Banner: An Image Sensor Reconfiguration Framework for Seamless Resolution-based Tradeoffs

Jinhan Hu, Alexander Shearer, Saranya Rajagopalan, Robert LiKamWa
Arizona State University
Tempe, Arizona
jinhanhu,acshear1,srajag25,likamwa@asu.edu

ABSTRACT

Mobile vision systems would benefit from the ability to situationally sacrifice image resolution to save system energy when imaging detail is unnecessary. Unfortunately, any change in sensor resolution leads to a substantial pause in frame delivery – as much as 280 ms. Frame delivery is bottlenecked by a sequence of reconfiguration procedures and memory management in current operating systems before it resumes at the new resolution. This latency from reconfiguration impedes the adoption of otherwise beneficial resolution-energy tradeoff mechanisms.

We propose Banner as a media framework that provides a rapid sensor resolution reconfiguration service as a modification to common media frameworks, e.g., V4L2. Banner completely eliminates the frame-to-frame reconfiguration latency (226 ms to 33 ms), i.e., removing the frame drop during sensor resolution reconfiguration. Banner also halves the end-to-end resolution reconfiguration latency (226 ms to 105 ms). This enables a more than 49% reduction of system power consumption by allowing continuous vision applications to reconfigure the sensor resolution to 480p compared with downsampling 1080p↓480p, as measured in a cloud-based offloading workload running on a Jetson TX2 board. As a result, Banner unlocks unprecedented capabilities for mobile vision applications to dynamically reconfigure sensor resolutions to balance the energy efficiency and task accuracy tradeoff.

CCS CONCEPTS

• **Computer systems organization** → *Sensors and actuators.*

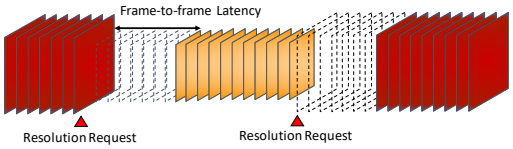
KEYWORDS

Operating systems; Energy efficiency; Reconfiguration; Resolution-based tradeoff; Efficient visual computing; Device drivers

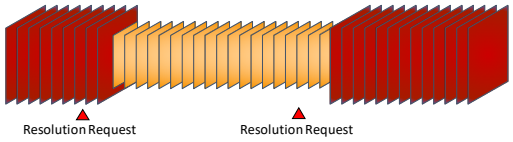
ACM Reference Format:
Jinhan Hu, Alexander Shearer, Saranya Rajagopalan, Robert LiKamWa. 2019. Banner: An Image Sensor Reconfiguration Framework for Seamless Resolution-based Tradeoffs. In *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19), June 17–21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307334.3326092

MobiSys '19, June 17–21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6661-8/19/06…$15.00
https://doi.org/10.1145/3307334.3326092



(a) In legacy systems, any change in sensor resolution leads to a substantial pause in frame delivery.



(b) Banner completely removes frame-to-frame latency for reconfiguring sensor resolution.

Figure 1: Compared with current systems, Banner enables rapid and seamless sensor resolution reconfiguration.

1 INTRODUCTION

The high energy consumption of visual sensing continues to impede the future of mobile vision in which devices will continuously compute visual information from sensory data, e.g,. for visual personal assistants or for augmented reality (AR). While vision algorithms continue to improve in task accuracy and speed, mobile and wearable vision systems fail to achieve sufficient battery life when vision tasks are continuously running. Continuous video capture drains the battery of Google Glass in 30 minutes [4].

It is well known that a common culprit is the energy-expensive traffic of image data [20, 22]. Transferring high resolutions at high frame rates draws substantial power consumption from the analog-digital conversion, the sensor interface transactions, and the memory usage. Simply capturing 1080p frames at 30 frames per second consumes more than 2.4 W of system power measured on a MOTO Z smartphone. However, capturing and displaying 480p frames only consumes 1.3 W of system power.

