SIEVE: Speculative Inference on the Edge with Versatile Exportation

Babak Zamirai, Salar Latifi, Pedram Zamirai, Scott Mahlke
Computer Science and Engineering Department, University of Michigan, Ann Arbor
{zamirai, salar, pedramz, mahlke}@umich.edu

***Abstract*—This paper proposes *SIEVE*, Speculative Inference on the Edge with Versatile Exportation, which dynamically distributes CNN computation between the cloud and edge device based on the input data and environmental conditions to maximize efficiency and performance. A speculative CNN is created through aggressive precision reduction techniques to run most of the inferences on the edge device, while the original CNN is run on the cloud server. A runtime system directs each input to either the edge or cloud and decides whether to accept speculative inferences made on the edge or invoke recovery by replaying the inference on the cloud. Compared to the cloud-only approach, SIEVE reduces energy consumption by an average of 91%, 57% and 26% and increases performance by an average of $12.3\times$, $2.8\times$ and $2.0\times$ for 3G, LTE and WiFi connections without accuracy loss across a range of nine CNNs.**

I. Introduction

Current edge devices are capable of generating huge amounts of high-quality video and images to feed CNNs for various applications. Due to the prohibitive energy and latency of communicating these data to cloud servers, there is a growing trend towards CNN execution on edge devices themselves. For example, the Qualcomm Snapdragon 835 with machine learning capabilities enables running some trained models directly on the mobile device. However, the size and complexity of CNNs are increasing more rapidly to improve their accuracy and functionality than the hardware capabilities, which results in computations with energy requirements beyond device's battery constraints even with hardware accelerators [1].

To address the shortcomings of cloud-only and mobile-only approaches, two categories of hybrid cloud-edge systems have evolved, which partition the computation between cloud servers and edge devices for maximizing performance and efficiency [2]. First, applications, such as Apple's Siri and Amazon Alexa, consist of specialized speech recognizers for keyword spotting on the device and automatic speech recognition, natural language interpretation, and various information services on the cloud. In these applications, the computation is partitioned manually during the design process, and servers and devices are responsible for different tasks. In the second category, since the energy efficiency and data transfer rates vary for different wireless technologies, edge devices dynamically decide whether to offload and which parts of the computation need to be offloaded to maximize the energy efficiency and performance [3]. For example, MCDNN [4] utilizes Approximate Model Scheduling to manage multiple DNN execution requests under resource constraints. It enables trading off classification accuracy for resource use by reasoning about on-device/cloud execution trade-offs. Another example is Neurosurgeon [5], which dynamically finds the best partitioning point at layer granularity by analyzing the CNN topology and network connection. Then, the computation of the first set of layers is kept on the device, and the remaining layers are offloaded to the cloud.

Our analysis of prior partitioning approaches reveals that they over utilize the cloud because they are input data invariant. The partitioning decision is either hardwired or dynamically changes only in response to the environment, e.g., network connectivity. However, CNNs are usually over provisioned and most of the inputs do not require the entire computational power of the model to produce an accurate final output [6]. Consequently, we hypothesize that efficiency improvements can be achieved through data-dependent partitioning.

We take inspiration from traditional speculation-recovery techniques and present *SIEVE*, Speculative Inference on the Edge with Versatile Exportation, which is data-dependent and dynamically distributes CNN computation between the cloud and device to achieve maximum efficiency and minimum latency in various environments. SIEVE uses aggressive compression to form a small CNN that can speculatively perform inferences for a large fraction of inputs on the device. A runtime system sends each input to either the edge or cloud based on user preferences, input size and environmental conditions. In addition, the runtime system can selectively invoke the original CNN on the cloud server to recover from misspeculation, wherein the speculative CNN can only provide a low-confidence answer.

SIEVE has four main advantages. First, the runtime system automatically adapts how and where inferences are performed to the environment (network conditions and battery lifetime) in order to reduce energy consumption on inferences. Second, relying on the original CNN for recovery enables aggressive precision reduction of the speculative CNN, which increases the opportunities of processing more complex CNNs efficiently on the edge device. Third, the average latency of the system is improved in comparison to both cloud-only and mobile-only approaches because of the elimination of the extra round trips to the server and the excessive complexity of CNNs for marginal accuracy improvements. Last, not sending requests to the server for all inputs reduces the load on the server and releases some resources to support more users on the cloud.

