Low-Latency Proactive Continuous Vision

Yiming Gan
ygan10@ur.rochester.edu
University of Rochester
Rochester, NY,USA

Yuxian Qiu
qiuyuxian@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Lele Chen
lchen63@cs.rochester.edu
University of Rochester
Rochester, NY,USA

Jingwen Leng
leng-jw@cs.sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Yuhao Zhu
yzhu@rochester.edu
University of Rochester
Rochester, NY,USA

Abstract

Continuous vision is the cornerstone of a diverse range of intelligent applications found on emerging computing platforms such as autonomous machines and Augmented Reality glasses. A critical issue in today's continuous vision systems is their long end-to-end frame latency, which significantly impacts the system agility and user experience. We find that the long latency is fundamentally caused by the serialized execution model of today's continuous vision pipeline, whose key stages, including sensing, imaging, and vision computations, execute sequentially, leading to long frame latency.

This paper seeks to reduce the end-to-end latency of continuous vision tasks. Our key idea is a new *proactive* vision execution model that breaks the sequential execution of the vision pipeline. Specifically, we propose to allow the pipeline front-end (sensing and imaging) to predict future frames; the pipeline back-end (vision algorithms) then predictively operates on the future frames to reduce frame latency. While the proactive execution model is generally applicable to any vision systems, we demonstrate its effectiveness using an implementation on resource-constrained mobile Systems-on-a-chips (SoC). Our system, PVF, incorporates two techniques to overcome key challenges that arise in deploying proactive vision in mobile systems: it *enables multiple outstanding speculative frames* by exploiting the hardware heterogeneities in mobile SoCs; it *reduces the energy overhead of prediction* by exploiting the error-resilient nature of vision algorithms. We show that PVF reduces the frame latency by up to 92% under the same energy.

CCS Concepts

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; **Special purpose systems**.

Keywords

Continuous Vision, Proactive Execution

ACM Reference Format:
Yiming Gan, Yuxian Qiu, Lele Chen, Jingwen Leng, and Yuhao Zhu. 2020. Low-Latency Proactive Continuous Vision. In *Proceedings of the 2020 International Conference on Parallel Architectures and Compilation Techniques (PACT '20), October 3–7, 2020, Virtual Event, GA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3410463.3414650

1 Introduction

Domain specific architectures (DSA) provide more compute capability with lower energy consumption under the same silicon budget. A significant implication of DSAs is the high non-recurring engineering (NRE) cost of designing custom architectures and hardware chips [33]. Therefore, we must identify key application domains whose demands and social impacts are high enough to justify the efforts of designing DSAs. This paper focuses on the domain of *continuous vision*, which processes real-time images from camera sensors to extract visual insights that guide high-level decision making. Continuous vision is key to many emerging applications in both consumer mobile devices and "always-on" embedded systems such as robotics, Augmented Reality, and smart-city sensing.

Today's continuous vision system is bottlenecked by its long frame latency, which is fundamentally caused by its *serialized execution model*. The three major stages in a vision pipeline – sensing, imaging, and vision computation – process a frame sequentially, leading to high per-frame latencies. Many existing system optimizations such as pipelining and batching are designed to improve throughput (i.e., frame rate), but further exacerbate the frame latency.

The long frame latency is detrimental to vision-enabled embedded systems. For instance, a typical embedded robot today has a 200 ms responsive latency from event to command, in which 100 ms is attributed to the vision sub-system [13]. Such a high latency puts a hard bound on the agility of the robotic system, limiting user-experience and functionalities [12, 25]. Improving the latency of continuous vision systems, however, is mainly constrained by the tight energy and power budgets because embedded computing platforms routinely operate without active cooling or constant power supply [28].

This paper seeks to reduce the end-to-end latency of continuous vision tasks. Our key idea is to break the sequential execution chain. Specifically, we propose a new vision execution model where the vision front-end, i.e., the sensing and imaging stages, predicts future frames, and the vision computation stage operates *proactively* on predicted future frames. Once the actual frame is generated by the front-end and vindicates the predicted frame, the vision results are likely already available, reducing the frame latency.

While proactive vision is applicable to general vision systems, we perform an important case-study on mobile Systems-on-a-chip (SoCs), which are widely used in edge use-cases (e.g., AR/VR headsets, smartphones, robots) that are latency- and energy-constrained. We introduce PVF, a proactive mobile vision system, which significantly reduces the end-to-end frame latency with lower energy consumption while largely relying on existing mobile SoC hardware.

In its efficient realization of the proactive vision pipeline, PVF addresses two key challenges. First, the proactive execution model exposes more concurrent computations (frames) on the fly, increasing the hardware resource contention. PVF exploits the hardware heterogeneities available on mobile SoCs, which naturally exposes different IP blocks (e.g., GPU, DSP, NPU), to execute multiple outstanding frames concurrently, mitigating the resource contention.

Second, predictive execution introduces new computations due to frame prediction and thus increases the energy consumption. To reduce the speculation-induced energy overhead, PVF exploits the error-resilient nature of the *image formation* process (i.e., the sensing and imaging stages) and allows the predicted frames to be directly consumed by the vision algorithms without being checked with the captured frames. This relaxation saves energy by switching off the sensor and the Image Signal Processors (ISP), which in many use-cases could consume up to 50% of the total energy [42]. This form of approximation is different from conventional approximation in computer vision that approximates the vision algorithm itself. Instead, we approximate the inputs to the vision algorithms, an opportunity uniquely enabled by prediction, and in turn mitigates the prediction-induced energy overhead.

Leveraging the SoC heterogeneities while approximating the vision front-end exposes a complex scheduling problem to PVF. The scheduler must 1) control the approximation to meet accuracy requirements, and 2) optimize for latency under energy constraints by wisely mapping multiple outstanding frames onto multiple execution targets (i.e., IP blocks). To that end, PVF uses an offline-online collaborative strategy. The offline component identifies key accuracy-sensitive knobs and empirically constructs a model between the knobs and the accuracy. The online component schedules the frames on the SoC and dynamically tunes the knobs to minimize the latency while meeting the energy budget and accuracy goals.

To enable efficient implementation of the speculative execution model, PVF extends today's mobile SoC only minimally. We demonstrate the effectiveness of PVF on two common vision tasks: object tracking and object detection. We show that PVF is able to achieve up to 92% latency reduction under the same energy budget, or 60.3% energy reduction at same frame latency, all with negligible accuracy loss. In summary, this paper makes four contributions:

- We propose the proactive continuous vision execution model that predicts the vision front-end to reduce the end-to-end frame latency.
- We propose a front-end approximation technique that approximates the image formation process to mitigate the energy overhead introduced by speculation.
- We design an offline-online collaborative software system that controls the accuracy while minimizing the frame latency in proactive vision executions.
- We demonstrate PVF, an efficient implementation of the proactive vision execution model. PVF reduces the frame latency under the same or lower energy budget with little hardware overhead.

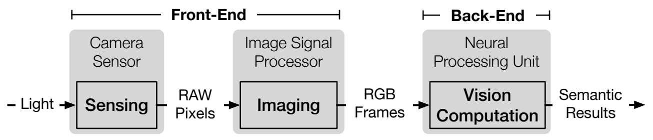


Fig. 1: The conventional sequential execution model of the continuous vision pipeline.

2 Background and Motivation

Continuous vision refers to a class of applications that perform computer vision (CV) tasks on live camera video streams. Different from offline video processing, continuous vision performs CV tasks in real-time and thus has much tighter performance and energy constraints. This section introduces the bottleneck of continuous vision systems, and describes the insufficiency of existing optimizations.

2.1 Latency Bottleneck in Continuous Vision

A typical continuous vision pipeline consists of three main stages: sensing, imaging, and vision computation, as Fig. 1 shows. The sensing stage uses an image sensor to convert photons to RAW pixels, which are then processed by an Image Signal Processor (ISP) at the imaging stage to generate RGB frames. The vision computation stage processes the generated RGB frames to extract semantic information (e.g., object location) for high-level decision making. We call the sensing and imaging stages the vision "front-end", and the vision computation stage "back-end."

The three stages execute sequentially because there is a strict dependency between any two adjacent stages. For instance, the vision computation stage cannot start until the RGB frames are generated by the ISP. The serialized execution limits the frame latency. That is, the time between when the sensor starts the sensing and when the vision results are available is limited by the cumulative execution time of all three stages.

Unfortunately, today's vision system optimizations further exacerbate the frame latency. For instance, pipelining the three stages increases the throughput, but introduces additional control and bookkeeping overhead that increases latency. In addition, batching multiple frames improves hardware utilization and data reuse, but also increases the per-frame latency. As a result, even if all three stages individually operate in real time, e.g., 30 frames per second (FPS), the end-to-end per frame latency could add up to over 100 ms. Long frame latency severely limits the applicability of vision-enabled systems. For instance, an autonomous vehicle needs to respond to an event within 100 ms as it can travel 2-3 meters during the interval. Similarly, a 100 ms rendering latency causes nausea and is intolerable to AR users [20].

