimensions $[N, C, H, W]$. N represents the number of filters in the convolution, C is the number of feature channels in the input and the filter, and H and W represent the height and width of the filter, respectively. Each filter is therefore composed of $n_{\text{elm}} = C \cdot H \cdot W$ components. Filter sampling with `stride` k removes every kth component of the filter's n_{elm} components, starting at element specified by `offset`. The technique, thus, reduces the amount of computation by keeping only $n_{\text{elm}-\text{samp}} = n_{\text{elm}} - (n_{\text{elm}} - \mathtt{offset}/\mathtt{stride})$ filter components at the cost of the overall convolution accuracy. To interpolate missing values, each retained filter component is multiplied by a factor of $n_{\text{elm}}/n_{\text{elm}-\text{samp}}$.

Finally, Mobiprox also provides an optional *half-precision quantization* that can be used to approximate any floating point tensor operation. While such quantization is meaningful only if the underlying hardware supports it, we opted for enabling it as modern mobile GPUs, such as those of Arm Mali series, natively support the IEEE FP16 16-bit format.

IV. Mobiprox Framework

Mobiprox, our novel framework for enabling dynamic approximation of mobile DL, is sketched in Fig. 2. An Android app compiled with Mobiprox can use an arbitrary runtime approximation adaptation strategy for its DL models (e.g., "run low-quality network when battery is low," "run high-quality inference when user is at a specific location," etc.). To achieve this, Mobiprox operates with approximation configurations, i.e., combinations of per-layer approximations of a pretrained DL model. Mobiprox first uses ApproxTuner to examine the impact of different configurations on the inference accuracy and the speedup. Each approximation configuration yields a point in the accuracy–speedup space, and Mobiprox identifies the most promising configurations that form the Pareto front in this space and then profiles their actual performance on the mobile platform using the novel *HPVM Profilier for Android*. Mobiprox's Android-based *OpenCL runtime* then enables execution of and dynamic switching

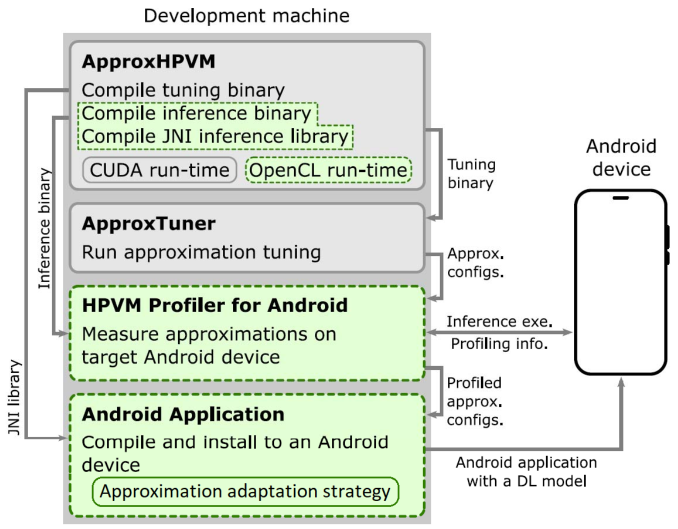


Fig. 2. Mobiprox overview. OpenCL run-time supports running the inference binary (controlled either directly from the C code or via JNI from the main Java/Kotlin app) with a varying level of approximation. The HPVM profiler for Android helps us chart the *approximation-resource usage* space, so that the approximation adaptation strategy within the Android app can set the approximation level dynamically at runtime. Main Mobiprox modules are colored green, while the supporting pre-existing modules are grayed out.

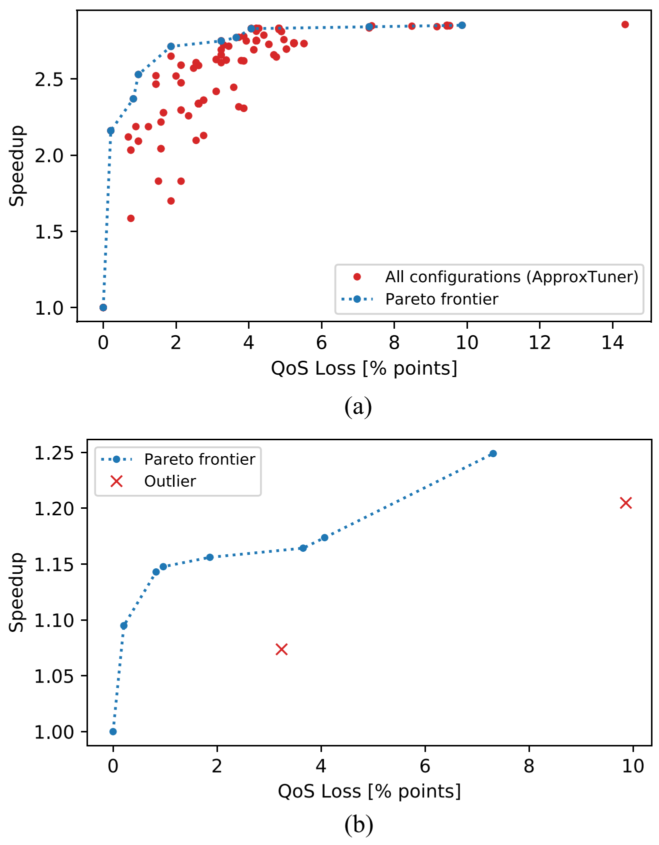


Fig. 3. Comparison of the achieved speedup and the resulting QoS (inference accuracy) loss for approximation configurations selected by the on-server tuning with the same configurations ran on a mobile platform. Note the different scaling of the *y*-axis. (a) Tuning on server. (b) Tuning on mobile.

between approximation configurations on a mobile device. Using the *JNI interface library* generated by the framework, the mobile application can control the approximation level of the NN. Finally, as part of Mobiprox, we also devise *Approximation adaptation strategies* that leverage the generated tradeoff curves to match the required and delivered quality of computation, thus enable energy-efficient DL on mobile devices.

A. Charting Approximation Space

Each of the approximation techniques described in Section III-A exposes one or more *approximation knobs* that can change the level of approximation and thus adjust the accuracy and the execution time (consequently the energy efficiency) of a tensor operation. These knobs are `offset` and `stride` for convolution perforation and filter sampling, and an indicator `_fp16` of whether an operation is executed using half-precision quantization. An *approximation configuration* is a set of pairs $\langle$Op., KnobValue$\rangle$ for all operations in a given NN. Each of the configurations leads to a single *tradeoff point* on an speedup-accuracy tradeoff curve.

The tuner heuristically searches the space of possible approximations and determines a Pareto frontier of approximation configurations that maximize the execution speedup at different Quality of Service (QoS) loss points. This loss is a real number defined as a difference between the classification accuracy, over a representative validation data set, of a nonapproximated and an approximated DL model.

However, the method described above for determining the optimal approximation configurations does not readily translate to mobile devices. The mobile platform is substantially different from the server used for fast heuristic-based approximation configuration profiling. The specifics of GPU-based execution (e.g., CUDA versus OpenCL), heterogeneous CPUs with fewer cores, and other factors mean that the results of the configuration search performed on a server are a rather poor representation of the actual approximated NN performance on a mobile. In Fig. 3(b), on the example of a MobileNetV2 model used for HAR (detailed in Section VI), we show the actual on-mobile-device speedup and QoS loss achieved by the approximation configurations that ApproxTuner identified as the most promising. While the Pareto points obtained on a server generally remain relevant, the achieved on-device speedup is about 50% lower on the mobile.