The contributions of this paper are as following:

- We propose an intelligent hybrid cloud-edge CNN computation partitioning technique to achieve efficiency and performance gains. A lightweight software runtime is designed to dynamically select between speculative inference on the edge or inference on the server using the original CNN. It models the latency and energy of the speculative and original CNNs, available hardware on the edge device, data transfer latency, and speculative CNN accuracy and its associated misspeculation recovery costs.
- We provide a misspeculation detector that dynamically determines a confidence threshold on the speculative CNN output based on the distribution of correctable errors learned during the training phase and the target output quality to detect potential faulty outputs, which are sent to the original CNN for replay.
- We develop an automatic procedure to deterministically prune the design space, and then search for the most compressed floating-point format for weights and activations of individual layers. Heuristics are used to combine layers to form a speculative CNN with maximum compression and minimum accuracy loss, while not requiring any fine-tuning or retraining.
- We also introduce lightweight (0.05% area overhead) hardware compressor and decompressor units responsible for efficient floating-point reformatting for the speculative CNN.

On a benchmark suite of nine CNNs, SIEVE on average reduces mobile energy consumption by 91%, 57%, and 26% and improves

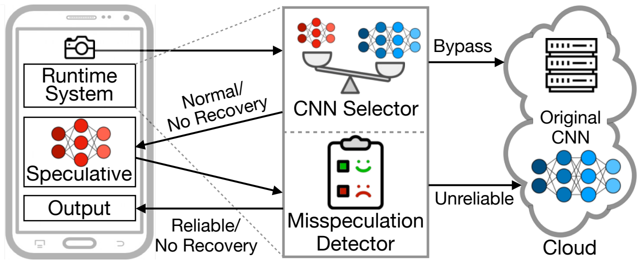


Fig. 1: Overview of SIEVE. Camera provides input to the runtime system to choose between the speculative CNN on the device and the original CNN on the cloud. The misspeculation detector examines speculative CNN outputs and triggers recovery for faulty ones.

latency by 12.3×, 2.8×, and 2.0× for 3G, LTE and WiFi connections in comparison to the cloud-only approach, without any accuracy loss.

II. Background and Motivation

We measured the energy and latency of AlexNet inference on the NVIDIA Jetson TK1 mobile platform for mobile-only and cloud-only approaches, considering three different network connections: 3G, LTE and WiFi. We found that the cloud-only result is heavily dependent on the type of the wireless network. We had two key observations. First, the mobile-only results in lower end-to-end latency for all three types of connections. Second, although the mobile-only consumes less energy than transferring data via LTE or 3G, the cloud-only, when using WiFi, is the most energy-efficient approach. Therefore, previous work [5] proposes dynamic CNN computation partitioning at layer granularity by analyzing the CNN topology, computation, and communication characteristics to improve energy efficiency and performance. However, the proposed hybrid techniques are data invariant and do not change the partitioning decision per input when the CNN topology and wireless network remain unchanged.

III. SIEVE

To elevate CNN computation partitioning, we take inspiration from traditional speculation-recovery techniques and propose *SIEVE* which is an intelligent hybrid deep learning cloud-edge system for improving energy efficiency and performance in comparison to running complex CNNs on edge devices or offloading the entire computation to cloud. First, we introduce our system and a new approach for dividing the labor between the server and mobile. After that, each component of the runtime system is described.

A. System Overview

SIEVE is based on the hypothesis that inference energy on edge devices can be substantially reduced with an introspective software system that is both data and environmentally aware. As reported in prior work, the availability of high speed networks have a large impact on whether offloading to the cloud is feasible and efficient [5]. But, we also believe the characteristics of individual inputs play an important role in how and where inference should be performed. A simpler, lighter weight CNN is often capable of rendering accurate inferences for a significant fraction of the inputs. Thus, an intelligent runtime system can examine the characteristics of the data to help partition the computation. SIEVE is designed to partition computation between a lightweight version of the CNN that runs on the edge device (speculative inference), and the original CNN on the cloud, and to detect misspeculations on the edge device that must be replayed in the cloud. By supporting data-aware partitioning, SIEVE enables a new level of efficiency gains that are not possible with environmental-only partitioning methods [5].

Figure 1 shows the major parts of SIEVE including a camera, runtime system, mobile hardware and cloud server. At the beginning, the camera on the mobile device provides the input to the runtime system, which consists of a CNN selector and a misspeculation detector. Then, the CNN selector evaluates user preferences (accuracy requirements and privacy) and environment (network speed, available hardware, and server load) to dynamically route each input through either the original CNN on the cloud or the speculative CNN on the device. The main purpose of CNN selection is to minimize the energy and latency while meeting the accuracy requirements of the user. After that, if the speculative CNN is selected, its outputs are sent to the misspeculation detector unit to identify untrustworthy answers and initiate replay on the cloud. When the original CNN is selected, SIEVE bypasses the speculative CNN and misspeculation detector and sends the inputs to the cloud and downloads the results as with traditional cloud offloading.