2.2 Limitations of Optimizing Only Vision Algorithms

To reduce the frame latency, prior work has mostly focused on improving the performance of the vision computation stage, i.e., the back-end of the vision pipeline. Back-end optimization techniques include designing more efficient vision algorithms (e.g., simplified

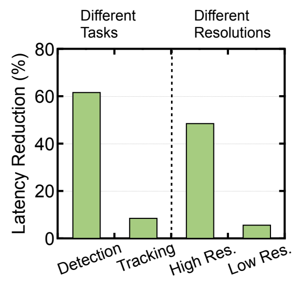


Fig. 2: **Per-frame latency reductions using a back-end optimization (motion extrapolation) to different tasks and input resolutions.**

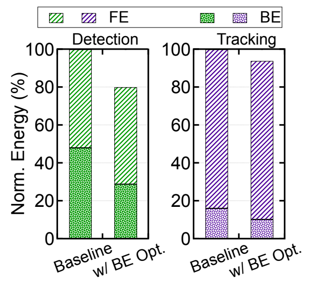


Fig. 3: **Per-frame energy breakdown between the front-end (FE) and back-end (BE) before and after applying motion extrapolation.**

Deep Neural Network (DNN) models [16, 35, 52, 67], compressing/pruning DNN models [29, 31]) andleveraging motion information [10, 22, 71]. Back-end optimizations are effective when the back-end dominates the frame latency. However, in many cases, depending on the complexity of the vision task and the image resolution, the front-end could contribute significantly to the frame latency, making the back-end optimizations ineffective.

We use the examples of the object detection and the object tracking task to illustrate the point. We assume that both the sensor and the ISP operate at a typical 30 FPS, and use a systolic-array-based DNN accelerator operating at 500 MHz to process vision DNNs (see Sec. 5 for detailed experimental setup). The detection task uses YOLOv2 [53], the state-of-the-art detection DNN, as the vision algorithm, while the tracking task uses the classic KCF algorithm [34].

YOLOv2 is compute-intensive, and operates at only 10 FPS; the KCF algorithm is much simpler and operates at 60 FPS. As a result, the frame latency of the detection task is back-end-dominant and the frame latency of the tracking task is front-end-dominant. Applying back-end optimizations would thus lead to less latency reduction for the tracking task than for the detection task. For instance, we apply the motion-extrapolation optimization proposed in Euphrates [71], which reduces the latency of the vision stage by a factor of 2. Fig. 2 compares the frame latency reductions between detection and tracking. A 2$\times$ vision stage latency reduction translates to only 9% frame latency reduction for tracking, much lower compared to the 62% reduction for detection.

Even for the same task, the latency distribution changes as the input resolution changes. Using object detection as an example, Fig. 2 compares the latency reductions of motion-extrapolation when applied to a 608 $\times$ 608 image and a 256 $\times$ 128 image. The latter spends significantly less time in the back-end, and thus gets improved only by 6% as compared to 49% observed on higher resolution inputs.

Along with latency reductions, most of the back-end optimizations also reduce the back-end's energy consumption. The energy reduction, however, is even less significant compared to latency reduction. This is because the vision front-end (sensor and ISP) usually contributes significantly to the total energy consumption. Fig. 3 shows the energy breakdown between the front-end and the back-end for object detection and tracking. The back-end energy contributes to only about 50% in detection and about 20% in tracking. Applying the motion-extrapolation optimization to the back-end thus reduces the overall energy consumption only marginally.

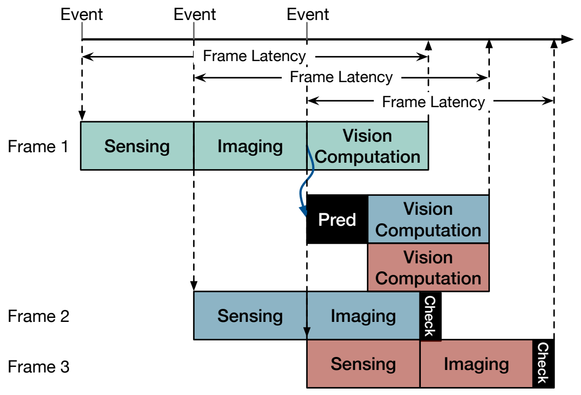


Fig. 4: **The speculative execution model. Pred denotes the frame predictor, and Check denotes the checking module.**

We propose to improve the latency and energy efficiency at the same time by breaking the sequential execution model. It is meant to complement, not replace, back-end optimizations to achieve greater latency and energy reductions.

3 Proactive Vision Execution Model

To overcome the limitation of the sequential execution in continuous vision pipelines, we propose the proactive execution model (Sec. 3.1). However, the new execution model introduces two roadblocks that obstruct efficient implementations. We discuss two enabling mechanisms, exploiting SoC heterogeneities (Sec. 3.2) and approximating vision front-end (Sec. 3.3), that remove the roadblocks and pave the way for efficient proactive execution in continuous vision.

3.1 Proactive Execution model

The key idea of the predictive execution model is to allow the vision computation stage to operate speculatively on predicted future frames before the sensing and imaging stages generate the actual frames. Once an actual frame is generated, it is used to validate the predicted frame. If the predicted frame is checked to match the actual frame under certain metrics, the vision task results are likely already available and can be directly used, reducing the end-to-end frame latency. Otherwise, the speculated work is discarded, and the system executes the vision stage using the actual frame.

We illustrate the new predictive execution model in Fig. 4 with an example of three frames. The system starts with the sequential model where the vision computation in Frame 1 waits for the actual frame. In contrast, Frame 2 and 3 are executed predictively where the vision stage operates on the predicted frames. Compared to the conventional sequential model, the predictive execution model now requires two new system components: a frame predictor and a checker (Pred and Check in Fig. 4, respectively).

First, the frame predictor predicts future frames to enable speculation. Frame predictors range from simple motion-based approaches (e.g., extrapolation and optical flow) that are computation-efficient especially in constrained environments [51] to recent generic approaches that use deep learning [9, 24, 44, 47, 49, 51, 58]. Critically, many frame prediction algorithms could *predict multiple future frames in a sequence* [9, 24, 44]. This allows us to speculatively process multiple outstanding frames at the same time (Frame 2 and 3 in Fig. 4), further reducing the latency.

Second, the predictive vision pipeline requires a checking component which commits vision results on predicted frames only if the predicted frames are validated to be similar to the actual frames captured by the sensor and ISP. This is similar to processor speculation that commits instructions on the predicted path only if the prediction is true. Many image similarity metrics and algorithms have been proposed in the computer vision literatures, ranging from pixel-level comparison (e.g., Sum of Absolute Differences [54]) to feature-level correlation [19, 44]. These algorithms are usually lightweight to calculate, enabling efficient checks.

3.2 Mitigating Resource Contention

Predictive execution enables multiple outstanding frames to be processed concurrently. However, it leads to hardware resource contention at the vision computation stage. For instance in Fig. 4, the vision stages of Frame 2 and Frame 3 are overlapped. If only one IP block is available to execute vision algorithms, the vision stage of the two frames must be serialized (similar to structure hazards in a processor pipeline), defeating the purpose of speculation.

To address the resource contention in the vision computation stage introduced by prediction, we propose to exploit the hardware heterogeneities available on today's mobile SoCs. State-of-the-art mobile SoCs such as Qualcomm Snapdragon 835 and Apple A11 all provide multiple IP blocks that can be used to execute vision algorithms, including the GPU, DSP, and NPU. Nvidia's Xavier SoC even provides dedicated accelerators for non-NN-based vision algorithms as well as an accelerator dedicated to stereo vision algorithms [5], further increasing the SoC hardware heterogeneities.

The heterogeneities are not fully exploited by today's vision systems, especially when the front-end dominates the frame latency, in which case there will be only one outstanding frame in the vision stage, and thus only one IP block is needed. In the predictive execution model, however, the frame predictor could predict multiple future frames in a sequence, exposing more outstanding frames that could make effective use of the multitude of the IP blocks. For instance in Fig. 4, the vision computations of Frame 1, 2, and 3 are concurrent, and could be scheduled to different IP blocks. How exactly the frames are scheduled to different IP blocks is critical to ensuring that the latency target is met in an energy-efficient manner. We discuss our run-time scheduling mechanisms in Sec. 4.3.

3.3 Mitigating Energy Overhead

While the predictive execution model reduces the frame latency, it increases the energy consumption, for three reasons. First, speculation fundamentally trades energy for latency by performing extra work (e.g., prediction and checking). Second, using multiple IP blocks, while alleviating resource contention, also increases the energy consumption because CPU/GPU/DSP are less energy-efficient than NPU for executing vision algorithms (e.g., DNNs). Finally, mis-prediction wastes energy on executing frames whose results are eventually discarded.