Image resolution can create an interesting tradeoff for visual tasks: low resolution promotes low energy consumption, while high resolution promotes high imaging fidelity for high visual task accuracy. For example, as we explore with our AR marker-based pose estimation case study (§1.1), lower resolutions suffice when an AR marker is close, but high resolutions are needed when the AR marker is far away or small. This tradeoff has been explored by several visual computing system works including marker pose estimation, object detection, and face recognition [5, 11, 12, 15, 22, 26, 31, 33, 39]. We too advocate that mobile vision systems

should be able to benefit from the ability to situationally sacrifice image resolution to save system energy when imaging detail is unnecessary.

Unfortunately, any change in sensor resolution leads to a substantial pause in frame delivery. This is illustrated in Figure 1a. We measure that reconfiguring sensor resolution in the Android OS prevents the application from receiving frames for about 267 ms, the equivalent of dropping 9 frames (working at 30 FPS) from vision processing pipelines [15]. Consequently, computer vision applications don't change resolutions at runtime, despite the significant energy savings at lower resolutions. For example, Augmented Reality applications such as "Augment" and "UnifiedAR" constantly work at 1080p, drawing 2.7 W of system power.

Thus, in this paper, we target image sensor *resolution reconfiguration latency* as a chief impediment of energy-efficient visual systems. Referring to [15], we break the resolution reconfiguration latency into two types of latency. *End-to-end reconfiguration latency* is the time between an application's request to change resolution and the time the application receives a frame of the new resolution. *Frame-to-frame latency* is the interval between two frames provided to the application in which the latter frame is configured at the new resolution.

The problem of long resolution reconfiguration latency is common across all mobile platforms, as we measured on different devices. In the Android OS, there is a 400 ms end-to-end reconfiguration latency [15]. In the Linux V4L2 media framework, we observe a 260 ms end-to-end reconfiguration latency. End-to-end reconfiguration latency in iOS also takes around 400 ms, as measured by timestamping a simple video capturing application we built in Xcode [16]. Similarly, end-to-end reconfiguration latency in Gstreamer with NVIDIA Libargus occupies more than 300 ms.

Almost all of the resolution reconfiguration latency originates from the operating system; at the sensor level, hardware register values are effective by the next frame [2, 27, 28, 34, 38]. We proposed several alternatives at the Android framework and HAL level for discussion in our pervious work [15]. However, from deeper understanding of the Android OS, we find that the problem stems from the lower level system, i.e., media frameworks in the kernel. The underlying issue is that the kernel's media frameworks require user space applications to frequently invoke a sequence of expensive system calls in order to request a new sensor resolution.

Our study of the media frameworks exposes several key insights. First, the current streaming pipeline needs to be preserved during resolution reconfiguration. Frames already captured at the previous resolution are useful and need to be read out. Second, resolution change should also be immediately effective in the next capture. This capture will be available after moving through the pipeline. Third, synchronizing the resolution of frame buffers across the system stack is expensive and should be avoided. As it stands, media frameworks require the application to initiate expensive system calls to repeatedly allocate memory for the frame buffers.

To exploit these key insights, we design Banner: a system solution for rapid sensor resolution reconfiguration. Banner revolves around two techniques. *Parallel reconfiguration* maintains video capture streams and schedules sensor reconfiguration in parallel while the application is processing the frame. *Format-oblivious memory management* removes repeated memory allocation from the reconfiguration procedure, avoiding expensive system calls initiated by the application. Using these techniques, Banner completely eliminates frame-to-frame latency, as illustrated in Figure 1b, allowing for seamless multi-resolution frame capture. Banner also achieves the minimum possible end-to-end reconfiguration latency, fundamentally bounded by the pipeline latency of frame readout (usually larger than two frames). In extreme cases, if the application requests only one buffer to be allocated (not allowed for video streaming) or if the system allows frames already captured to drop, the end-to-end reconfiguration latency can further be reduced in Banner.

Because of the unavailability of open-source camera drivers and camera host drivers for Android devices, our Banner prototype is implemented in the Linux kernel. However, we propose several suggestions in §6 to help developers who want to integrate Banner to the Android OS. We evaluate Banner's efficacy within the Linux V4L2 framework by running three workloads on a Jetson TX2 board with the Ubuntu system, including display-only, cloud-based offloading, and marker-based pose estimation. Our evaluation confirms that Banner completely eliminates frame-to-frame latency, even for workloads operating at 30 FPS. Furthermore, Banner creates a 54% reduction in end-to-end reconfiguration latency (from 226 ms to 105 ms).