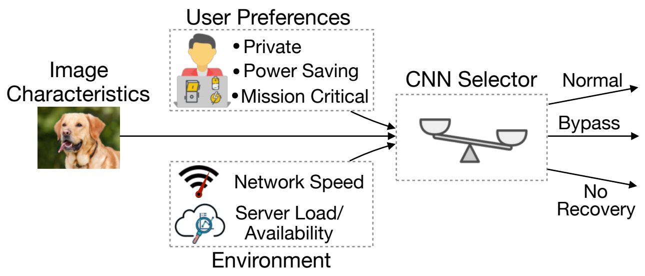


Fig. 2: CNN selector decides based on the input, user preferences and environmental conditions.

SIEVE produces energy savings in two ways. First, successful speculation on the edge device enables a lighter weight CNN to perform the inference and avoid data transfer costs. Second, direct inference on the cloud because data transfer is actually cheaper than edge inference and predicted misspeculation costs. Conversely, SIEVE requires more energy during misspeculations as inferences must occur on both the edge (speculative) and cloud (replay). Thus, it is important to manage speculation carefully to ensure that it is profitable. We found in our evaluation that only modest success rates are necessary for speculation to be useful, $> 60\%$ is generally profitable. In our results, we achieved a minimum of 80% success, which was well above the threshold.

Offline Training: SIEVE designs and trains the speculative CNN and configures runtime system units using the topology and parameters of the original CNN as well as characteristics of the mobile hardware, cloud servers, and communication networks.

B. CNN Selection

Figure 2 shows the first major component of SIEVE runtime system, which is the CNN selector. It tries to run most of the inputs speculatively on the device, however, there are various factors that have different effects on the final selection decision. It estimates and compares the efficiency of data communication and local hardware to find the proper computation partitioning. Moreover, it takes into account environmental conditions and regulates the number of requests to the server. In addition, it considers user preferences by providing knobs to enable and disable different components of SIEVE.

Input Data and Environmental Conditions: To make the correct decision, SIEVE must estimate energy consumption and latency of the data transfer to the cloud (E_t, L_t) and execution on the mobile device (E_m, L_m). Additionally, it requires the probability of initiating recomputation by the misspeculation detector (P_r).

SIEVE uses techniques from Huang et al. [7], [8] to estimate E_t and L_t based on the size of the data, communication technology and bandwidth. Since the computation on the cloud GPU is much faster than the mobile GPU, the latency of the computation on the cloud is neglected in the estimation of L_t. In addition, it trains a regression

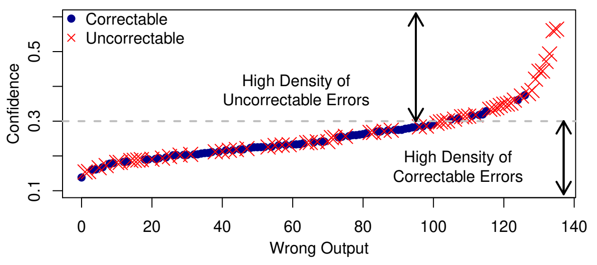


Fig. 3: Confidence of wrong outputs for 4-bit compressed LeNet-5 on MNIST test set. For low confidences, density of correctable errors is higher than uncorrectable ones.

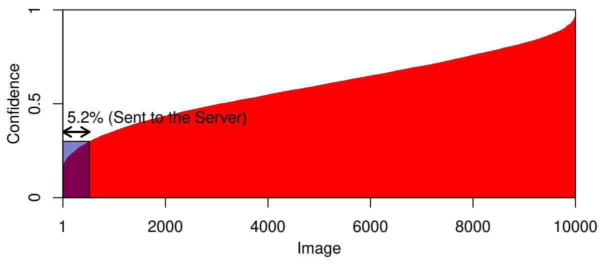


Fig. 4: Confidence of 4-bit compressed LeNet-5 on MNIST test set. The confidence threshold for indicating correctable errors results in limited number of recomputations.

model, same as Neurosurgeon [5], to estimate E_m and L_m based on the number and size of input and output feature maps as well as the characteristics of each layer including layer type, stride, kernel and group size for convolutional and pooling layers. These models are CNN invariant. Hence they are trained once for a set of CNNs and wireless connections, then they could be used for different CNN applications during runtime. P_r is calculated by comparing the output difference of the speculative CNN and original CNN on the validation set, which is explained in more details in Section III-C.

During runtime, it compares E_t (L_t) and $E_m + E_t \times P_r$ ($L_m + L_t \times P_r$) to pick between the original and speculative CNNs for minimum energy (latency). In addition, if the difference of these two numbers becomes larger than a defined threshold in a way that all inputs are sent to the original CNN, it concludes that the speculative CNN is not proper for the current wireless network speed and replaces it with another available topology.