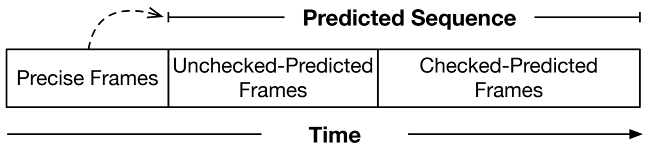


Fig. 5: Precise frames refer to non-predicted frames, which are used to predict future frames. Some predicted frames are unchecked against the actual frames while the rest are checked using a relaxed similarity measure.

To mitigate the energy overhead, we observe that the predictive exectuion is precise only if 1) the checking module enforces a pixel-perfect match between the predicted and the captured frame, and 2) every predicted frame is checked against the actual frame captured by the vision front-end. However, both are unnecessary because computer vision tasks are known to tolerate low-quality images [11, 18, 68, 71]. We relax both constraints, and allow inexact frames to be consumed by the vision algorithms to save energy.

- **Relax Checking Criterion** The strictest form of checking is to accept a predicted frame only when it is a pixel-perfect match with the actual frame. This would lead to an unnecessarily high mis-prediction rate and also significant energy waste. Instead, we propose to relax the checking process by thresholding the similarity metric. A predicted frame is regarded as mis-prediction only if its similarity measure is below the threshold.
- **Relax Checking Frequency** Taking relaxed checking one step further, we propose to forego the checking for a subset of predicted frames that are likely to be similar to the actual frames. In this way, not only the checking energy could be saved, the sensor and ISP that generate the actual frames could also be switched off, making up for the energy overhead introduced elsewhere. Since the sensor and ISP could take up to 50% of the total system power consumption [71], relaxing the frame checking frequency saves energy significantly.

Overall, there are three types of frames as Fig. 5 shows. *Precise frames* are frames that are generated from the sensor and ISP, and are used to predict future frames by the frame predictor. Among the predicted frames, the first few frames are *unchecked-predicted frames*, which are speculatively executed and do not require checking against the actual frames, thus saving both the checking and front-end energy. The rest frames in the prediction sequence are *checked-predicted frames*, which must be checked with the actual frames, albeit using a relaxed similarity measure. We make the design decision that the unchecked-predicted frames always precede the checked-predicted frames. This is because most frame predictors work in a recurrent fashion that use past frames to predict future frames [60]. As prediction progresses, prediction accuracy gradually drops and checking against actual frames becomes important.

Overall, two key parameters dictate the trade-off between accuracy and latency: the *similarity threshold* used in checking predicted frames, and the number of frames in a predicted sequence that are unchecked, which we dub "*unchecked-degree*." Sec. 4.2 will describe our offline-online collaboratively scheme to tune the two knobs to bound the accuracy while reducing latency.

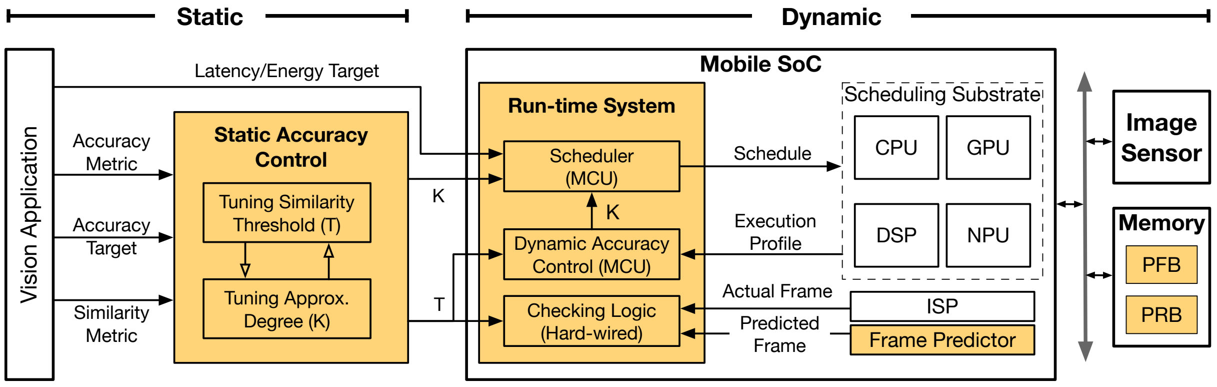


Fig. 6: Overview of the PVF system. Augmentations are colored. PFB: Pending Frame Buffer; PRB: Pending Result Buffer.

4 PVF framework

PVF is our system design that efficiently realizes the proactive execution model. We first provide a framework overview (Sec. 4.1), which consists components for accuracy control (Sec. 4.2), resource scheduling (Sec. 4.3), and mis-prediction handling (Sec. 4.4). They work together to minimize the latency of a vision task while sustain its accuracy target. We then discuss the hardware support to enable an efficient implementation of PVF (Sec. 4.5).

4.1 System Overview

Fig. 6 provides an overview of PVF. The goal of the PVF framework is to deliver a given latency target using the least energy while meeting the accuracy requirement. Accordingly, there are two main tasks of PVF: 1) ensuring the accuracy, and 2) delivering the latency target in an energy-efficient manner.

At run time, a frame predictor first predicts multiple future frames, which are directly fed to the back-end to perform vision tasks without waiting for the vision front-end. While a predicted frame could be executed by any IP block provided by the hardware, we propose a run-time design that intelligently maps the predicted frames to the IP blocks in a way that minimizes the overall frame latency while meeting a given energy budget. When an actual frame is produced by the front-end and matches with a predicted frame by the checking logic, the corresponding speculative result is committed. Upon a frame mis-prediction, the run time discards the current result, and re-starts the vision pipeline from the mis-predicted frame.

The vision task could experience accuracy drop due to the relaxations introduced in Sec. 3.3. PVF relies on static-dynamic collaboration to control accuracy. The static component identifies a set of knobs that accuracy is most sensitive to, and uses the profiling data to empirically set the knobs to meet the accuracy target with the least compute cost. With the "initial guess", the run-time system monitors the program execution, and dynamically tunes the knobs to adapt to run-time dynamisms and to meet the accuracy target.

4.2 Accuracy Control

Proactive execution is precise if every predicted frame is checked to be a pixel-perfect match with the actual captured frame. As established in Sec. 3.3, however, PVF exploits the error-resilient nature of vision tasks and introduces two sources of inexactness to improve efficiency. First, a portion of the predicted frames, controlled by the *unchecked-degree* K, are not checked against the actual frames. Second, the checking module does not enforce a pixel-perfect match; rather, it accepts a predicted frame if its similarity measure is above a *similarity threshold* T. PVF must carefully calibrate the two knobs, to control the accuracy while improving efficiency.

To balance the overhead and effectiveness of accuracy control, PVF uses the profile-guided optimization, where offline profiling data is used to guide the initial calibration of the knobs, which are further calibrated at run time to adapt to online dynamisms.

Static Control PVF first uses representative sample data to obtain initial values of K and T offline. To that end, PVF requires users to specify an image similarity metric (e.g., SSIM [61]), an accuracy metric (e.g., mean average precision, mAP, for object detection [40]), and an accuracy target. The offline accuracy control component profiles the vision pipeline against the sample input frames while sweeping T and K. The samples could either come from the training data or be supplied from users who have particular use-cases in mind. In the end, we obtain the combination of T and K that meets the accuracy requirement using the least compute. The statically tuned T and K are then used as the initial setting at run time.

Dynamic Control It is possible that the accuracy knob setting obtained/profiled offline is occasionally overly aggressive or conservative at run time. Therefore, PVF introduces a dynamic control mechanism to respond to run-time dynamisms that are not captured by offline profiling. Intuitively, if frame prediction is accurate, PVF could approximate the front-end more aggressively, and vice versa. In our design, if the mis-prediction rate is higher than a threshold in a predicted sequence, PVF will increase the checking frequency (i.e., increase the unchecked-degree K), and vice versa. Other more complex mechanisms such as using control theory [23] is also possible, but we empirically find that this simple mechanism performs well in practice.

4.3 Runtime Scheduling

The proactive execution model exposes a complex scheduling problem to the run-time system where multiple predicted frames are to be executed on a heterogeneous SoC with multiple execution targets (i.e., IP blocks). The scheduler must intelligently map frames to IP blocks in order to minimize the average frame latency without violating the energy budget. The scheduler could be easily extended to consider the dual problem of minimizing the energy under a latency constraint. Here we describe the former, but will evaluate both later.