Mobiprox therefore introduces a novel configuration identification approach. First, we perform tuning on a computer cluster to identify candidate approximation configurations. Then, we develop an Android-based profiler (described in Section IV-B3) that runs each candidate configuration on a mobile device and obtains a realistic picture of the approximated NN performance. The resulting picture of the speedup—QoS loss space charted by these configurations is then used to guide the dynamic adaptation of the approximation. As a final result, the profiler creates a file listing configurations that will be switched during the mobile app runtime (according to a strategy, e.g., from Section V), yet

Algorithm 1: Mobiprox Compilation. Compilers Used at Each Step Are Shown in Comments

```
1 $IR_{LLVM}$ ← Transform source code into LLVM-IR ;        // Android LLVM
2 $IR_{HPVM}$ ← Transform $IR_{LLVM}$ into HPVM-IR ;          // ApproxHPVM
3 for each IR transformation $T_i$ of the compiler do
4     $IR_{HPVM}$ ← $T_i(IR_{HPVM})$ ;                        // ApproxHPVM
5 end
6 $IR_{LLVM}$ ← Transform $IR_{HPVM}$ into LLVM-IR ;          // ApproxHPVM
7 Compile $IR_{LLVM}$ to machine code ;                     // Android LLVM
```

only a single network model definition gets deployed on a mobile.

B. Mobiprox—Android Implementation

Mobiprox, as a concept, is not tied to a particular mobile platform. Yet, amassing 75% of the smartphone market share Android is the most common mobile DL platform and that stands to gain the most from dynamically adaptable approximation, thus, in this section we develop a full Mobiprox compilation pipeline targeting Android devices.

1) Mobiprox Android Compiler: Mobile application development with Mobiprox involves compiling the tuning binary and the inference binary (Fig. 2). While the tuning binary is confined to the server environment and is handled by the ApproxHPVM compilation pipeline, the inference binary is cross-compiled from a server to a mobile (Android). We implement a mechanism for turn-taking between ApproxHPVM and Android NDK LLVM compiler toolchains (Algorithm 1). We enable this by clearly partitioning the compilation steps and harnessing the fact that LLVM-based compilers apply transformations to an intermediate representation termed LLVM-IR. Note that ApproxHPVM extends LLVM-IR by defining HPVM-IR to which approximation-related transformations are applied. This clear division allows us to use Android NDK for generating the initial LLVM-IR suitable for Android applications and for generating the machine code containing approximate NN operations suitable for mobile GPUs in the final compilation step, while using HPVM-IR transformations for the internal part of the compilation pipeline to insert the description of the desired approximate tensor operations.

2) OpenCL Tensor Runtime for Android: A core component of Mobiprox Android is a Tensor Runtime, which implements tuneable approximable tensor operations for NN inference. The existing support for approximate NN operations for Nvidia CUDA GPUs [14] is not suitable for mobiles, which seldom host such hardware. Instead, Mobiprox implements an own tensor runtime using OpenCL, an open standard for GPU-accelerated computation which is available on a wide variety of hardware, including mobile platforms.

To enable an enhanced control over low-level concepts (such as memory allocation), we implemented the tensor runtime for Android using CLBlast [30], an OpenCL implementation of BLASs. However, this library is not intended for DL: it does not implement operations commonly used in NNs. Therefore, we extended CLBlast with the following operators: 1) pointwise tensor addition; 2) bias addition; 3) activation functions (ReLU, clipped ReLU, and tanh); 4) FP-16–FP-32 tensor conversion; 5) batch normalization; 6) pooling (min, max, and *average*); and 6) convolution approximations operators optimized with tiling and vectorization: image-to-column (im2col) transformations with row perforation, column perforation, and filter sampling, Kernel-to-row (kn2row) transformation with filter sampling, and *Interpolation* of missing values in convolution perforation. Finally, during the mobile app compilation JNI is exposed, enabling the tensor runtime initialization and destruction, NN inference invocation, and dynamic approximation configuration loading.

3) HPVM Profiler for Android: To assess the speedups and consequently the energy efficiency of approximated NNs we implement a profiler tool. The profiler, in the form of a Python library, for a given NN binary measures the accuracy, Softmax confidence, and execution time of NN inference on a given test data set. Due to a high discrepancy between the speedup observed on a mobile device and on a server for the same approximated network (Fig. 3), the profiler uses the Android Debug Bridge (ADB) [31] to run measurements on an actual Android device and to transfer the profiling information files back to the host machine for analysis.

V. Approximation Adaptation Strategies

Mobiprox's key strength is its support for context-based adaptation of mobile DL approximation. The framework itself deliberately does not prescribe the adaptation strategy allowing a developer to implement an arbitrary set of rules driven by energy needs (e.g., "use higher approximation when battery level falls below 10%"), the purpose of use (e.g., "use more accurate HAR models when a user is exercising"), or even business models (e.g., "use input-adaptable approximation for premium users"). Programming such strategies is trivial, yet, one can envision a more challenging-to-achieve goal, such as "minimize the energy usage without sacrificing the inference accuracy." In this section, we harness the natural temporal dependence of the instances of sensed data that is characteristic in many mobile computing applications, and devise two strategies demonstrating that a widely applicable goal of energy minimization can be met with Mobiprox.

A. State-Driven

Many mobile sensing domains deal with the recognition of states that do not vary rapidly over time: human physiological signals do not change erratically, people have conversations, not random utterances, movement is continuous in space, etc. Our state-driven adaptation strategy is based on the observation that rapid variations, especially in human behavior, are rare (e.g., [32]). We hypothesize that inputs that are less difficult to classify can be processed with more "aggressive" energy-saving approximation configurations, whereas more difficult-to-classify inputs require computationally more expensive, more accurate configurations, and that the "difficulty" of the input correlates with the class an instance belongs to.

Starting from this assumptions we implement an adaptation algorithm that adjusts the approximation configuration based on the reliability of classification determined by looking at a subset of the most recent predictions made by the network.

Algorithm 2: State-Driven Adaptation Engine

```
M = [] ;                    // FIFO memory with maximal capacity $N$
V = 0 ;                     // Reliability index on interval $[-V_L, V_L]$
while $p$ = nextPrediction() do
    push($M, p$);
    if len($M$) < $N$ then
        continue;
    end
    if all predictions in $M$ are equal then
        $V = \max(0, V) + 1$;
    end
    else
        $V = \min(0, V) - 1$;
    end
    if $V \leq -V_L$ then
        Approximate less;
    end
    else if $V \geq V_L$ then
        Approximate more;
    end
    pop(M);
end
```

After each inference, a vote is cast on the measure of reliability V, which is increased by 1 if all previous N predictions are equal, and decreased by 1 otherwise. The functionality of this approach is described in detail in Algorithm 2.