Furthermore, it measures the round-trip time for each request and compares it with L_t estimation. A big difference between these two numbers indicates a high load on the server. Consequently, SIEVE temporarily turns off the misspeculation detector to eliminate recovery requests and keep the computation on the device. At the same time, it monitors the load on the server to find an opportunity to get back to its normal operation. This mechanism sacrifices some accuracy to maintain the constraints of time-sensitive applications when the server response time is low.

User Preferences: In addition, CNN selector considers user preferences and provides a knob for the user to switch among mission critical, power saving, private and normal mode dynamically. The mission critical mode disables the speculative CNN and misspeculation detector and offloads every input on the cloud to maximize the accuracy and robustness. On the other hand, the private mode keeps the entire computation on the device and prevents any offloading to the cloud to increase the privacy and security. And, the power saving mode relaxes the speculative CNN and the misspeculation detector to sacrifice a marginal amount of accuracy to obtain further energy gains. Finally, the normal mode leaves the other units unchanged.

C. Misspeculation Detection

The second major component of the SIEVE runtime system is the misspeculation detector. Since the speculative CNN is a simplified version of the original CNN and trades off some accuracy for better efficiency, it is necessary to detect and recover errors to bridge this accuracy gap to achieve the same accuracy as the baseline.

For classification tasks, CNNs convert an input instance (x) to an output vector of K elements (K is the number of classes). Using a softmax ($\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$) activation function after the output layer provides the estimated probability of that the correct output is j for $j = 1, ..., K$ ($P_j = P(y = j|x)$). Consequently, $\max\limits_{1 \leqslant j \leqslant K} P_j$ represents the confidence level of the CNN in the final output.

Figure 3 illustrates the confidence of wrong outputs for LeNet-5 CNN, compressed to 4 bits per weights and activations, on MNIST test set for handwritten digit recognition. The circles represent correctable errors, which are wrong on the compressed CNN but correct on the original CNN. And, the crosses indicate uncorrectable errors, which are wrong on both CNNs. If we pick a threshold equal to 0.3, shown by the dashed line, the density of correctable errors is much higher than the uncorrectable ones under that threshold.

On the other hand, Figure 4 shows the confidence of the same CNN on the entire test set, and all 10000 images of the dataset are sorted from low to high confidence. As shown, only 5.2% of the images result in outputs with less than 0.3 confidence. Hence, it is possible to recover most of the correctable errors by initiating the recomputation on the server for a small portion of inputs. In addition, the remaining correctable errors, which could not be recovered by this mechanism, will be compensated by inputs that are classified correctly by the compressed CNN but misclassified by the original one.

SIEVE determines the confidence distributions of the speculative CNN for correct, and correctable and uncorrectable wrong outputs on the validation set during training. During runtime, the misspeculation detector defines a confidence threshold by examining those distributions and considering the output target quality to minimize the number of recomputations. Then, it compares the confidence of the speculative CNN with that threshold for each inference to detect and recover possible errors and meet the output target quality.

IV. Speculative CNN Design

An important component of SIEVE is deriving a lightweight speculative CNN from the original CNN and making sure it performs faster and more efficiently than the mobile-only and cloud-only approaches. There are various well-studied methods for reducing the computation and memory accesses of CNNs for small or no accuracy reductions, such as pruning [1], compression [9], fixed-point computation [10] and precision reduction [11]. Although all these techniques could be employed to design the speculative CNN, we prefer to minimize in-depth hardware modifications, rigorous CNN modifications and fine-tuning. Hence, our system combines compression and custom floating-point representation techniques to design and implement an efficient speculative CNN on the mobile. To keep hardware modifications at the minimum level of complexity while gaining considerable improvements, SIEVE takes advantage of custom floating-point formats for compressing weights and activations stored in the memory, but performs the computation in full precision.

There are three advantages for this approach. First, since the numbers will be reformatted to full precision before the functional unit, it is not necessary to use the same format for all numbers in the memory. Hence, weights and activations of different layers could be compressed differently based on their required precision as explained

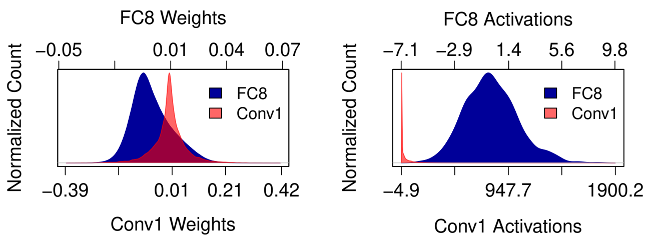


Fig. 5: Distribution of activations and weights in AlexNet. Each layer requires its specific optimized representation.

in prior work [12]. Second, the compression and decompression are basically reformatting between two different floating-point representations. Therefore, the hardware compressor and decompressor become very cheap and easy to implement. Third, since the computation is precise, each layer could be compressed very aggressively to minimize the memory accesses. In this section, first the opportunities for compressing CNNs are investigated. Then, we explain how SIEVE prunes the design space and derives the speculative CNN from the original CNN. Lastly, we describe the hardware design.