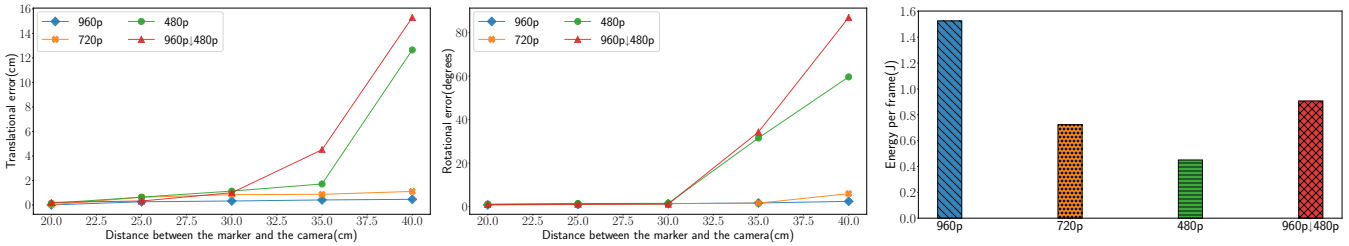
The reduction in reconfiguration latency results in a 49% power consumption reduction by reconfiguring the resolution from 1080p to 480p compared with computationally downsampling 1080p↓480p, measured on a Jetson TX2 board. While we implement and evaluate our design choices based on Linux V4L2, they can be generalized to other media frameworks, such as Gstreamer (which can capture videos from V4L2 devices), and Linux-based operating systems, including the Android OS.

Our contributions in this paper are as follows:

- We propose design strategies for media frameworks to reduce sensor resolution reconfiguration latency.
- We introduce Banner as a rapid sensor reconfiguration framework that eliminates frame-to-frame latency and halves end-to-end reconfiguration latency.
- We evaluate the effectiveness of reconfiguring sensor resolution dynamically to improve power efficiency for vision tasks, comparing with downsampling.

Altogether, Banner will unlock new classes of vision algorithms that can balance the resolution-based energy efficiency and accuracy tradeoffs to maximize performance in a variety of continous mobile vision tasks.

The rest of the paper is structured as follows. §1.1 shows the motivation for reconfiguring resolution. §2 explains resolution reconfiguration procedure in common media frameworks. §3 elaborates the Banner design for tackling the inefficient sensor resolution reconfiguration problem. §4 introduces the Banner implementation in the Linux kernel. §5 demonstrates the effectiveness of Banner for improving energy efficiency in three workloads. §6 discusses the limitations of Banner and our future work. §7 compares Banner to related works.



(a) Low resolution maintains a low translation error when marker is close to camera.

(b) Low resolution maintains a low rotation error when marker is close to camera.

(c) System energy consumption at 480p is 70% and 50% less than 960p and downsampling.

Figure 2: In our marker-based pose estimation case study, task accuracy (translation and rotation error) can be maintained and system energy consumption (energy per frame) can be reduced by 70% if sensor resolution is reconfigured from 960p to 480p when the distance between the marker and the camera is reduced from 35 cm to 20 cm.

1.1 Case Study for Resolution-driven Tradeoffs

To motivate our work, we present a case study around a marker-based pose estimation application running on a Moto Z mobile phone. Marker-based pose estimation forms the foundation for many AR frameworks, including Vuforia [37], ARCore [9], and ARKit [3] for image-based tracking. Our exploration of marker-based pose estimation allows us to analyze the resolution-based energy and accuracy tradeoff in mobile vision tasks. The pose estimation application uses an ORB feature detector, Flann-based matcher, and Perspective-n-Point algorithm to detect keypoints in an image frame, match keypoints with model descriptors, and estimate the position of the virtual camera against the physical environment respectively on a frame-to-frame basis.

The energy efficiency is characterized by the power traces acquired from the Trepn Profiler and the number of frames processed per second measured at different resolutions. To evaluate task accuracy, we use the MSE rotation and translation vector errors compared with the "ground truth" acquired from the highest resolution. Prior work [15] has already demonstrated that based on the distance and viewing angle between the camera and the marker, sensor resolution needs to be actively reconfigured to balance efficiency and performance. Similarly, results in Figure 2 show that, by reconfiguring the sensor from 960p to 480p while the sensor is approaching the marker from 35 cm to 20 cm, the task accuracy can be maintained (Figure 2a and 2b) and a 70% energy consumption reduction can be achieved (Figure 2c).