In this algorithm, V_L refers to the number of required votes that need to be cast consecutively in order to change the approximation configuration—this parameter avoids the situation where the configuration is changed at every inference point. The second parameter N defines the capacity of the FIFO memory M. A larger memory would increase the robustness of the algorithm to classification errors (since it will consider a larger subset of previous predictions), but at the same time would hinder switching to more approximate configurations after a change in the observed/modeled phenomenon.

B. *Confidence-Driven*

In the second adaptation strategy, we use the classifier's confidence as a proxy for accuracy. The Softmax layer probability can accurately reflect the actual confidence of the classifier [33]. However, Guo et al. [34] pointed out that calibration is required to achieve a high correlation between the Softmax confidence and the expected inference accuracy. Hence, we perform calibration by applying the temperature scaling during Softmax confidence calculation. More specifically, for an N-class classification task where the N-dimensional vector z contains class scores, for any class i, its calibrated Softmax confidence is computed as

$$\sigma_i(z; T) = \frac{e^{z_i/T}}{\sum_{j=1}^{N} e^{z_j/T}} \quad (1)$$

where T is a scalar temperature parameter, which "softens" the Softmax (raises the output entropy) when $T > 1$ and is optimized with respect to negative log likelihood on the validation data set, so that the confidence value for the datapoints classified with accuracy p is as close a possible to p [34].

Our adaptation strategy then uses the calibrated Softmax confidence to identify incorrect classifications. The Android profiler (Section IV-B3) also reports per-class confidence averages for correct ($C_+^{(i)}$) and incorrect ($C_-^{(i)}$) predictions and adds this information to approximation configuration files. The algorithm is then driven by a hysteresis outlined by two thresholds $C_{\text{less}}^{(i)}$ and $C_{\text{more}}^{(i)}$, where $C_-^{(i)} > C_{\text{less}}^{(i)} > C_{\text{more}}^{(i)} > C_+^{(i)}$. If the classification confidence of the predicted class of the immediately preceding instance is higher than $C_{\text{more}}^{(i)}$, the algorithm moves toward more aggressive approximation. If it is lower than $C_{\text{less}}^{(i)}$, the algorithm moves toward less approximated configuration. We empirically find that the values of $C_{\text{less}}^{(i)}$ halfway and $C_{\text{more}}^{(i)}$ three-quarters-way between $C_-^{(i)}$ and $C_+^{(i)}$, respectively, perform well in our experiments.

VI. Experimental Setup

To evaluate our framework we first, through a series of microbenchmarks, assess the energy savings and speedup achieved through approximate neural networking operations implemented in Mobiprox, and also evaluate the overhead incurred by the accuracy–speedup profiling that Mobiprox relies on. Then, we evaluate Mobiprox's dynamic adaptation in two domains: 1) HAR and 2) SKR. Finally, we deploy Mobiprox on commodity Android phones and demonstrate its usability for real-time adaptable DL inference.

Microbenchmarks With Standard Architectures: We evaluate Mobiprox first through a series of experiments aiming to assess the energy savings and speedup achieved through different approximate NN operations over a selection of networks. We first investigate the performance on standard image recognition architectures (AlexNet, VGG16, and MobileNet) and the CIFAR-10 data set. However, since Mobiprox primarily targets dynamic mobile environments and inference from time-series sensor data, we also include two NN architectures (MobileNet and ResNet50) trained on UCI-HAR data set [35]. We implement all networks in PyTorch.

Mobiprox is fully compliant with consumer off-the-shelf Android devices. Yet, modern unibody smartphones do not allow for batteries to be easily removed, precluding the use of high-accuracy power metering. Therefore, when energy consumption is examined, we use ASUS TinkerBoard S[2] single-board computer running Android 7 OS. We power it through the high-frequency Monsoon Power Monitor and use the accompanying PowerTool[3] measurement processing software. Our Python-based profiler using ADB runs compiled approximated NNs on the board. The approximation's main, yet not the only (as we will see in Section VII-B) impact on the energy consumption stems from the decreased DL processing time. The profiling for each network is, thus, executed on a predefined fraction of the data in 10 batches for UCI-HAR networks and in 8 batches for CIFAR-10. This was done to 1) reduce overall time requirement and 2) obtain more robust measurements by measuring each batch separately. We report the mean and standard deviation of each configuration's energy consumption.

[2]https://tinker-board.asus.com/product/tinker-board-s.html
[3]https://www.msoon.com/hvpm-software-download

Real-World HAR Traces: We assess the expected energy savings Mobiprox brings in real-world environments by taking a recent trace of human activity obtained through a body-mounted mobile sensing platform [36]. The data set contains traces of 21 participants (13m, 8f), with an average age of 29 (std. dev 12) years. The traces consist of the acceleration and angular velocity in all three axes sampled at 50 Hz from an UDOO Neo Full board,[4] a compact IoT embedded computing device equipped with an accelerometer and a digital gyroscope, strapped to each participant's waist.

In this study, which took place at a university campus, the participants performed the six activities in a row. First the static ones—sitting, standing still, and lying—for 2 min each. Then, the dynamic activities—walking up and down a hallway (summing up to two to three minutes for each participant) and walking down and up the stairs (about 45 s in each direction, the duration being limited by the total number of stairs). This experiment features the same six activities that are present in the original UCI-HAR data set. Yet, by using traces collected with a different device, in a different environment, and with different participants than the original experiment, we aim to obtain a realistic picture of Mobiprox's ability to adapt to previously unseen users. Separately, we conduct an experiment with Mobiprox running directly on Android smartphones that further assesses the performance of our framework with HAR in unscripted scenarios (elaborated below).

Real-World SKR Traces: We also examine the savings Mobiprox brings in real-world environments by considering the problem of a SKR from microphone recordings. For this we use the Google Speech Commands (GSC) v0.01 [37], a data set containing 65,000 one-second long utterances of 30 short words by thousands of different speakers. Interested in the recognition of keywords in realistic situations, where a word has to be spotted in sound segments that may also contain words we are not interested in as well as recordings of silence, we follow the approach presented in [38] and use twelve classes for ten selected keywords (yes, no, up, down, left, right, on, off, stop and go) and two extra classes: 1) "unknown" (for the remaining 20 words in the data set) and 2) "silence." In Section VII, we evaluate Mobiprox's ability to bring energy savings when a compact NN is used for on-device SKR within the GSC-based trace.

Live Smartphone-Based HAR: We recruit ten users (all students or staff at University of Ljubljana, 7 female/3 male) to perform a 10-min experiment during which they are given an option of conducting six activities: 1) sitting; 2) standing still; 3) lying; 4) walking; 5) going up the stairs; and 6) going down the stairs. Unlike with the lab-based studies, such as [35], the order and the duration in which the activities were to be performed, was not in any way prescribed in our experiment. A Samsung Galaxy M21 smartphone, in the portrait orientation with the screen facing forward, was attached to each user's waist. The phone sampled accelerometer and gyroscope at 50Hz, and ran Mobiprox for live on-device inference of human activity using dynamically approximated NN. The model used was `mobilenet_uci-har` pretrained on UCI-HAR data set without any further retraining. Finally, accelerometer and gyroscope samples were stored and reran through a non-approximated `mobilenet_uci-har` model to obtain the baseline activity prediction (the output of the nonapproximated model is not the ground truth, as due to the unscripted nature of the experiment we do not know the exact activity a user performed at a given moment).