A. Compression

Conventional processors usually use IEEE 754 32-bit base-2 floating-point variables for CNN applications. These variables consist of three parts: one sign (s) bit, eight exponent (e) and 23 mantissa (m) bits. In addition, the exponent uses a bias (b) equal to +127 to represent the range from -126 to +127 by unsigned integer format (0 and 255 are interpreted specially). The value of each number is computed as $(-1)^s \times 2^{(e-b)} \times (1 + \sum_{i=1}^{23} m_{23-i} \times 2^{-i})$. Therefore, this format covers the huge range from -3.4×10^{38} to $+3.4 \times 10^{38}$.

However, the range of numbers even in very deep NNs is much smaller than the range provided by single-precision floating-points. Figure 5 shows the distribution of weights and activations for the first convolutional (Conv1) and last fully-connected (FC8) layer of AlexNet. The comparison of these four distributions results in three important observations. First, the 32-bit floating-point is not the most suitable format to represent CNN numbers efficiently. Second, the range of numbers in various layers is very different from each other. For instance, the range of weights in Conv1 is about 7 times bigger than FC8. Third, the weights and activations of the same layer have very different distributions. In conclusion, it is necessary to pick the most representative format for weights and activations of each layer separately to maximize the compression rate.

It is very time consuming to sweep the entire design space and find the optimum number of bits for weights and activation of layers separately. The exponent is the part that determines the range of changes. Hence, defining the proper bitwidth for exponents is vital for accuracy maintenance. Figure 6 shows the histogram of weight exponents for FC8 of AlexNet, which change from -4 to -32. However, more than 99.5% of exponents are between -5 and -14. Due to the natural error resiliency of CNNs [11], we can easily omit the outliers and reduce the bitwidth e_{bw} from 8 to 5, which covers the range between -14 to +15. Yet there is no need to represent numbers higher than -4. To maximize the compression, SIEVE uses bias adjustment. Since the histogram is not symmetric around zero, a bias equal to $2^{e_{bw}-1}-1$ is wasteful. The optimum bias is determined by $b = 1 - \min|e|$. Then, there will be more room for bitwidth reduction, $e_{bw} = \lceil \log_2(\max|e| - b - 2) \rceil$. For FC8, adjusting the bias to 15 reduces e_{bw} to 4 bits for a final range of -1 to -14.

To pick the best representation for each layer, SIEVE runs the original CNN on the validation set to gather exponent histograms. Then, for each set of numbers, it finds the best e_{bw}, called e_{max}, and $bias$ deterministically. Next, it sweeps e_{bw} from e_{max} to 2 bits,

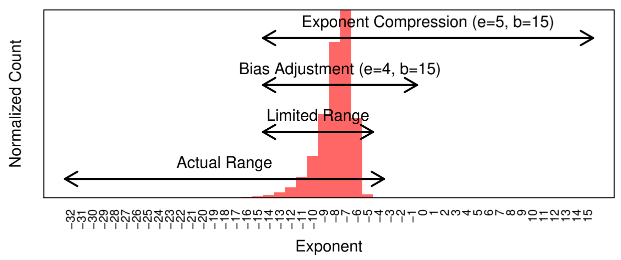


Fig. 6: Exponent histogram for AlexNet FC8 weights. Bias adjustment reduces the bitwidth from 8 to 4.

computes the $bias$ that maximizes the coverage and stops whenever the coverage is below 50%. For each iteration, it sweeps the m_{bw} from 6 to 0 bits, tests the CNN on the validation set while the other weights and activations are using full precision, and stops whenever the normalized accuracy is below 99.8%. Since the search space is considerably pruned, and there is no need for retraining, the design space exploration is fast enough to derive the speculative CNN in a few hours on a single GPU even for huge CNNs on large datasets.

After that, our system picks the best configuration for each layer and finalizes the speculative CNN design. Each explored CNN has a compressed layer and is a trade-off between efficiency and accuracy. It is important to note that even combining layers of two 100% accurate compressed CNNs will not necessarily result in a 100% accurate CNN with two layers compressed, because of the aggregation of precision losses.