Essentially, the scheduling task can be formulated as a constrained optimization problem. Without losing generality, assuming that the frame predictor predicts M frames, which are to be scheduled onto N IP blocks, and the unchecked-degree is K (i.e., the first K frames in the M predicted frames are not checked against the actual frames). The scheduling problem is:

$$min(\sum_{i=1}^{K} L_i^U + \sum_{i=K+1}^{M} L_i^C)\ s.t.\ E\ < E_{budget} \tag{1}$$

where L_i^U and L_i^C denote the frame latency of an unchecked-predicted frame and a checked-predicted frame, respectively. E is the total energy consumes across all the frames:

$$E = \sum_{i=1}^{K}\sum_{n=1}^{N} \beta_{in} \times E_n + \sum_{i=K+1}^{M}\sum_{n=1}^{N} \beta_{in} \times E_n \tag{2}$$

where E_n denotes the inherent energy consumption of IP block n, which could be dynamically profiled once at run time. The binary variable β_{in} denotes whether the frame i is scheduled to IP block n. The collection $\phi = \{\beta_{in}\}$ ($i \in \{1,2,...,M\}$, $n \in \{1,2,...,N\}$) thus uniquely determines an *execution schedule* for the M predicted frames. L_i^U and L_i^C are both expressed as functions of β_{in}; please refer to Appendix A for a detailed derivation.

Overall, our formulation optimizes over the schedule $\phi = \{\beta_{in}\}$ ($i \in \{1,2,...,M\}$, $n \in \{1,2,...,N\}$). Unfortunately, this optimization formulation is non-convex due to the complex interactions among $\{\beta_{in}\}$. Solving it by exhaustive search would require an algorithm with a time complexity of $O(N^M)$.

Instead, we use a lightweight greedy algorithm that works well in practice. Algorithm 1 describes the pseudo-code. Specifically, the scheduler schedules frames one at a time. Each frame is scheduled to the IP block that would provide the lowest frame latency — provided that the rest of the frames can possibly finish with the remaining energy budget using the least energy-consuming IP block; otherwise the rest of the frames are scheduled to the least energy-consuming IP. This algorithm is also lightweight to compute. It takes about tens of microseconds to execute on a micro-controller, as will be discussed in the hardware architecture later.

Note that scheduling (i.e., frames to IPs mapping) must be done at run time, because PVF assumes no prior knowledge of the energy consumption of each IP block.

4.4 Checking and Handling Mis-predictions

The run-time scheduler itself operates under the assumption that no frame mis-prediction occurs. While mis-predictions are rare as we will quantify in Sec. 6, they must be dealt with in a lightweight fashion. Upon a mis-prediction, the run-time system takes two actions. First, it discards the result generated for the mis-predicted frame. Second, it replaces the predicted frame with the actual frame and re-executes its vision stage. We define mis-prediction penalty as the time (energy) wasted in executing mis-predicted frames that are eventually discarded. As we will quantify in Sec. 6.8, the mis-prediction penalty is low. Note that it is possible that when an actual frame is generated the predicted frame have started its vision stage yet (i.e., waiting for the availability of the IP block that it is scheduled onto), in which case the mis-predicted frames are simply discarded, and there is no mis-prediction penalty.

Algorithm 1: Greedy Frame Scheduling Algorithm.

```
Input: Energy Budget $E_{budget}$; Predicted sequence length $M$;
       Number of IPs $N$; Latency for each IP $L_1,\dots,L_n$;
       Energy consumption for each IP $E_1,\dots,E_n$; Earliest
       available time for each IP $A_1,\dots,A_n$;
       Unchecked-degree $K$; ISP finish time for each frame
       $T_1,\dots,T_m$; Checking latency $C$.
Result: IP selection $\beta_i$ for all the predicted frames.
if $E_{budget} < M \times min(E_1,\dots,E_n)$ then
    return -1;
end
else
    for $i$ = 1 to $M$ do
        $t = \infty$
        for $j$ = 1 to $N$ do
            if $E_{budget}$ - $E_j \geq (M$-1$) \times min(E_1,\dots,E_n)$ then
                if $i < K$ then
                    if $A_j + L_j < t$ then
                        $\beta_i = j$;
                        $t = A_j + L_j$;
                    end
                end
                else
                    $f = max(A_j+L_j, T_j+C)$
                    if $f < t$ then
                        $\beta_i = j$;
                        $t = f$;
                    end
                end
            end
        end
        $E_{budget} = E_{budget}$ - $E_{\beta_i}$;
        for $j$ = 1 to $N$ do
            $A_j = A_j + (j == \beta_j)\ ?\ L_j : 0$;
        end
    end
end
```

4.5 Architectural Augmentations and Implementation Details

A key design objective of PVF is to maximally reuse existing mobile SoC architecture. PVF requires four principled augmentations to existing mobile systems as shown in Fig. 6.

Frame Predictor PVF requires a frame predictor. While any frame prediction algorithms would be compatible with the PVF framework, we choose to a CNN-based frame prediction algorithm PREDNET [43]. The recurrent model predicts a sequence of consecutive frames by using both the current frame and the hidden state that contains historical information. Intuitively, predicting more frames increases the scheduling window, but also introduces a higher chance of mis-prediction. We empirically find that 10 frames are desirable (i.e., M in Equ. 1), which in turns leads to about 81 million MAC operations, indicating a compute cost over two orders of magnitude lower than that of a typical vision task such as object tracking and detection [71]. Other even lighter predictors (e.g., using motion vectors [54]) could alternatively be used in ultra-low power systems.

To not compete for the resources with the vision tasks, the frame predictor is executed using a dedicated NPU rather than reusing the main NPU. Compared to hard-wiring the predictor logic, using a programmable NPU allows PVF to support other predictors in the future if needed.

Checking Logic The frame checking is done in hardware. We use the widely-used SSIM metric [61] to assess the quality of the predicted frames compared to the actual frames. SSIM is lightweight to compute compared to other system components. Given an input resolution of 608 $\times$ 608, calculating the SSIM requires only 7.4 million MAC operations, a few orders of magnitude lower than the vision algorithms.

Memory The proactive execution model requires two new data structures: the Pending Frame Buffer that stores predicted frames and the Pending Result Buffer that stores finished results that are not yet committed. In our implementation, we reserve two regions in the physical memory for the two data structures. Sec. 6.1 will show that the memory overhead is negligible.

MCU We execute the run-time scheduler on a micro-controller unit (MCU), which is common to mobile SoCs. The run-time system is lightweight in computation and requires programmability to adapt to future changes in the run-time policy, which a MCU provides.

5 Experimental Methodology

We describe the experimental setup (Sec. 5.1), the baselines (Sec. 5.2), and the evaluation scenarios (Sec. 5.3).

5.1 Basic Setup

Software Setup We evaluate PVF using two common vision tasks: object detection and object tracking. We use YOLO [53], a state-of-the-art CNN as the detection algorithm. We conduct the object detection applications on the widely-used KITTI dataset [26]. We evaluate two input resolutions: 608 $\times$ 608 and 256 $\times$ 128. We use the common mean Average Precision (mAP) as the accuracy metric [21], which is the mean of the average precisions across all the object classes. We use ECO [17], a state-of-the-art CNN as the tracking algorithm. We use VOT challenge 2017 benchmark as our dataset [40] and use the commonly used Expected Average Overlap (EAO) as the accuracy metric [39], which is the mean of the average per-frame overlap across sequences with typical lengths.

Hardware Setup We develop an in-house simulator parameterized with real hardware measurements for modeling the continuous vision pipeline. We model the baseline as a typical mobile SoC consisting of key IP blocks including the CPU, GPU, ISP, and DSP. We model the AR1335 image sensor [1], and use the energy numbers reported in the data sheet. The ISP power is measured from the Nvidia Jetson TX2 module. Given a 256 $\times$ 128 low resolution image, the sensor and ISP are set to run at 40 FPS, and consume 4.5 mJ and 3.8 mJ per frame, respectively. Given a 608 $\times$ 608 high resolution image, the sensor and ISP are set to run at 30 FPS, and consume 6.5 mJ and 5.4 mJ per frame, respectively.

We model an NPU in the baseline to execute vision DNNs and a separate NPU for the frame predictor (Sec. 4.5). Any existing NPUs could be integrated into PVF, since PVF does not change the microarchitecture of a NPU. For the purpose of our evaluation, we use SCALESIM [56], a RTL-validated DNN accelerator simulator, to model the behaviors of both NPUs. The NPU has an array size of 20×20 operating at 500 MHz with a SRAM size of 1.5 MB. The predictor NPU has an array size of 10×10 operating at 350 MHz with a SRAM size of 128 KB. We implement the checking module as a 6-way SIMD MAC units running at 500 MHz.

PVF makes use of other non-NPU IP blocks during scheduling. We model a Qualcomm Hexagon 680 DSP, whose power consumption is modeled through direct measurement from the Open-Q 820 Hardware Development Kit [6]. We model the Pascal GPU from the Jetson TX2 [3], whose power consumption is modeling through direct measurement using its built-in power measurement. The DRAM is modeled after four Micron 16 GB LPDDR3-1600 channels [4]. Finally, we model an ARM M4-grade micon-controller [2], which is used for executing the PVF run-time scheduler. Appendix B shows the measurement results that we use for parameterizing the simulator.