[4]https://shop.udoo.org/en/udoo-neo-full.html

TABLE I
TIME NEEDED FOR IDENTIFYING PARETO-OPTIMAL CONFIGURATIONS USING A SINGLE GPU ON A SUPERCOMPUTER. STANDARD DEVIATIONS ARE IN THE PARENTHESES. THE DURATION MAY VARY WITH DIFFERENT SEARCH PARAMETERS

DL model	Initial tuning time [s]	Subsequent tuning time [s]
alexnet2_cifar	574 (97)	184 (6)
mobilenet_cifar10	1301 (226)	370 (18)
vgg16_cifar10	1216 (62)	264 (2)
resnet50_uci-har	1602 (316)	276 (14)
mobilenet_uci-har	1203 (246)	253 (1)

VII. EVALUATION

In our evaluation of the Mobiprox framework we aim answer to the following research questions.

1) *RQ1: Usability:* Time to find configurations and generalizability across different devices?
2) *RQ2: Generalizability Across NN Models:* How does Mobiprox perform across different NN models and classification tasks, in terms of accuracy, speedup, and energy saved?
3) *RQ3:* What energy savings would Mobiprox bring in a real world scenario and at what tradeoff with regard to inference accuracy?

A. Configuration Identification Time and Generalizability

In Mobiprox, the identification of suitable approximation configurations is split into two phases: 1) identifying the candidate Pareto front among all possible configurations and 2) measuring each configuration's speedup and inference accuracy on the target platform. The first part of the configuration identification process relies on ApproxTuner and is performed on a CUDA GPU-enabled machine. The second part of the process must be executed on a machine (e.g., a PC or a laptop) that connects to the target mobile platform via ADB. In this step, the candidate configurations get executed directly on the mobile, thus, what matters are the capabilities of the connected mobile platform, not the machine that controls the execution.

In our experiments we use a single node of a grid supercomputer equipped with Nvidia Quadro GV100 GPUs for the first phase of the configuration identification process. The node uses a single GPU for the tuning task, which is executed in a batch processing mode. In Table I we list the times needed for finding the Pareto front of the configurations for different networks used in our experiments. The heuristic search done by ApproxTuner is not deterministic, thus different runs may be completed in slightly different amounts of time. In addition, subsequent tunings of the same network often take significantly less time, as they build upon already cached results. In any case, we observe that the even the most complex tuning (for resnet50_uci-har) completes in less than 30 min.

TABLE II
TIME NEEDED FOR PROFILING PARETO-OPTIMAL CONFIGURATIONS' SPEEDUP AND ACCURACY ON A LOW-END MOBILE DEVICE

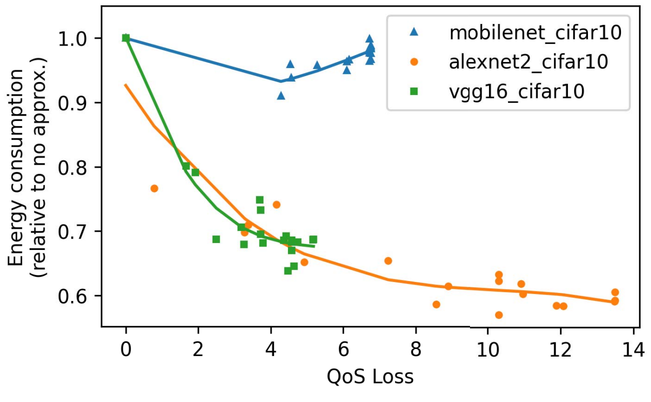
DL model	Dataset (batch)	Configs.	Profiling time [mins]
alexnet2_cifar	800 (100)	20	36
mobilenet_cifar10	800 (100)	20	58
vgg16_cifar10	200 (25)	20	42
resnet50_uci-har	250 (50)	20	26
mobilenet_uci-har	1450 (145)	10	42

For the second phase of configuration identification we use an ASUS TinkerBoard S, which, with its Rockchip RK3288 system-on-chip released in 2014, represents a lower end platform. In Table II we list the times needed for measuring the classification accuracy and the speedup for all Pareto front configurations found in the first phase of the identification process. The size of the test data set, the batch size, as well as the number of configurations in the Pareto front varied for different networks. Nevertheless, this one-off process completes in less than 58 mins. even on our low-end mobile platform, thus, we conclude that Mobiprox can be comfortably used within the existing Android application compilation process.

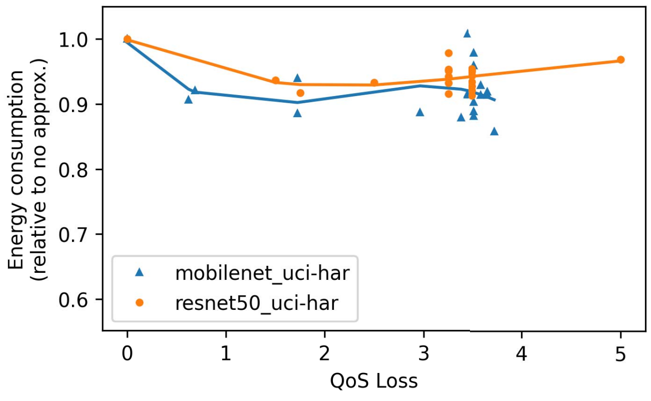
To answer RQ1, we now discuss the generalizability of an application compiled with Mobiprox. Mobiprox requires an actual mobile hardware for the second part of the configuration identification. Tens of thousands of different Android devices exist on the market, however, the choice of which to use should not have a major impact on the approximate configuration profiling. First, despite different hardware, the platforms use the same OpenCL-based primitives we developed in Section IV-B2. Second, while the speed at which NN will be executed may differ among different devices, there is no reason to expect that speedups (relative to a nonapproximated network) will be different for the same configuration ran on different devices. This is especially true if these devices belong to the same architectural category. Currently, Android apps can be built for four such categories, i.e., application binary interfaces (ABIs). Yet, 99% of the smartphones rely on one of the two ARM-based ABIs,[5] armeabi-v7a and arm64-v8a, both of which we successfully tested on ASUS TinkerBoard S and a range of phones (Samsung Galaxy M21, Samsung Galaxy S21, Xiaomi Pocophone) in our lab. For the DL models presented in this article we have not observed any differences in ordering among speedups obtained by different configurations on the platforms we have experimented with.

B. Popular CNN Benchmarks

To answer RQ2, we obtain approximation configuration sets for NNs trained on CIFAR-10 and UCI-HAR data sets using the Mobiprox tuner[6] and present the profiling results in Fig. 4(a) and (b). The reported energy consumption reduction is *system-level*, i.e., idle consumption has not been subtracted. We can observe a key difference between the approximation



(a)



(b)

Fig. 4. System-wide energy consumption (relative to no approximation) of an ASUS TinkerBoard S running inference on NNs trained different data sets. Different point types correspond to different NN architectures; each point represents a single approx. configuration. The *x*-axis represents the actual QoS loss from the model deployed on a mobile device. (a) CIFAR-10. (b) UCI-HAR.

configurations of different NN architectures—larger networks, such as VGG and AlexNet, are more amenable to approximation, and Mobiprox yields higher energy savings for these networks. This may suggest that certain models that are prohibitively expensive for mobile DL can be made mobile-ready using Mobiprox even without retraining.