SIEVE constructs three CNNs for different environmental conditions and user preferences: exact, accurate and approximate. The exact CNN contains the most accurate layers from the explored CNNs. The accurate CNN, contains the most compressed layers from the explored CNNs with accuracy not less than 100%. And the approximate CNN is same as the accurate CNN, but with threshold of 99.9%. If there is no explored CNN to meet the threshold for a layer, that layer will use the most accurate explored configuration.

Each CNN represents a trade-off between the number of correctable errors relative to the uncorrectable ones and the percentage of inputs that need to be recomputed on the server. For example, the approximate speculative CNN is suitable for the situations when the wireless communication is efficient, so we can make the speculative CNN as compressed as possible by increasing the recovery frequency. The CNN selector uses latency and energy models to convert these trade-offs to a single Pareto frontier based on the wireless communication speed. This Pareto helps the CNN selector to pick the most efficient speculative CNN which meets the output quality target.

B. Hardware

Prior work [13] identifies that the key bottleneck for CNN execution on GPUs is the on-chip memory bandwidth. Hence, SIEVE keeps the weights and activations of the speculative CNN compressed in the entire memory hierarchy, and decompresses them right before writing to the register file for computation. Since on-chip memory data compression at this level requires very frequent data reformatting, a low-overhead hardware implementation of the compression/decompression mechanism is mandatory. Compressor and decompressor are located before register file to keep the data in the entire memory hierarchy compressed. The compressor (decompressor) unit contains the same number of compressor (decompressor) engines as the number of load/store units in the SM. And, each compressor (decompressor) engine is capable of processing one compressed (decompressed) value per cycle.

Compressor: Figure 7 shows the implementation of a compressor engine in details. This engine is responsible for both bitwidth

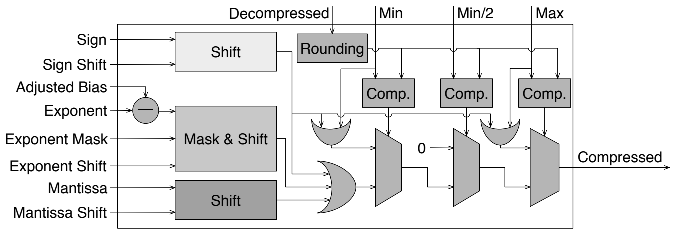


Fig. 7: Compressor engine.

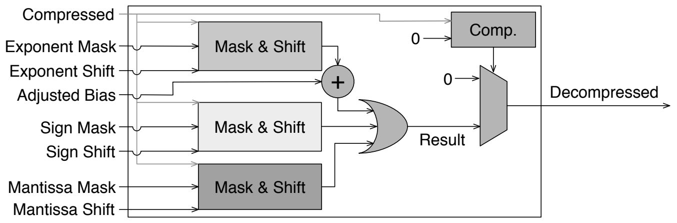


Fig. 8: Decompressor engine.

reduction and bias adjustment. The output format is dynamically configurable to use the same engine for different layers of the speculative CNN. In addition to the 32-bit decompressed number, it is fed by several constant values including Min, Min/2, Max, Sign Shift, Adjusted Bias, Exponent Mask, Exponent Shift and Mantissa Shift. All constants are computed based on the target compression format by a constant calculator, which is shared among all compressors.

Decompressor: Figure 8 demonstrates the hardware design of a decompressor engine in the same way as the compressor engine.

To measure the overhead, we implemented and synthesized the compressor and decompressor unit for an NVIDIA Tegra K1 mobile processor using the ARM Artisan IBM SOI 45 nm library. It has an area overhead of $0.06mm^2$ (0.05%), and an active power consumption of 0.23W (2.06%). Moreover, the compression/decompression takes less than one clock cycle and could be implemented as an additional pipeline stage. Considering the low branch misprediction rate of CNN computation, the performance overhead is negligible.

V. Evaluation

Benchmarks: We evaluate SIEVE using a benchmark suite of nine CNNs using three wireless connection technologies: 3G, LTE and WiFi. A brief description of each benchmark is presented in Table I. We use a randomly sampled subset of the training set as validation set to configure our system (same size as the test set). Third column contains the top-1 accuracy of the trained original CNNs obtained from Caffe Model Zoo. Fourth column shows the input size of each CNN, which is the amount of data to be sent to the cloud in the case of cloud-only or recovery. And the last column is the number of floating-point multiply-accumulate (MAC) operations required to process a single input, which is a good indicator of complexity.

Hardware platforms: The NVIDIA Jetson TK1 embedded development kit is used as our mobile platform. It is built around NVIDIA Tegra K1 SoC: a quad-core ARM A15, a Kepler mobile GPU with a single streaming multiprocessor and a 2 GB DDR3L memory. In addition, for the server side, we use an NVIDIA TITAN X GPU.