As an alternative to changing sensor resolution, the system can computationally downsample the frames to reduce the computational workload of the vision algorithm. However, results in Figure 2c show that, capturing at 480p costs almost 50% less energy than computational downsampling 960p↓480p, not to mention the higher task accuracy.

In conclusion, physically reconfiguring the sensor resolution is the most viable way to balance the resolution-based energy efficiency and task accuracy tradeoff for continuous mobile vision tasks. However, sensor resolution reconfiguration is limited by a substantial latency.

2 UNDERSTANDING THE RESOLUTION RECONFIGURATION LATENCY

In this section, we elaborate on how user applications request different sensor resolutions using the Video4Linux2 (V4L2) framework. The V4L2 framework provides APIs for applications to manipulate cameras on Linux. V4L2 is commonly used by almost all Ubuntu desktops and Android devices built upon the Linux system.

In V4L2, image sensor resolution reconfiguration follows a strict sequential procedure. This sequential procedure leads to a substantial amount of end-to-end and frame-to-frame reconfiguration latency, which impedes the ability for applications to utilize resolution-based energy tradeoffs. Through this section, we explore the V4L2 implementation on an NVIDIA Jetson TX2 board with an ON Semiconductor AR0330 sensor.

V4L2 system architecture: In the V4L2 framework, there are four main driver modules in the kernel that collaborate to provide camera services. The **V4L2 driver** is responsible for exposing camera control operations to the user application, such as opening the V4L2 camera or setting its exposure or brightness. The **camera host driver**, which implements the V4L2 driver and V4L2 camera interfaces, is responsible for ensuring the proper input and output format for frames flowing between the camera and the memory, as well as starting and stopping the reception of camera frame data. The **video buffer driver**, which is a helper to the V4L2 driver, is responsible for allocating and deallocating buffers with proper sizes for corresponding resolution requests. The **camera driver**, which communicates with the camera hardware, is responsible for setting up the camera for the requested output.

Before the application can start streaming with the V4L2 camera devices, it needs to request at least a pair of buffers to store and process the captured frames. The buffer ownership transfer is realized by dequeuing buffers `ioctl(VIDIOC_DQBUF)` and queuing buffers `ioctl(VIDIOC_QBUF)` between the application and the camera host driver. The application dequeues a buffer when the capture is completed by the image sensor. The application queues a buffer back for sensor capture after the processing on it is done – the application relinquishes control of the buffer. Depending on the needs of the imaging pipeline, the application can require more

Reconfiguration operation	Average execution time
Stop streaming	75 ms
Initialize device	31 ms
Start streaming	72 ms

Table 1: Expensive operations and their average cost in current sensor resolution reconfiguration procedure measured on the TX2/AR0330 setup.

buffers, such that multiple pipeline stages can simultaneously address buffers. All buffers ready for applications to read are stored in the camera host driver. Typically, only one buffer is transferred to the application at a time.

2.1 Reconfiguration is a sequential process

Once the video stream has been started, V4L2 requires the application to reconfigure sensor resolution in a sequence of steps. Notably, each subsequent step in the sequence invokes a different subsystem, creating synchronization issues. We illustrate this sequential procedure in Figure 4a and detail it here.

(1) The application initializes a resolution request while the camera is capturing.
(2) The application calls V4L2 `ioctl(VIDIOC_STREAMOFF)`, which is implemented in the camera host driver and the camera driver to turn off current working streams. This step takes around 75 ms.
(3) The application calls `munmap()`, which is implemented in the video buffer driver to deallocate the memory. This step takes less than 1 ms.
(4) The application calls V4L2 `ioctl(VIDIOC_S_FMT)`, which is implemented in the camera host driver and the camera driver to set the sensor's output format.
(5) The application calls V4L2 `ioctl(VIDIOC_REQBUFS)` and `mmap()`, which are implemented in the video buffer driver to request, allocate, and map new sets of buffers. Together with step 4, initializing the device takes around 31 ms.
(6) The application finally calls V4L2 `ioctl(VIDIOC_STREAMON)`, which is implemented in the camera host driver and the camera driver to set the input and output format of the channel and then start the video stream. This step takes around 72 ms.
(7) The first frame at new resolution is returned after a pipeline latency, typically several frame times later, depending on the pipeline depth.