5.2 Baselines

We compare with three different baselines:

- Base, which is the baseline system without predictive execution. Base uses YOLO [53] for object detection and ECO [17] for object tracking.
- BO, which optimizes Base by optimizing the vision back-end, i.e., the vision algorithm. It represents a common strategy today to reduce the frame latency. While there are many back-end optimization techniques such as model compression [15, 30, 37] and motion compensation [71], they all have the same effect of reducing the vision algorithm's latency. In our evaluation we choose to replace complex but accurate DNN models with simplified but less accurate DNN models.

  In particular, we train a simplified version of YOLO, which we call O-YOLO, which has 41.1% fewer MAC operations than YOLO with only 0.15 mAP loss on low resolution inputs and 2.3 mAP loss on high resolution inputs. For object tracking, we use KCF [34], which uses hand-crafted features and has 80% lower compute cost with 0.03 EAO drop on low resolution inputs and 0.153 on high resolution inputs compared to ECO.
- FCFS, which improves BO by utilizing all the IP blocks available for vision computation but *without* prediction capabilities. Therefore, comparing PVF with FCFS will show the benefits of the proactive execution model. The FCFS scheduler maps incoming frames to the fastest available IP block using a First-Come-First-Serve policy.

5.3 Evaluation Scenarios

Since PVF predicts and sometimes even skips the vision front-end, its effectiveness is naturally sensitive to the latency distribution between the front-end and the back-end. To understand when and why PVF provides latency reduction, we evaluate it under three different usage scenarios:

- Back-End Dominant, which refers to a scenario where the back-end dominates the original vision pipeline (i.e., a high-latency but accurate vision algorithm is used). Object detection with high resolution falls into this category.
- Front-End Dominant, which refers to a scenario where the front-end dominates the original vision pipeline. Object tracking with low resolution falls into this category.
- Mixed, which refers to a scenario where the back-end dominates the original vision pipeline, but front-end becomes the bottleneck when an optimal vision algorithm is used. Object tracking (both high and low resolutions) fits this scenario.

6 Evaluation

We first show the software and hardware overhead of PVF (Sec. 6.1). We then show that with careful accuracy tuning, the accuracy drop of front-end approximation is negligible (Sec. 6.2). We further demonstrate that PVF achieves latency reduction with the same or lower energy budget in different scenarios. (Sec. 6.3, Sec. 6.4, and Sec. 6.5). We apply the same run-time optimization formulation to the duel problem of saving energy under latency budgets (Sec. 6.6). We show how sensitive PVF is to different accuracy knobs (Sec. 6.8). Finally, we compare PVF with an alternative that simply lows the image resolution in an iso-accuracy setting (Sec. 6.7), and compare our PVF implementation with an ideal implementation that has an oracle predictor (Sec. 6.8).

6.1 Overheads

Hardware Overhead PVF adds two new hardware components: predictor and checker. In a 16 nm technology node, the predictor is about 170,000 um^2 in area and the checking module is about 1,500 um^2 in area. Both combined contribute to less than 0.15% of the total SoC die area [7]. In addition, the two pending buffers are pinned in the DRAM (Sec. 4.5). The Pending Frame Buffer has a size of 1.5 MB and the Pending Result Buffer has a size of 200 Bytes.

Prediction Overhead The prediction overhead is low. For high resolution inputs, the latency of the predictor is 12.5 ms per frame, which is about 5 $\times$ faster than the front-end; the energy overhead is less than 0.5 mJ per frame, which is 22 $\times$ lower than the front-end. For low resolution inputs, the latency and energy overhead is less than 1.0 ms and 0.1 mJ per frame, respectively, which are about 50 $\times$ faster and consumes 83 $\times$ lower energy than the front-end.

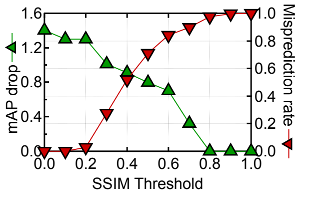
Checking Overhead Checking also have insignificant overhead, with a mere 2.5 ms and 0.2 ms latency for high and low resolution inputs, respectively.

Scheduling Overhead The run-time scheduler uses a simple greedy algorithm. The run-time scheduler executes within 100 μs on an M4-like micro-controller.

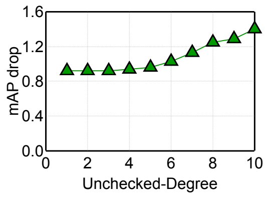
6.2 Accuracy Results

We evenly split the dataset into an offline profiling set and an online test set. The profiling set is used by the static accuracy control module to empirically decide the two accuracy knobs, similarity threshold T and unchecked-degree K, while the test set is used at run-time evaluation.

Object Detection We set an accuracy target of less than 1.5 mAP loss compared to Base, on par with the accuracy drop in prior

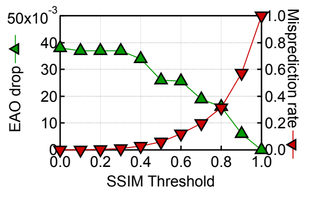


(a) **Accuracy sensitivity to different similarity (SSIM) thresholds.**

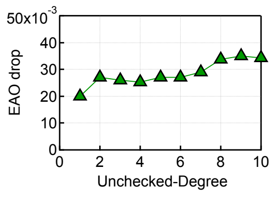


(b) **Accuracy sensitivity to different unchecked-degrees.**

Fig. 7: Accuracy sensitivity in object detection.



(a) **Accuracy sensitivity to different similarity (SSIM) threshold.**



(b) **Accuracy sensitivity to different unchecked-degree.**

Fig. 8: Accuracy sensitivity in object tracking.

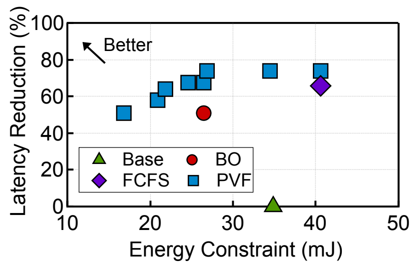
work [10, 71]. Fig. 7 shows how the mAP drop (left y-axis) and mis-prediction rate (right y-axis) vary with the SSIM (similarity) threshold for the *profiling set*. As the SSIM threshold relaxes from right to left, the mis-prediction rate decreases and therefore PVF accepts more inexact input frames. At the same time, the accuracy drop also increases. Similarly, Fig. 7b shows how the mAP changes with the unchecked-degree. As more frames are predicted without being checked, the mAP drop increases.

The offline accuracy control module tunes both knobs together and settles for the configuration with $T = 0.5$ and $K = 6$. We find that this setting is stable at run time. The average number of unchecked frames increases by only 0.72 in each predicted sequence. Overall, the static-dynamic collaborative approach in PVF leads to an 1.13 mAP loss on the *test set*, which would have been 4.4% higher if only the static component is employed.

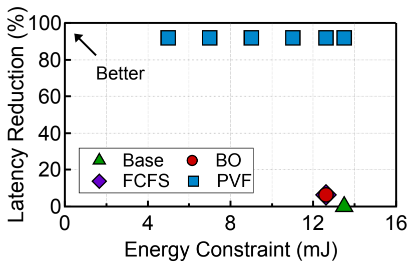
Object Tracking We set an accuracy target of less than 0.1 EAO loss compared to Base, which is better than switching from a CNN-based tracker to the best non-CNN tracker [38]. Fig. 8a and Fig. 8b show how the accuracy drop varies with the SSIM threshold and the unchecked-degree for the *profiling set*. The offline tuning module chooses 0.7 as the SSIM threshold T and 6 as the unchecked-degree K, which again is stable at run time. The average number of unchecked frames increases by only 0.57 in each predicted sequence. Overall, the static-dynamic scheme in PVF leads to leads to 0.051 EAO loss on the *test set*, which would have been 13.7% higher if the accuracy knobs are tuned only offline.

6.3 Latency in Back-End Dominant Scenario

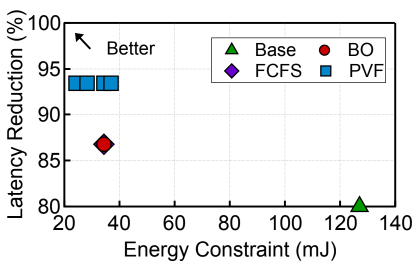
PVF run-time system schedules frames in a predicted sequence onto different IP blocks given a specific energy budget in order to minimize latency. We show in this section the results of PVF under different energy budgets when the vision back-end dominates the



(a) Latency reduction under different energy budgets in back-end dominant scenario.

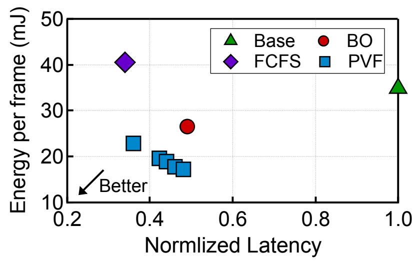


(b) Latency reduction under different energy budgets in front-end dominant scenario.