It is interesting to juxtapose the measured energy savings with the speedup expected at the tuning time. In Fig. 3(a) in Section IV-A, we show that server-based profiling indicates that our approximations can lead to more than 2.5× speedup of inference on the MobileNet architecture trained on the UCI-HAR data set. The actual reduction in energy consumption is much smaller and is consistent with the speedup measurements presented in Fig. 3(b). We believe that realizing the full potential of the approximation on the mobile platform requires careful consideration of the mobile processing hardware. The overheads and thread scheduling inefficiencies in the ARM Cortex-A17 computing architecture on which the experiments were performed may be a likely culprit [39].

In Fig. 5 we analyze how speedups translate to energy consumption reduction. We observe a clear relationship between the two lines indicating that a higher speedup indeed reduces

[5]https://stackoverflow.com/questions/46453457/which-android-abis-cpu-architectures-do-i-need-to-serve

[6]For clarity, we limit the number of configurations and instruct the tuner not to consider configurations that result in QoS loss below a given threshold.

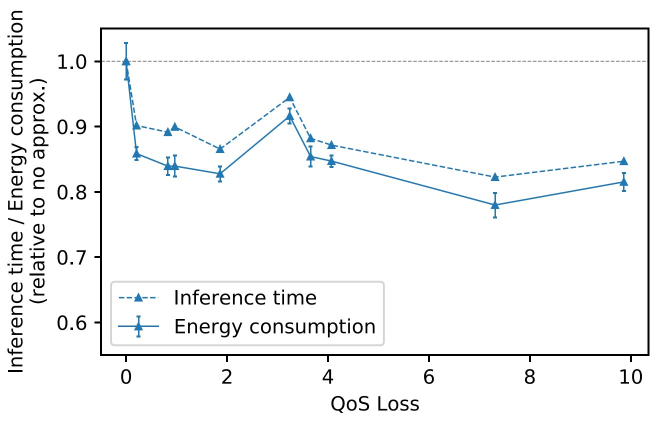


Fig. 5. Relative energy consumption compared to relative inference time reduction for `mobilenet_uci-har` at various approximation configurations. The x-axis shows the actual QoS loss from the model deployed on a mobile.

energy consumption. Due to dynamic voltage-frequency scaling this relationship does not necessarily hold for any general computing task, as at light loads the CPU/GPU governor might lower the CPU/GPU frequency, and thus, the power consumption. This would lead to a more complex relationship among power, energy, and speedup. However, DL computation is, based on our experience with mobile devices, highly demanding leaving no space for the governor to reduce the frequency and make the speedup—energy consumption relationship nontrivial.

The adaptation strategy calculation, i.e., deciding which approximation level to use, plays virtually no role in the overall energy consumption. Irrespective of which of the two strategies presented in Section V we employ, the process boils down to either assessing the equality of M predictions, where M is a small integer, and comparing the updated integer reliability metric V with a constant threshold ("state-driven" strategy), or comparing the Softmax confidence with a constant threshold ("confidence-driven" strategy). Each of these calculations is performed in a constant time that is negligible compared to an execution time of even a single NN operation. The choice of the strategy, however, impacts the levels that the NN will be approximated with. Thus, in the remainder of the evaluation we examine the mobile DL accuracy and energy efficiency afforded by different approximation adaptation strategies.

C. Adaptation Strategy Evaluation

To answer RQ3, we assess the Mobiprox's ability to deliver energy savings when adaptation according to the strategies developed in Section V is performed in realistic dynamic scenarios. This we evaluate in two mobile DL domains.

1) Human Activity Recognition: We run the MobileNet-V2 dynamically approximable NN on the HAR traces described in Section VI. These traces were collected in a completely different session and by different authors than the original UCI-HAR traces used for the network training. We evaluated the adaptation strategies from Section V and compared the results with the ones obtained by the nonapproximated MobileNet-V2 (Table III). For each strategy we choose the

TABLE III
INFERENCE ACCURACY AND ENERGY CONSUMPTION ON THE HAR TRACES FROM [36] FOR MOBILENET-V2 TRAINED ON THE UCI-HAR DATA SET

Adaptation	Incr.	Accuracy	Relative Energy
Non-approximated	-	0.65	1.0
Confidence-based	Expon.	0.63	0.854
State-based ($V_L = 2$)	Linear	0.63	0.867

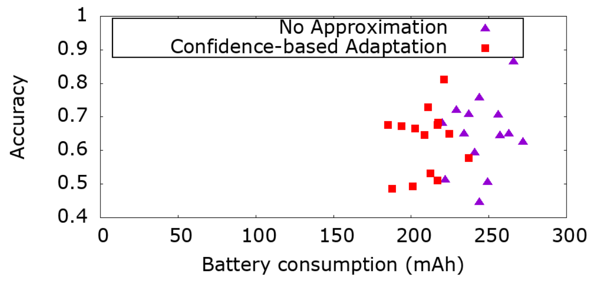


Fig. 6. Average accuracy versus average energy consumption for each user for the nonapproximated network and the confidence-based adaptation.

option for moving to more aggressive approximations (linear versus exponential) that yielded the best results.

These results show that all adaptation engines are more energy efficient than the vanilla MobileNet-V2 with a small drop in average accuracy. The optimal tradeoff between the energy saved and the drop in accuracy is obtained using the Confidence-based adaptation engine, which is 15% more energy efficient with just a 2% drop in overall average accuracy. The accuracy results are modest (the accuracy of the nonapproximated network on the UCI-HAR test set was 90%), which is to be expected given that we used a network trained on a data set collected in one environment for performing human activity inference on data collected in a completely different environment. Thus, the results are more in line with other efforts involving HAR on free-living data for using networks trained on the UCI-HAR data set [40].

Finally, to understand whether the accuracy-energy impact is uniform across the users, in Fig. 6 we show the average accuracy versus average energy consumption for each user trace for both the nonapproximated network and the approximated network using the confidence-based adaptation engine. There is a general trend in the reduction of the energy consumption while maintaining the comparable classification accuracy.

2) Spoken Keyword Recognition: Mobiprox approximation strategies are not restricted to a particular domain. Thus, we also demonstrate approximation adaptation of a SKR DL model. Understanding voice commands is a critical affordance in many ubiquitous computing settings, such as for providing driving assistance or smart home functionalities.