Table 1 shows the latency costs of several expensive operations in the sensor resolution reconfiguration procedure measured on the TX2/AR0330 setup.

2.2 Resolution Synchronization Creates Latency

Throughout the reconfiguration process, there are several strict resolution synchronizations among the camera host driver, the video buffer driver, and the camera driver, each of which introduces a substantial reconfiguration latency.

First, resolution synchronization between the video buffer driver and the camera driver is established by requesting buffer size based on specific sensor format. This synchronization ensures that there will be enough frame buffer space to hold complete frames. If the syscall `ioctl(VIDIOC_S_FMT)` is called to set the sensor resolution, `ioctl(VIDIOC_REQBUFS)` and `mmap()` also need to be called for a new set of buffers.

Second, resolution synchronization between camera host driver and camera driver is established by updating the camera driver host state based on the camera's format. If `ioctl(VIDIOC_S_FMT)` is called to set the sensor resolution, the input state of camera host driver also needs to be reconfigured. This synchronization ensures that the video input module on board has the proper format to receive frames flowing from the camera.

Third, the previous two synchronizations force a resolution synchronization between the camera host driver and video buffer driver. That is, if the system requires a new set of buffers, the output state of camera host driver also needs to be reconfigured.

By synchronizing resolution among these drivers, the camera service ensures correct capture, delivery, and render of image frames. But this strong resolution coupling among drivers creates bottlenecks; if an application requests a new resolution, the whole configuration procedure described above in §2.1 will be invoked, creating a substantial latency.

2.3 Reconfiguration Latency Drops Frames

As shown in Figure 7, the overall end-to-end reconfiguration and the frame-to-frame latency are both about 230 ms in the legacy V4L2 framework, as we measured on the TX2/AR0330 system. For a camera running at 30 FPS, a 230 ms frame-to-frame reconfiguration latency is equivalent to the system dropping 8 camera frames. In addition, the legacy V4L2 framework abandons all captured frames that are stored in buffers once the application requests a new sensor resolution. Thus, depending on how many buffers are requested by the application (N), the number of frames dropped could be $N + 8$. The number of requested buffers (N) must be larger than 2 (typically 3 or 4) [19].

We see resolution reconfiguration latency manifested on all devices we tested. We measure that end-to-end resolution reconfiguration latency in Android and iOS devices both consume about 400 ms. A fast sensor resolution reconfiguration solution needs to be introduced to media frameworks so that frame-critical computer vision applications on top of them can frequently reconfigure the sensor resolution to improve energy efficiency.

2.4 Design Guidelines

We declare the following insights for inspiring the design of a rapid and seamless sensor resolution reconfiguration system:

Preserve the pipeline of existing frames: Frames already captured and stored in the pipeline are still meaningful. The legacy V4L2 framework abandons those frames to fulfill the new resolution request immediately. On the contrary, the Android OS will issue the new resolution request only after pipelined frames are processed and delivered properly. For some visual tasks – including marker-based pose estimation – every frame is critical to task performance because of the potential negative influence on task accuracy and user experience. The sensory data should be continuous, i.e., frame

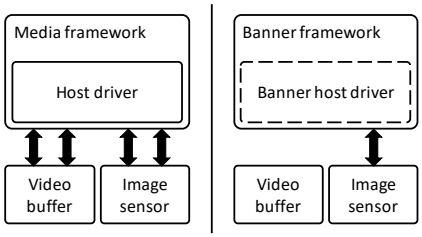


Figure 3: Banner helps the application reduce the number of required system calls to reconfigure sensor resolution to one `ioctl(VIDIOC_RECONFIGURE)` call, instead of multiple systems calls: `mmap()`, `munmap()`, `ioctl(VIDIOC_STREAMON)`, and `ioctl(VIDIOC_STREAMOFF)`.

drop is unacceptable. The system should find a way to maintain current streams while reconfiguring the sensor for new resolution.