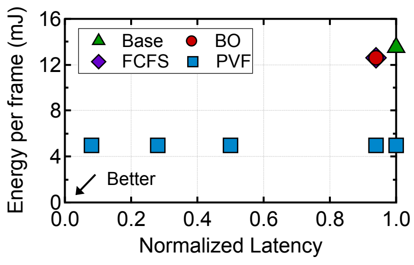


(c) Latency reduction under different energy budgets in mixed scenario.

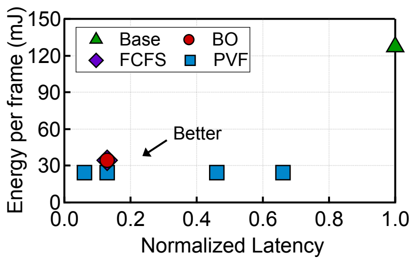
Fig. 9: Frame latency reductions under different energy constraints in the three evaluated scenarios.



(a) Energy consumptions under different latency constraints in back-end dominant case.



(b) Energy consumptions under different latency constraints when front-end dominates.



(c) Energy consumptions under different latency constraints in the mixed scenario.

Fig. 10: Frame energy consumptions under different latency constraints in the three evaluated scenarios.

pipeline latency. We use object detection with high resolution as a concrete use-case for this scenario.

Fig. 9a shows the latency reduction of PVF over Base (y-axis) under different energy constraints (x-axis). Note that while PVF can flexibly adapt to different energy budgets thanks to its run-time scheduler, the three baselines are fixed designs and their latencies do not change with the energy budget. We purposefully choose the energy budgets for PVF to match the energy consumptions of the three baselines, so that we can compare PVF with the baselines in an iso-energy manner.

Compared with Base, BO achieves 51.0% latency reduction. The significant latency reduction comes from shaving off the vision stage latency that dominates the pipeline. FCFS further reduces the latency by 14.9% because it employs multiple IP blocks to reduce the resource contention of the vision back-end. Compared to FCFS, PVF is able to further achieve 8.0% latency reduction under the same energy budget. This is because through prediction PVF provides more frames to the back-end and thus better utilizes the available IP blocks in a heterogeneous SoC. Compared with Base and BO, PVF's latency reduction is 67.4% and 16.4%, respectively.

In addition, PVF extends the energy-latency trade-off space. Given different energy budgets, PVF's scheduler automatically identifies the most suitable frame-to-IP mapping that meets the energy with minimal latency. Overall, PVF pushes the Pareto-optimal frontier further than that of the three baselines, providing better system operating choices that are more energy-efficient with lower latency.

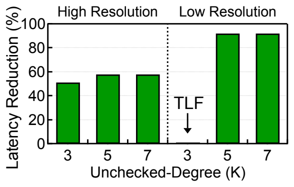
6.4 Latency in Front-End Dominant Scenario

Fig. 9b shows the latency reduction of BO, FCFS, and PVF over Base in front-end dominant scenarios. Overall, both BO and FCFS have little latency improvement over Base, but PVF provides significant latency reduction.

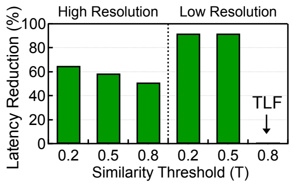
When the front-end dominates the end-to-end latency, the back-end (i.e., vision stage) is not the bottleneck. Therefore, BO only has marginal latency reduction over Base (6.4%) as the former optimizes the back-end vision algorithms. In addition, in front-end dominant scenarios, whenever a new frame is produced by the front-end, the NPU will always be available because the back-end is fast. Therefore, the FCFS scheduler will schedule incoming frames only to the NPU (as other non-NPU IP blocks have higher energy consumption than the NPU), essentially falling back to BO. As a result, FCFS leads to an almost identical latency compared to BO.

In contrast, PVF provides significant latency reduction due to the ability to proactively execute future frames without waiting for the front-end. Under the same energy budget, PVF provides 85.6%, 85.6%, and 92.0% frame latency reduction compared to BO, FCFS, and Base, respectively.

In front-end dominant scenarios PVF does not provide a large energy-latency trade-off as it does in back-end dominant scenarios. This is because the NPU is always available and the scheduler will always schedule incoming frames to the NPU.

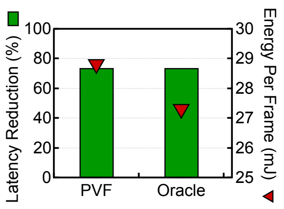


(a) Latency reductions under different unchecked-degrees K.

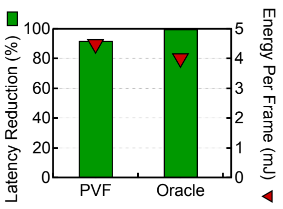


(b) Latency reductions under different similarity thresholds T.

Fig. 11: Sensitivity Analysis. TLF denotes "too low to finish."



(a) Back-end dominant case.



(b) Front-end dominant case.

Fig. 12: Oracle Analysis using object detection.

6.5 Latency in Mixed Scenario

Fig. 9c shows the latency reductions of BO, FCFS, PVF over Base in the mixed scenario. In this scenario, the back-end dominates the pipeline latency with the original vision algorithm in Base. Therefore, by optimizing the vision algorithm BO significantly outperforms Base, reducing the latency by 86.8%. However, once the vision algorithm is optimized, the front-end dominates the latency. Thus, FCFS doesn't achieve any latency improvement over BO, and the PVF achieves only 6.7% further latency reduction over FCFS.

6.6 Energy Improvements

So far, we have been evaluating PVF under the objective of minimizing latency under different energy budgets. PVF could also operate under a dual problem of minimizing energy consumption under latency constraints. We briefly evaluate PVF under this objective.

Fig. 10a, Fig. 10b, and Fig. 10c show how PVF can reduce the frame energy consumption under given latency constraints for the same three scenarios as described in Sec. 5.3, respectively. Similarly to the latency-oriented optimization, PVF shows the most significant energy savings when the front-end dominates the vision pipeline latency, such as the scenario shown in Fig. 10b. In that case, PVF is able to achieve 55.6%, 52.4%, and 52.4% per frame energy saving compared to Base, BO, and FCFS, respectively.

6.7 Comparison Against Lower Image Resolution

PVF out-performs an alternative that simply lowers the image resolution. Due to the space limit, we show the results of the front-end dominant scenario (i.e., object detection with low resolution). We designed an iso-accuracy experiment, in which we lower the input image resolution by 25%. Compare against simply lowering the resolution, PVF reduces latency by 90.0%, slightly lower than the 92.0% reduction on the original resolution but is still significant.

6.8 Sensitivity Study

We study the sensitivity of PVF with respect to K and T, the two knobs that affect the accuracy-latency trade-offs. Due to limited space, we use object detection as a case-study, but the general trend holds. Fig. 7 shows the accuracy sensitivity, and this section focuses on the latency and energy sensitivities.

Unchecked-Degree Fig. 11a shows the latency reduction of PVF with different unchecked-degrees (K) under the same energy constraint (21.0 mJ for high resolution inputs and 5.0 mJ for low resolution inputs). Generally, as K increases, the front-end could be switched off more often, leaving more energy budget for the back-end to active more IP blocks to reduce the frame latency. For high resolution inputs, the latency reduction improves from 51.0% to 57.8% as K increases from 3 to 5. The improvement plateaus as K increases to 7. The reason is that by then the saved energy from switching off the front-end is not enough to allow the scheduler to activate another IP block to gain more latency reduction.

For low resolution inputs, under a K of 3 the energy budget is not enough to let the system finish all the frames (denoted as "too-low to finish", TLF). As K increases, the energy saved from switching off the front-end allows the system to finish executing all the frames and achieve significant latency reduction. Similarly to the high resolution scenario, the latency reduction plateaus beyond $K = 5$.

Similarity Threshold A higher SSIM threshold (T) would result in more mis-predictions, leading to higher energy waste since the vision results calculated on the mis-predicted frames will be discarded. As a result, the latency reduction would decrease because the back-end has less energy to activate new IP blocks to reduce latency. Fig. 11b shows the latency reduction with different Ts under the same energy constraint (22.8 mJ for high resolution input and 5.0 mJ for low resolution input). As T increases, the latency reduction decreases. For the low resolution case as T increases to 0.8 the energy budget is too small to let the system finish executing all the frames.

6.9 Oracle Analysis and Mis-Prediction Penalty

The effectiveness of PVF is correlated with the accuracy of the frame predictor. We now assess how PVF would perform with an oracle predictor (Oracle), which uses the same prediction algorithm [43] and the same hardware, but has a perfect prediction accuracy. That is, the only difference between PVF and Oracle is that the latter does not pay the mis-speculation penalty (the energy wasted on executing vision algorithms on mis-predicted frames). By comparing PVF with Oracle, we can not only assess the full potential of PVF, but also understand its mis-speculation penalty.