Software framework: We use Caffe, an open-source deep learning library, and cuDNN, NVIDIA's GPU-accelerated library for DNNs, to build and test SIEVE.

SIEVE could be configured to prioritize efficiency (performance) improvements for maximum gains, while still obtains latency (energy) savings. To demonstrate the supremacy of SIEVE, we compare it with both cloud-only and mobile-only.

TABLE I: CNN information for different benchmarks.

Network	**Dataset**	**Acc.**	**Input Size**	**MACs (G)**
LeNet-5	MNIST	99.19%	$1 \times 28 \times 28$	0.002
ConvNet	CIFAR10	82.12%	$3 \times 32 \times 32$	0.01
NIN-CIFAR10	CIFAR10	89.57%	$3 \times 32 \times 32$	0.2
AlexNet	ImageNet	56.90%	$3 \times 227 \times 227$	0.7
SqueezeNet_1.0	ImageNet	57.67%	$3 \times 227 \times 227$	0.7
NIN-ImageNet	ImageNet	56.34%	$3 \times 224 \times 224$	1.1
GoogLeNet	ImageNet	68.92%	$3 \times 224 \times 224$	1.6
ResNet-18	ImageNet	66.62%	$3 \times 224 \times 224$	1.8
VGG-16	ImageNet	68.35%	$3 \times 224 \times 224$	15.5

A. Energy Saving

Figure 9 shows the energy consumption of SIEVE compared to the mobile-only and the cloud-only methods for different wireless connection speeds. In each plot, each group of bars is dedicated to one of the nine benchmarks. In each group, the left bar represents the result of running the original CNN on the mobile (mobile-only) and the right bar is dedicated to the energy breakdown of SIEVE. Both bars are normalized by the cloud-only energy consumption result. The bottom part of right bars is related to the amount of energy consumed for running the speculative CNN on the mobile device. The middle part is dedicated to uploading the input data to the server and downloading the final result for two situations: when the CNN selector predicts that the data network is more efficient than the speculative CNN, or when the misspeculation detector triggers the recovery phase. The top shows the energy consumption of the decoder and encoder (reformatting).

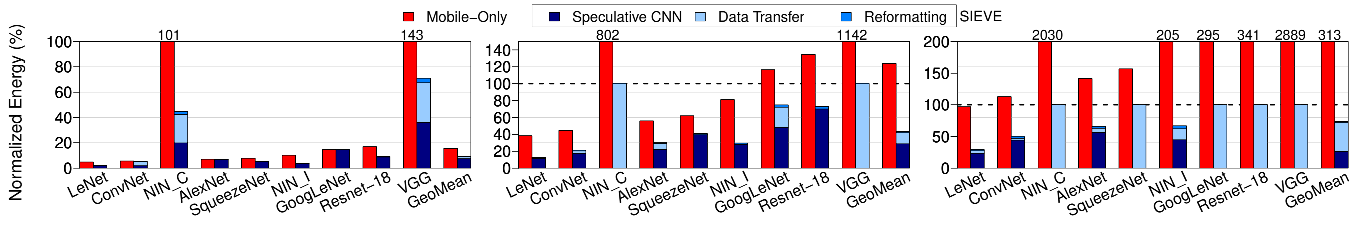
On average, SIEVE reduces the energy consumption of the cloud-only approach by 91%, 57% and 26% for 3G, LTE and WiFi, respectively. In contrast, the mobile-only approach increases the energy consumption by 24% and 213% in comparison to when the cloud-only method uses LTE and WiFi connection, respectively. In addition, for 3G connection, SIEVE results in 7% more energy savings than the mobile-only.

SIEVE is capable of decreasing the energy of the cloud-only, except when the complexity of the baseline CNN is much higher than the input dimensions. For example, based on Table I, the number of MACs per input size for NIN_CIFAR10, ResNet-18, GoogLeNet and VGG-16 is one and two orders of magnitude higher than other benchmarks. Hence, the complexity of the baseline CNN for these four benchmarks is much higher than the input dimensions. Consequently, when network connection is efficient (WiFi), CNN selector correctly predicts that the data transfer over the network for these benchmarks is more efficient than executing on the device.