From a number of DL models have been crafted for SKR we focus on CNN-based models introduced by Sainath and Parada [41]. The family of models presented in this work is light-weight, both in terms of memory usage and computation requirements, thus, well-suited for mobile devices. We adopt a particular PyTorch implementation of a model from this family consisting of two convolutional layers and one fully

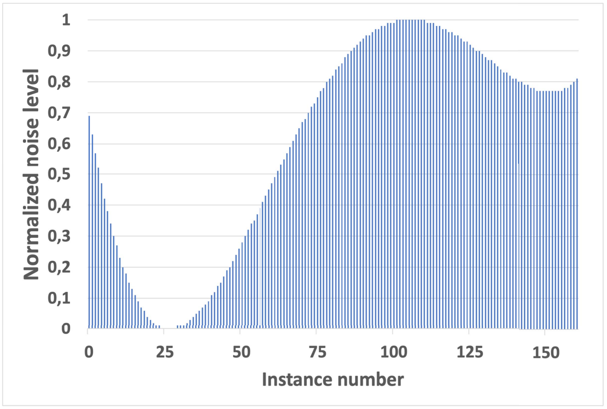


Fig. 7. Noise distribution for the SKR trace: 160 noise level values, one for each sample in the trace. Each sample contains one random utterance (from the 12 classes) on which noise is added with the level specified according to the distribution of the ambient noise during a regular day [43].

connected layer with the Softmax output presented in [38]. The code accompanying [38] already contains the DL model weights obtained through training on the 80% of the GSC data set (validation on 10%), and we reuse these weights in our model. This model is then funneled to Mobiprox's on-server and later on-device tuning on the ASUS Tinkerboard S to obtain a 10-point Pareto front of approximate configurations of the network. For tuning we use a half of the 10% of the GSC that was not used for the training/validation.

Opportunities for dynamic approximation in SKR come with a naturally varying level of background noise. For instance, it has been show that when different levels of noise are present, a different complexity of a DL model is needed to successfully recognize spoken keywords [42]. In our experiments we examine how the adaptation strategies developed in Section V cope with time-varying noise levels. For this, we first construct a trace consisting of 160 word utterances from a previously unseen part of the GSC data set mixed with time-varying white noise whose level corresponds to the level measured in a realistic environment over 24 h [43] (Fig. 7). The trace is then used for SKR with our mobile DL model, while a Mobiprox's adaptation strategy decides on which approximation configuration to use at subsequent inference step. Unlike in the HAR experiment, in this experiment there is no notion of "state" (e.g., a period of time during which a user is likely to keep performing the same activity), rather, keywords are randomly distributed in the trace. Thus, we do not evaluate the state-based adaptation method, but focus on the confidence-based adaptation.

We run the above trace on ASUS Tinkerboard S connected to a Monsoon power meter. We run both the original compact network from [38] and the same network dynamically approximated with two flavors of our confidence-based adaptation scheme (with a linear and an exponential increase in approximation level). The results are shown in Table IV. Both the original network and the two flavors of the approximated network achieve the same accuracy 96.3%, while Mobiprox adaptation leads to 15% system-wide energy savings.

TABLE IV
INFERENCE ACCURACY AND ENERGY CONSUMPTION ON THE SKR TASK FOR A NETWORK INTRODUCED IN [38]

Adaptation	Incr.	Accuracy	Relative Energy
Non-approximated	-	0.96	1.0
Confidence-based	Expon.	0.96	0.852
Confidence-based	Linear	0.96	0.852

TABLE V
AVERAGE RELATIVE ACCURACY (AGREEMENT WITH BASELINE) AND RELATIVE ENERGY CONSUMPTION ON TEN USER TRACES COLLECTED IN AN UNSCRIPTED SCENARIO FOR MOBILENET-V2 TRAINED ON THE UCI-HAR DATA SET. STANDARD DEVIATIONS ACROSS USERS ARE IN PARENTHESIS

Adaptation	Incr.	Agreement w. baseline	Rel. energy
Confidence-based	Expon.	0.83 (0.04)	0.85 (0.01)
State-based ($V_L = 2$)	Linear	0.91 (0.02)	0.88 (0.01)

D. Smartphone-Based Adaptation in Unscripted Scenario

We investigate how Mobiprox performs on a battery-powered commodity Android phone when the scenario of use is not prescribed. On all ten phones Mobiprox successfully ran real-time adaptation and the inference of `mobilenet_uci-har` for HAR. In addition, we reran the collected sensor traces on the same phone with both state-based and confidence-based approximation strategy employed. Analyzing the logs we have not observed any discrepancies (in terms of inference delay or inferred class mismatch) between on-device sampling and inference, and trace-based inference, confirming that Mobiprox affords smooth real-time approximation of mobile DL.

In Table V we compare the inference performance and energy savings for the confidence-based adaptation engine with exponential increase of approximation and the state-driven adaptation strategy with the linear increase. Since the experiments were unscripted and we do not have the ground truth labels, we show the relative accuracy (i.e., the agreement with the nonapproximated baseline model) and relative energy consumption (i.e., compared to the consumption of the nonapproximated model). From the table we observe that the state-based adaptation engine achieves a higher average agreement with the baseline nonapproximated model—91%, while consuming 12% less energy than the baseline. The confidence-based engine allows for more energy savings—up to 15%—but with the downside of a lower agreement with the nonapproximated network—83%.

To further understand the functioning of the approximation adaptation strategy, in Fig. 8 we show an example of the adaptation timeline for one of the user traces collected in this experiment. The black dotted line presents the adaptation, where the higher approximation configuration number (right y-axis) indicates a higher level of approximation. We compare the activities inferred (left y-axis) by the baseline nonapproximated model (green dots) with the activities inferred by the currently used approximation configuration. Should these differ, we plot a red triangle indicating the mismatched activity inferences. Finally, we show the cumulative energy savings (compared to the nonapproximated model and normalized to the graph dimensions) extrapolated from the currently used

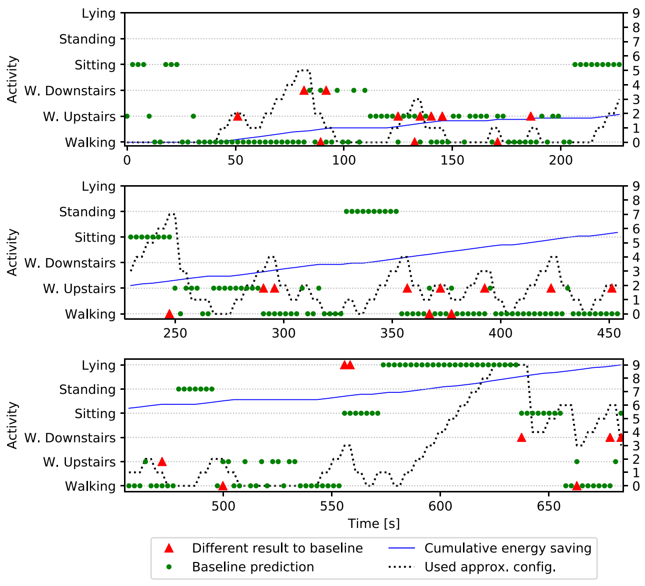


Fig. 8. State-driven adaptation timeline with linear increase of approximation.

configuration and the precise energy measurements from the ASUS Tinkerboard.

The figure confirms that the Mobiprox adaptation strategy harnesses the *accuracy-energy consumption* tradeoff points determined during the tuning phase. The state-driven strategy remains cautious (i.e., uses more accurate configurations) when a user performs dynamic activities that are more difficult to classify (e.g., walking and walking up or down the stairs), thus when long periods of uniform classification results are not present. The strategy jumps to more aggressive approximation configurations when a user lingers in an easier-to-classify static activities (e.g., standing, lying, and sitting). In terms of the mismatches with the baseline, we observe that the differences are often in individual timesteps (i.e., they agree soon again) and that diverging prediction are between similar activities (walk upstairs versus walking and sitting versus lying).