Fig. 12a shows the results of the back-end dominant scenario, in which Oracle consume 5.2% lower energy (right y-axis) than PVF, equivalent to PVF's mis-speculation energy penalty. PVF has almost identical latency reduction compared to Oracle. This is because when the back-end dominates speculating the front-end has little impact on improving the frame latency. The gap between Oracle and PVF is more notable in the front-end dominant case, which benefits from a more accurate frame predictor. The results of the front-end dominant case are shown in Fig. 12b. Oracle consumes 12.3% lower energy and 8.0% lower latency compare to PVF.

7 Limitation and Discussion

Applicability The accuracy loss of PVF in its current is similar to prior work on approximating vision computations [10, 71]. PVF currently does *not* target mission-critical systems that have tight accuracy requirements. PVF could be used in vision systems such as AR and robotics navigation that are less accuracy sensitive. Note that PVF is compatible with any frame predictor. Therefore, PVF can readily benefit from a higher quality frame predictor with the algorithmic innovations from the vision community.

Predictor Alternatives PVF predict future frames. However, obtaining the exact pixels in future frames is not always necessary. We discuss two alternatives here for future developments.

First, one could directly predict frame features rather than pixels [59]. This is potentially efficient as CNNs extract features from pixels anyways, and operate on features rather than raw pixels. The challenge is that there is no well-established comparison metric in the feature space, unlike the SSIM metric used in the pixel space. We empirically find that simply comparing the Euclidean distance between two feature vectors leads to high error rates.

Second, for applications where the semantics output is a region-of-interest (ROI), we could directly predict the ROI location. However, predicting images allows PVF to be generally applicable to any backend vision algorithms, including those that generate more than ROIs (e.g., the object class in object detection).

Security Vulnerabilities The proposed speculative vision system does not share the security vulnerabilities introduced by speculations in CPU microarchitecture such as Spectre and Meltdown. The fundamental reason is that Spectre and Meltdown exploit speculation at the *microarchitecture*-level (branch prediction and out-of-order execution) to leak information, whereas the proposed speculative vision execution is an *application*-level technique, in which the predicted frames are in non-speculative states from the microarchitecture's perspective. Thus, our proposal does not expose new security vulnerabilities to Spectre and Meltdown attacks.

However, the predictive execution model does expose mis-predicted frames, which could be potentially be exploited to infer system internal behaviors. The security implication of predictive vision execution should be carefully studied, which we leave for future work.

8 Related Work

Vision Optimizations Literature is rich with techniques that optimize or approximate the vision algorithms. Classic techniques include using smaller, simplified models [35, 52, 67], model compression [15, 30, 37], quantization [36, 63–65], and using specialized hardware [14, 27, 48]. Recent developments have also leveraged the temporal/spatial redundancies in real-time frames to simplify computations [10, 22, 46, 55, 71].

PVF differs from prior work in two key ways. First, PVF expands the optimization scope from optimizing only the back-end vision stage to the whole continuous vision pipeline. Specifically, PVF predicts the front-end, and is complementary to existing back-end optimization techniques. Second, PVF presents a new form of vision approximation by relaxing the checking constraint of the vision pipeline front-end, saving energy from the vision front-end.

Heterogeneity in Mobile Computing Mobile SoCs are rich in architectural heterogeneity to deal with a wide variety of use-cases that have different compute intensities. Industry has provided mature programming frameworks [50]. The heterogeneity has been exploited in various contexts such as Web browsing [69, 70] and machine learning [32, 41, 57].

Critical to exploiting hardware heterogeneity is scheduling, which decides which tasks are executed on which IP block to meet a given objective. While task scheduling in heterogeneous system has been extensively studied before [8, 45, 62, 66], PVF introduces unique challenges because the scheduler must deal with a stream of incoming tasks that have dependencies. We formulate the scheduling task in PVF as a constrained-optimization problem and demonstrate an efficient greedy algorithm that performs well in practice.

9 Conclusion

Long frame latency is detrimental to real-time vision systems. This paper argues that the sequential execution model is the culprit of the long latency. We propose PVF, a speculative vision execution model that reduces the frame latency under the same or lower energy budget. The key to PVF is to break the sequential execution between vision front-end and back-end. We present an efficient implementation of PVF by leveraging the heterogeneities in mobile SoCs and exploiting the error-tolerance nature of vision applications.

10 Acknowledgement

We thank the anonymous reviewers for their valuable feedback. We also thank Arm for their equipment donation. Jingwen Leng is the corresponding author of the paper. All opinions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsor.

Appendices

A Latency Derivation in Scheduling

The latencies in Equ. 1 are calculated as the time difference between the start time S_i^U (S_i^C) and the finish time F_i^U (F_i^C). For both checked- and unchecked-predicted frames, a frame start when its sensing stage starts. Thus, we have:

$$L_i^U = F_i^U - S_i,\ L_i^C = F_i^C - S_i \tag{3}$$

$$S_i = (i-1) \times T_{sense} \tag{4}$$

where T_{sense} is sensing stage time, which is statically determined given a particular sensor frame rate.

A frame's finish time depends on its type. An unchecked-predicted frame finishes when its vision stage finishes. If two frames' vision stages are mapped to the same IP block, the latter starts only after the former finishes. Thus:

$$F_i^U = \sum_{k=1}^{i-1} \alpha_{ik} \times (F_k + T_i), \alpha_{ik} \in \{0,1\}, \sum_{k=1}^{i-1} \alpha_{ik} = 1 \tag{5}$$

where T_i represents the vision stage time of frame i, and the binary variable α_{ik} denotes whether or not Frame i is executed right after Frame k on the same IP block. Frame k itself could either be an approximate frame or a predicted frame.

Table 1: Latency of object detection and object tracking using both high and low input resolution on different IPs. All numbers are in milliseconds.

IP	High-Res				Low-Res			
	YOLO	O-YOLO	ECO	KCF	Yolo	O-YOLO	ECO	KCF
NPU	173	100	-	-	13	7	-	-
GPU	651	491	-	-	447	421	-	-
DSP	743	650	-	-	521	490	-	-
CPU	>2000	>2000	83	22	>2000	>2000	72	14

For a checked-predicted frame, it finishes when the checking finishes, which depends on whether the vision stage finishes before or after the actual frame is generated:

$$F_i^C = max(\sum_{k=1}^{i-1} \alpha_{ik} \times (F_k + T_i), \text{i} \times T_{sense} + T_{isp}) + C_i,$$
$$\alpha_{ik} \in \{0,1\}, \sum_{k=1}^{i-1} \alpha_{ik} = 1 \quad (6)$$

where T_{isp} is imaging stage time, which is statically determined given the ISP frame rate; C_i is the checking time, which is also statically determined given the similarity metric and image resolution.

T_i, the vision stage execution time of a frame i, depends on the particular IP block that is scheduled to execute the frame. Thus, they can be expressed as follows:

$$T_i = \sum_{n=1}^{N} \beta_{in} \times L_n, \beta_{in} \in \{0,1\}, \sum_{n=1}^{N} \beta_{in} = 1 \quad (7)$$

where L_n denotes IP block n's latency, which is profiled given a particular image resolution. The binary variable β_{in} denotes whether the frame i is scheduled to IP block n. The collection $\phi = \{\beta_{in}\}$ $(i \in \{1,2,...,M\}, n \in \{1,2,...,N\})$ thus uniquely determines an execution schedule for the M predicted frames.

Note that α_{ik} is not an independent variable; it could be derived from β_{in} and β_{kn}. If both β_{in} and β_{kn} are 1 and β_{zn} is 0 for any $z \in (k,i)$, indicating that frame k and frame i are scheduled to execute on the same IP block n consecutively, α_{ik} has to be 1. Leveraging the arithmetic properties of binary values, α_{ik} could be expressed as:

$$\alpha_{ik} = !\prod_{n=1}^{N} (\beta_{in} \times \beta_{kn} \times (\sum_{z=i}^{k} \beta_{zn} - 1) - 1) \quad (8)$$

where ! is the logic negate that returns 0 for any non-zero values.

B Latency and Energy Characterizations

Table 1 and Table 2 show the latency and energy measurement results that are used to parameterize the simulator described in Sec. 5.1.

Table 2: Energy of object detection and object tracking using both high and low input resolution on different IPs. All numbers are in millijoules.

IP	High-Res				Low-Res			
	YOLO	O-YOLO	ECO	KCF	YOLO	O-YOLO	ECO	KCF
NPU	21	13	-	-	3	2	-	-
GPU	77	70	-	-	35	21	-	-
DSP	57	51	-	-	25	16	-	-
CPU	>160	>160	118	20	>160	>160	101	18

References

[1] [n. d.]. AR1335: CMOS Image Sensor, 13 MP, 1/3". https://www.onsemi.com/PowerSolutions/product.do?id=AR1335.

[2] [n. d.]. Cortex-M4. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4.