B. Latency Improvement

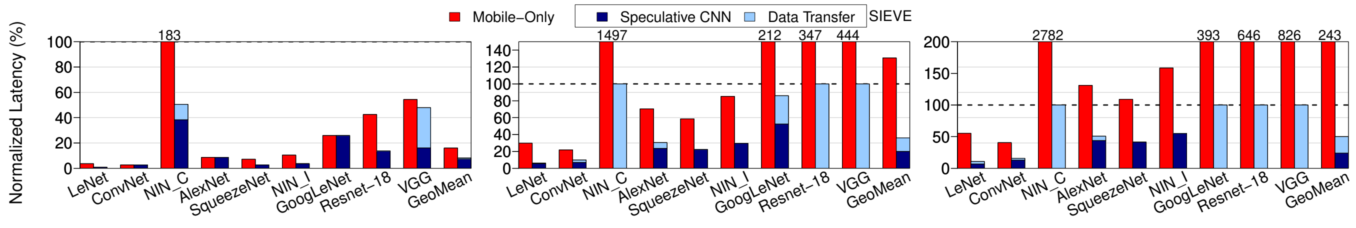
As explained in Section V-A, another approach to take advantage of SIEVE is configuring it for latency improvements. Figure 10 shows the latency of SIEVE versus the mobile-only approach normalized by the end-to-end latency of the cloud-only technique in the same format as Figure 9. Each right bar has two partitions. The bottom part indicates the latency of executing the speculative CNN on the device. And, the top part is dedicated to the end-to-end latency of sending one input to the cloud, processing that on the cloud GPU and downloading the final result for the cases that CNN selector or misspeculation detector activates the original CNN. On average, SIEVE improves the latency by 12.3×, 2.8× and 2.0× for 3G, LTE and WiFi, respectively.

SIEVE managed to reduce the latency for all benchmarks in comparison to the mobile-only approach. On average, the mobile-only increases the latency by 31% and 143% for LTE and WiFi, respectively. Even for 3G, SIEVE achieves 8% more latency reduction.



(a) 3G (b) LTE (c) WiFi

Fig. 9: Energy consumption of SIEVE vs. the mobile-only approach for 3G, LTE and WiFi normalized by cloud-only results.



(a) 3G (b) LTE (c) WiFi

Fig. 10: Latency of SIEVE vs. the mobile-only approach for 3G, LTE and WiFi normalized by cloud-only results.

C. Comparison to Prior Work

In this section, we compare energy of SIEVE with other techniques using WiFi connection.

Neurosurgeon [5] dynamically finds the interesting partitioning points within a CNN at layer granularity to reduce the data transfer to cloud by pushing as much computation as possible onto the mobile device. On average, SIEVE outperforms Neurosurgeon by 20% more energy improvements using data-dependent dynamic partitioning.

DeftNN [13] proposes near-compute data fission to scale down the on-chip data movement requirements by efficiently packing on-chip memory. On average, running speculative CNNs designed by SIEVE requires 51% less energy than the CNNs designed by DeftNN. Hence, SIEVE is capable of utilizing more compressed and energy-efficient CNNs by relying on the cloud for recovery.

Scalpel [14] proposes node pruning to reduce computation without sacrificing the dense matrix format. On average, Scalpel reduces energy by 42% on the mobile device, and SIEVE improves this result by 25%. Consequently, SIEVE could use aggressive precision reduction and speculative execution with cloud recovery on top of mobile-only compression techniques, including DeepX [15], Edge AI [16], for further savings.

VI. Conclusion

In this work, we introduce *SIEVE*, a novel *hybrid cloud-edge deep learning system*. It creates a heavily compressed CNN through aggressive precision reduction to enable speculative inferences on the device. A dynamic partitioning technique is employed to push most of the computation to this speculative CNN while data transfer to the original CNN is limited to recovery on low-confidence inferences. Compared to the cloud-only approach, SIEVE reduces the energy consumption by an average of 91%, 57% and 26% and increases the performance by an average of $12.3\times$, $2.8\times$ and $2.0\times$ for 3G, LTE and WiFi connection with 100% accuracy of baseline.

Acknowledgment

We thank our anonymous reviewers for their feedback and suggestions. This work was supported by the National Science Foundation Grant NSF-XPS-1628991.

References

[1] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[2] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[3] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[4] 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 *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 123–136.

[5] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 615–629.

[6] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[7] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of lte: effect of network protocol and application behavior on performance," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 363–374, 2013.

[8] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 225–238.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus." Citeseer.

[11] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[12] A. Delmas, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, and A. Moshovos, "Bit-tactical: Exploiting ineffectual computations in convolutional neural networks: Which, why, and how," *arXiv preprint arXiv:1803.03688*, 2018.

[13] P. Hill, A. Jain, M. Hill, B. Zamirai, C.-H. Hsu, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars, "Deftnn: addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 786–799.

[14] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 548–560.

[15] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*. IEEE, 2016, pp. 1–12.

[16] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, 2019.