VIII. Discussion and Limitations

Static training-time optimization is stifling further proliferation of mobile DL. Mobiprox builds upon the existing efforts toward dynamic DL optimization [11], [12], [13], [27], yet differs from them in three important ways: 1) being implemented at the compiler-level and exposed as an end-to-end pipeline, Mobiprox is not limited to a particular network architecture, supports various layer types present in NNs, supports models written in either PyTorch or Keras, and does not require any previous involvement from a data scientist designing and training the network; 2) Mobiprox supports quantization, perforated convolutions, and filter sampling, but is created to be easily extensible to a wide range of approximation techniques; and 3) Mobiprox allows context-dependent runtime adaptation of the approximation level; while we include two adaptation algorithms with the Mobiprox codebase, virtually any policy can be used. To facilitate future research and usage, we release Mobiprox as open-source software.[7]

Mobiprox, is subject to certain limitations. First, despite implementing an on-device profiler to better gauge the effect of different approximations on the QoS loss, Mobiprox cannot provide guarantees that the expected QoS will indeed be achieved, nor can it predict the maximal expected QoS loss on yet-to-be-seen data. Somewhat related is the issue of potentially reduced reliability of approximated models. While compression can, in certain situations, improve the generalizability of a model [44], different compression levels can lead to widely varying reliability outcomes [45].

Second, our measurements show the maximum speedup Mobiprox achieves on a mobile device remains relatively modest at 1.25×, while the same NN architecture achieves twice the speedup on a server. This discrepancy likely stems from the lack of optimized support for running (approximate) DL on mobile devices. The goal of the prototype version of Mobiprox presented in this article is to, for the first time, demonstrate dynamic mobile DL approximation adaptation. To unlock further benefits, we plan to examine integration with mobile DL compiler stacks that are already hand-optimized by large engineering teams in production environments, such as TVM [46], Pytorch Mobile, or TF Lite.[8] Since the approximations in Mobiprox reduce both the number of compute operations and memory loads and stores, the performance improvements of these approximations should seamlessly translate, if efficient library and compiler implementations listed above are used. Not only would this likely lead to improved speedup gains, but would also ameliorate the need for time-inefficient development of custom approximable tensor runtimes for various architectures.

Third, Mobiprox is general and can be applied to any NN architecture, yet, the richness of the approximate configurations and the efficiency of the approximation depends on the presence of convolutional layers in the network. Mobiprox specifically targets these layers with convolution perforation and filter sampling approximations, as convolutional layers tend to consume the majority of computational time and energy in mobile NNs [47]. For nonconvolutional layers, Mobiprox allows only one type of approximation—half-precision quantization—leading to at most 2^L possible approximation configurations in an L-layer network. To expand the range of approximation techniques, in future we plan to investigate the integration of dynamic pruning [13] of fully connected layers in Mobiprox. Techniques requiring up-front modification or specialized training to support approximation, such as SNNs [12], remain unsuitable, as with Mobiprox we provide a service that allows the integration of prebuilt networks oblivious to approximation, for instance, those acquired through Google Cloud AutoML, into mobile apps.

[7]https://gitlab.fri.uni-lj.si/lrk/mobiprox/

[8]This is also the key reason why a direct comparison between Mobiprox and current mobile DL compression implementations (e.g., quantization in TFLite) is impossible—neither do these approaches provide dynamic approximation adaptation, nor is Mobiprox optimized for performance.

Finally, with respect to Mobiprox's adaptation algorithms (Section V), these were developed for commonly encountered situations where the context does not fluctuate rapidly. Such behavior is present in numerous domains, including two examined in the previous section—the HAR where an activity a person is performing often stays the same over a certain time period, and the SKR, where the background noise gradually changes throughout the day. However, harnessing the slow-changing nature of many real-world phenomena, our adaptation strategies may not be suitable for tasks, such as anomaly detection, where sudden changes of the target phenomena are expected [48]. Note that this does not restrict the general domain in which Mobiprox can be applied. Indeed, with minor modifications, the strategies presented in this article can be used for adapting approximation of models built for tracking objects in live video [49], for instance.

IX. Conclusion

In this article, we introduced Mobiprox, to the best of our knowledge, the first end-to-end framework that enables dynamically adaptable, rather than static, approximation of mobile DL. Furthermore, Mobiprox works with arbitrary architectures, even with networks not initially designed with approximation in mind. To accomplishing this, we first implemented low-level support for approximate computing on mobile CPU and GPUs through compiler-level primitives. We then integrated the heterogeneous compilation infrastructure, the approximate configuration search framework, and our novel profiler into an Android-ready end-to-end approximate configuration search, selection, and compilation pipeline. We ran different DL architectures through the pipeline and demonstrated that Mobiprox identifies approximation configurations that enable a tradeoff between inference accuracy and energy consumption. Finally, we implemented approximation adaptation strategies for dynamic selection of energy-preserving DL configurations while ensuring that the quality of the resulting classification is not hurt. Experiments in human activity and SKR domains demonstrate that the adaptation strategies successfully accommodate varying context, reducing the system-wide energy usage by 15% in both domains, while sacrificing only 2% of the accuracy in the HAR domain, and leading to no loss of accuracy in the SKR domain.

References

[1] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *Proc. World Wide Web Conf.*, 2019, pp. 2125–2136. [Online]. Available: https://doi.org/10.1145/3308558.3313591

[2] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[3] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. ICML*, Long Beach, CA, USA, Jun. 2019, pp. 1–11.

[4] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE CVPR*, Salt Lake City, UT, USA, Jun. 2018, pp. 1–9.

[5] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "EmBench: Quantifying performance variations of deep neural networks across modern commodity devices," in *Proc. 3rd Int. Workshop Deep Learn. Mobile Syst. Appl.*, Seoul, South Korea, Jun. 2019, pp. 1–7.

[6] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE CVPR*, Las Vegas, NV, USA, Jun. 2016, pp. 1–9.

[7] W. Niu et al., "PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *Proc. ACM ASPLOS*, Mar. 2020, pp. 907–922.

[8] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," Dec. 2014, *arXiv:1503.02531*.

[9] O. Machidon, T. Fajfar, and V. Pejovic, "Watching the watchers: Resource-efficient mobile video decoding through context-aware resolution adaptation," in *Proc. EAI MobiQuitous*, Nov. 2020, pp. 168–176.

[10] O. Machidon, D. Sluga, and V. Pejović, "Queen Jane approximately: Enabling efficient neural network inference with context-adaptivity," in *Proc. 1st Workshop Mach. Learn. Syst.*, Feb. 2021, pp. 48–54.

[11] H. Yu, H. Li, H. Shi, T. S. Huang, and G. Hua, "Any-precision deep neural networks," *Eur. J. Artif. Intell.*, vol. 1, no. 1, 2020, Art. no. 37686.

[12] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. ICLR*, New Orleans, LA, USA, May 2019, pp. 1–12.

[13] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. NIPS*, Long Beach, CA, USA, Dec. 2017, pp. 1–11.

[14] H. Sharif et al., "ApproxHPVM: A portable compiler IR for accuracy-aware optimizations," in *Proc. ACM Program. Lang.*, vol. 3, 2019, p. 186.