[3] [n. d.]. Jetson TX2 Module. http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html.

[4] [n. d.]. Micron 178-Ball, Single-Channel Mobile LPDDR3 SDRAM Features. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf.

[5] [n. d.]. NVIDIA Reveals Xavier SOC Details. https://bit.ly/2qq0TWp. https://www.forbes.com/sites/moorinsights/2018/08/24/nvidia-reveals-xavier-soc-details/amp/

[6] [n. d.]. Open-Q 820 Development Kit. https://www.intrinsyc.com/snapdragon-embedded-development-kits/snapdragon-820-development-kitsnapdragon-820-apq8096/.

[7] [n. d.]. Qualcomm Snapdragon 835 First to 10 nm. https://www.wired.com/story/apples-neural-engine-infuses-the-iphone-with-ai-smarts/.

[8] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. 2011. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience* 23, 2 (2011), 187–198.

[9] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. 2018. Stochastic Variational Video Prediction. *Proc. of ICLR* (2018).

[10] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. 2018. *EVA*^{2}: Exploiting Temporal Redundancy in Live Computer Vision. In *Proc. of ISCA*.

[11] Mark Buckler, Suren Jayasuriya, and Adrian Sampson. 2017. Reconfiguring the Imaging Pipeline for Computer Vision. In *Proc. of ICCV*.

[12] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. 2009. *The DARPA urban challenge: autonomous vehicles in city traffic*. Vol. 56. springer.

[13] Andrea Censi and Davide Scaramuzza. 2014. Low-latency event-based visual odometry. In *Proc. of ICRA*.

[14] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan 2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[15] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. 2018. Universal deep neural network compression. *arXiv preprint arXiv:1802.02271* (2018).

[16] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.

[17] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. 2017. Eco: Efficient convolution operators for tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6638–6646.

[18] Steven Diamond, Vincent Sitzmann, Stephen Boyd, Gordon Wetzstein, and Felix Heide. 2017. Dirty Pixels: Optimizing Image Classification Architectures for Raw Sensor Data. In *Proc. of CVPR*.

[19] Alexey Dosovitskiy and Thomas Brox. 2016. Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems*. 658–666.

[20] David Drascic and Paul Milgram. 1996. Perceptual issues in augmented reality. In *Stereoscopic displays and virtual reality systems III*, Vol. 2653. International Society for Optics and Photonics, 123–135.

[21] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88, 2 (2010), 303–338.

[22] Yu Feng, Paul Whatmough, and Yuhao Zhu. 2019. ASV: Accelerated Stereo Vision System. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[23] Laurent Fesquet and Hatem Zakaria. 2009. Controlling energy and process variability in system-on-chips: needs for control theory. In *2009 IEEE Control Applications,(CCA) & Intelligent Control,(ISIC)*. IEEE, 302–307.

[24] Chelsea Finn and Sergey Levine. 2017. Deep visual foresight for planning robot motion. In *Proc. of ICRA*.

[25] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. 2002. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics* 25, 1 (2002), 116–129.

[26] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research* 32, 11 (2013), 1231–1237.

[27] Cong Guo, Yangjie Zhou, Jingwen Leng, Yuhao Zhu, Zidong Du, Quan Chen, Chao Li, Minyi Guo, and Bin Yao. 2020. Balancing Efficiency and Flexibility for DNN Acceleration via Temporal GPU-Systolic Array Integration. *arXiv preprint arXiv:2002.08326* (2020).

[28] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. 2016. Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction. In *Proc. of HPCA*.

[29] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). arXiv:1510.00149 http://arxiv.org/abs/1510.00149

[30] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). arXiv:1510.00149 http://arxiv.org/abs/1510.00149

[31] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 1135–1143. http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf

[32] Gopalakrishna Hegde, Siddhartha, and Nachiket Kapre. 2017. CaffePresso: Accelerating Convolutional Networks on Embedded SoCs. *ACM Trans. Embed. Comput. Syst.* 17, 1, Article 15 (Nov. 2017), 26 pages. https://doi.org/10.1145/3105925

[33] John L Hennessy and David A Patterson. 2018. *Computer architecture: a quantitative approach, Sixth Edition*. Elsevier.

[34] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. 2015. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence* 37, 3 (2015), 583–596.

[35] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[36] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[37] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530* (2015).

[38] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman Pflugfelder, Luka Cehovin Zajc, Tomas Vojir, Gustav Hager, Alan Lukezic, Abdelrahman Eldesokey, et al. 2017. The visual object tracking vot2017 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision*. 1949–1972.

[39] Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Luka Cehovin, Gustavo Fernandez, Tomas Vojir, Gustav Hager, Georg Nebehay, and Roman Pflugfelder. 2015. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE international conference on computer vision workshops*. 1–23.

[40] Matej Kristan, Jiri Matas, Aleš Leonardis, Tomas Vojir, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka Čehovin. 2016. A Novel Performance Evaluation Methodology for Single-Target Trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38, 11 (Nov 2016), 2137–2155. https://doi.org/10.1109/TPAMI.2016.2516982

[41] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 23.

[42] Robert LiKamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl. 2013. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 69–82.

[43] William Lotter, Gabriel Kreiman, and David Cox. 2016. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104* (2016).

[44] Chaochao Lu, Michael Hirsch, and Bernhard Schölkopf. 2017. Flexible Spatio-Temporal Networks for Video Prediction. In *Proc. of CVPR*.

[45] Muthucumaru Maheswaran, Shoukat Ali, HJ Siegal, Debra Hensgen, and Richard F Freund. 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*. IEEE, 30–44.

[46] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. 2018. Diffy: a Déjà vu-Free Differential Deep Neural Network Accelerator. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 134–147.

[47] Michael Mathieu, Camille Couprie, and Yann LeCun. 2015. Deep multi-scale video prediction beyond mean square error. *Proc. of ICLR* (2015).

[48] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *Proc. of ISCA*.

[49] Viorica Patraucean, Ankur Handa, and Roberto Cipolla. 2015. Spatio-temporal video autoencoder with differentiable memory. *Proc. of ICLR Workshop* (2015).

[50] Inc. Qualcomm Technologies. 2018. Qualcomm Snapdragon Heterogeneous Compute SDK Documentation and Interface Specification: Documentation and Interface Specification. (2018).

[51] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. 2014. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604* (2014).

[52] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[53] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proc. of CVPR*.

[54] Iain E Richardson. 2004. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons.

[55] Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. Computation reuse in DNNs by exploiting input similarity. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 57–68.

[56] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv preprint arXiv:1811.02883* (2018).

[57] Chenguang Shen, Supriyo Chakraborty, Kasturi Rangan Raghavan, Haksoo Choi, and Mani B. Srivastava. 2013. Exploiting Processor Heterogeneity for Energy Efficient Context Inference on Mobile Phones. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '13)*. ACM, New York, NY, USA, Article 9, 5 pages. https://doi.org/10.1145/2525526.2525856

[58] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. 2015. Unsupervised learning of video representations using lstms. In *International conference on machine learning*. 843–852.

[59] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. 2016. Anticipating visual representations from unlabeled video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 98–106.

[60] Vedran Vukotić, Silvia-Laura Pintea, Christian Raymond, Guillaume Gravier, and Jan C Van Gemert. 2017. One-step time-dependent future video frame prediction with a convolutional encoder-decoder neural network. In *International Conference on Image Analysis and Processing*. Springer, 140–151.

[61] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[62] Ryo Watanabe, Masaaki Kondo, Masashi Imai, Hiroshi Nakamura, and Takashi Nanya. 2007. Interactive presentation: Task scheduling under performance constraints for reducing the energy consumption of the GALS multi-processor SoC. In *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 797–802.

[63] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4820–4828.

[64] Haichuan Yang, Yuhao Zhu, and Ji Liu. 2019. ECC: Energy-Constrained Deep Neural Network Compression via a Bilinear Regression Model. *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[65] Haichuan Yang, Yuhao Zhu, and Ji Liu. 2019. Energy-Constrained Compression for Deep Neural Networks via Weighted Sparse Projection and Layer Input Masking. *International Conference on Learning Representations (ICLR)* (2019).

[66] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. 2008. Improving MapReduce performance in heterogeneous environments.. In *Osdi*, Vol. 8. 7.

[67] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[68] Yiren Zhou, Sibo Song, and Ngai-Man Cheung. 2017. On classification of distorted images with deep convolutional neural networks. *arXiv preprint arXiv:1701.01924* (2017).

[69] Yuhao Zhu, Matthew Halpern, and Vijay Janapa Reddi. 2015. Event-based scheduling for energy-efficient qos (eqos) in mobile web applications. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 137–149.

[70] Yuhao Zhu and Vijay Janapa Reddi. 2013. High-performance and energy-efficient mobile web browsing on big/little systems. In *IEEE International Symposium on High Performance Computer Architecture*. 13–24.

[71] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. 2018. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. In *Proc. of ISCA*.