[15] H. Sharif et al., "ApproxTuner: A compiler and runtime system for adaptive approximations," in *Proc. ACM PPoPP*, Jan. 2021, pp. 262–277.

[16] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surveys*, vol. 48, no. 4, pp. 1–33, 2016.

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

[18] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Comput.*, vol. 16, no. 3, pp. 82–88, Jul. 2017.

[19] K. Zhao, A. Jain, and M. Zhao, "Automatic attention pruning: Improving and automating model pruning using attentions," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2023, pp. 10470–10486.

[20] 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. (ECCV)*, 2018, pp. 1–17.

[21] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[22] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. ACM SenSys*, Delft, The Netherlands, Nov. 2017, pp. 1–14.

[23] Q. Jin, L. Yang, and Z. Liao, "AdaBits: Neural network quantization with adaptive bit-widths," in *Proc. IEEE/CVF CVPR*, Jun. 2020, pp. 2143–2153.

[24] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proc. AAAI*, New Orleans, LA, USA, Feb. 2018, pp. 1–9.

[25] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. ACM MobiSys*, Singapore, Jun. 2016, pp. 123–136.

[26] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. IEEE ICPR*, Cancun, Mexico, Dec. 2016, pp. 2464–2469.

[27] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic progressive inference of neural networks over device and cloud," in *Proc. ACM MobiSys*, Jun. 2020, pp. 1–15.

[28] M. Kotsifakou, P. Srivastava, M. D. Sinclair, R. Komuravelli, V. Adve, and S. Adve, "HPVM: Heterogeneous parallel virtual machine," in *Proc. ACM PPoPP*, Vienna, Austria, Feb. 2018, pp. 68–80.

[29] M. Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli, "PerforatedCNNs: Acceleration through elimination of redundant convolutions," in *Proc. NIPS*, Barcelona, Spain, Dec. 2016, pp. 1–9.

[30] C. Nugteren, "CLBlast: A tuned OpenCL BLAS library," in *Proc. Int. Workshop OpenCL*, Oxford, U.K., May 2018, pp. 1–10.

[31] "Android developers user guide." Android Debug Bridge (adb). Accessed: Jan. 30, 2022. [Online]. Available: https://developer.android.com/studio/command-line/adb

[32] R. Jabla, F. Buendía, M. Khemaja, and S. Faiz, "Balancing timing and accuracy requirements in human activity recognition mobile applications," in *Proceedings*, vol. 31, no. 1, p. 15, 2019.

[33] A. Mahmoud et al., "Optimizing selective protection for CNN resilience," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Wuhan, China, Oct. 2021, pp. 127–138.
[34] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proc. ICML*, Aug. 2017, pp. 1–14.
[35] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proc. Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn. (ESANN)*, Bruges, Belgium, Apr. 2013, pp. 1–6.
[36] T. Knez, O. Machidon, and V. Pejović, "Self-adaptive approximate mobile deep learning," *Electronics*, vol. 10, no. 23, p. 2958, 2021.
[37] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*.
[38] R. Tang and J. Lin, "Honk: A PyTorch reimplementation of convolutional neural networks for keyword spotting," 2017, *arXiv:1710.06554*.
[39] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "AsyMo: Scalable and efficient deep-learning inference on asymmetric mobile cpus," in *Proc. ACM MobiCom*, New Orleans, LA, USA, Mar. 2022, pp. 215–228.
[40] F. Cruciani et al., "Feature learning for human activity recognition using convolutional neural networks," *CCF Trans. Pervasive Comput. Interact.*, vol. 2, no. 1, pp. 18–32, 2020.
[41] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Annu. Conf. of Int. Speech Commun. Assoc. (Interspeech)*, Dresden, Germany, Sep. 2015, pp. 1–5.
[42] O. Machidon and V. Pejovic, "Energy-efficient adaptive keyword spotting using slimmable convolutional neural networks," presented at TinyML Summit (Poster Session), San Francisco, CA, USA, Mar. 2022.
[43] G. A. Flamme et al., "Typical noise exposure in daily life," *Int. J. Audiol.*, vol. 51, no. S1, pp. S3–S11, 2012.
[44] C. L. Giles and C. W. Omlin, "Pruning recurrent neural networks for improved generalization performance," *IEEE Trans. Neural Netw.*, vol. 5, no. 5, pp. 848–851, Sep. 1994.
[45] S. Cygert and A. Czyżewski, "Robustness in compressed neural networks for object detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 690–699.
[46] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. USENIX OSDI*, Carlsbad, CA, USA, Oct. 2018, pp. 1–16.
[47] D. Li, X. Wang, and D. Kong, "DeepRebirth: Accelerating deep neural network execution on mobile devices," in *Proc. AAAI Conf. Artif. Intell.*, New Orleans, LA, USA, Feb. 2018.
[48] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surveys*, vol. 54, no. 2, pp. 1–38, 2021.
[49] K. Kang et al., "T-CNN: Tubelets with convolutional neural networks for object detection from videos," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 10, pp. 2896–2907, Oct. 2018.

Matevž Fabjančič received the master's degree from the Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia, in 2021.

He is a member of Cosylab's Medical Services Department, Ljubljana, where he develops software that supports the process of cancer treatment through radiotherapy.

Dr. Fabjančič was awarded the Prešeren Award for Students of the University of Ljubljana for his master's thesis in 2022.

Octavian Machidon received the Ph.D. degree in reconfigurable computing from Transilvania University of Brasov, Braşov, Romania, in 2015.

He is an Assistant Professor with the Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia, where he is currently focused on implementing approximate mobile computing solutions for enabling energy-efficient mobile applications.

Hashim Sharif received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, USA.

He is a Postdoctoral Fellow of Computer Science with the University of Illinois at Urbana-Champaign. He is a Research Scientist with AMD Research, Santa Clara, CA, USA. He works in the areas of compilers and systems for machine learning.

Yifan Zhao is currently pursuing the Ph.D. degree from the University of Illinois at Urbana-Champaign, Champaign, IL, USA.

He works in the area of compilers and systems for machine learning, specifically accuracy-aware optimizations and autotuning techniques for better discovering approximation choices.

Saša Misailović received the Ph.D. degree from Massachusetts Institute of Technology, Cambridge, MA, USA, in Summer 2015.

He is an Associate Professor with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA. His research interests include programming languages, compilers, and software engineering, with an emphasis on improving performance, energy efficiency, and resilience in the face of software errors and approximation opportunities.

Dr. Misailović was recognized with NSF CAREER Award and multiple best paper awards.

Veljko Pejović received the Ph.D. degree in computer science from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 2012.

He is an Associate Professor with the Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia, and a Senior Researcher with the Institute Jožef Stefan, Ljubljana. He was a Research Fellow with the Computer Science Department, University of Birmingham, Birmingham, U.K. His research focuses on resource-efficient mobile systems, human–computer interaction, and cybersecurity in ubiquitous systems.

Dr. Pejović was a recipient of several awards including the Best Paper Nomination at ACM UbiComp, the Best Paper Runner-Up at IEEE Pervasive Computing, and the First Prize at Orange D4D Challenge for his work on epidemics modeling.