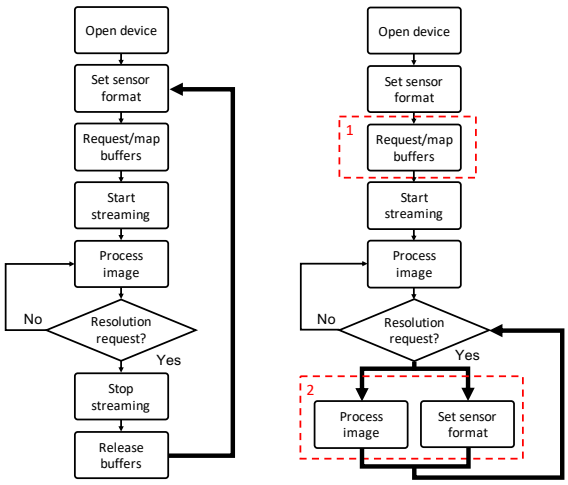
Resolution change should be immediately effective in the next capture: Sensor register changes can be effective in the next capture, as is done for setting up different exposure time for consecutive capture requests in HDR mode [2, 34, 38]. Similarly, the system should reconfigure related sensor registers immediately and asynchronously once there is a new resolution request. This would allow applications to expect and utilize the prompt resolution change.

Minimize synchronization across the video system stack, while ensuring correct sensory data: Resolution synchronizations among different driver modules lead to repeated sequential reconfigurations every time there is a new resolution request which causes huge amount of latency. In addition, resolution synchronizations trigger some expensive and redundant system calls initiated by the application, including `mmap()`. As long as the application knows the resolution of the frames it is processing and the sensor knows the resolution for each frame it is capturing, the data will be correctly delivered and interpreted. We argue that buffer size synchronization between the sensor and the application is unnecessary. Memory management can be oblivious to format.

3 DESIGN OF BANNER

Built on these derived design guidelines, we design Banner to address the resolution reconfiguration latency problem in the legacy V4L2 framework. Banner is a fast sensor resolution reconfiguration framework that can provide frames continuously, even between two frames at different resolutions. While we design Banner to interoperate with the V4L2 framework, the underlying concepts are generic to all media frameworks. Compared to resolution reconfiguration in today's system, Banner halves the end-to-end reconfiguration latency and completely removes the frame-to-frame reconfiguration latency, i.e., no frame drops. As a result, Banner unlocks a variety of continuous mobile vision applications to control their image sensors for desired resolutions. This allows new potential to balance energy efficiency and accuracy tradeoffs.

In particular, Banner employs two key techniques: parallel reconfiguration and format-oblivious memory management. Parallel



(a) Resolution reconfiguration in legacy V4L2 **(b) Resolution reconfiguration in Banner**

Figure 4: In Banner, most of the sequential procedures are avoided for reconfiguring sensor resolution. (1) Banner avoids repeated memory allocation; (2) Banner sets sensor format in parallel with user application.

reconfiguration aims at reconfiguring the sensor while the application is processing frames for the previous resolution such that the reconfiguration latency is hidden. Format-oblivious memory management aims at maintaining a single set of frame buffers – regardless of resolution – to eliminate repeated invocation of expensive memory allocation system calls.

System Overview: Banner is a media framework that allows applications to request sensor resolution reconfigurations with seamless frame delivery. It exposes a system call to the application and resides in the kernel as a camera host driver to interact with the video buffer driver and the camera driver, as shown in Figure 3. Banner minimizes the required number of system calls for vision applications to reconfigure the sensor resolution.

Figure 4b depicts the rapid sensor resolution reconfiguration procedure in our proposed Banner framework. When starting the stream, the system sets up the sensor, the sensor host, and the buffer with the highest supported resolution in a sequential procedure, as it does with the V4L2 framework. However, after the application requests a new resolution, Banner will not go through all steps in the sequential procedure again. Instead, Banner maintains the stream without reallocating buffers and then asynchronously reconfigures the sensor in parallel through only one `ioctl` call from the application. Frames at new resolution will be returned after reading out N frames (determined by the number of buffers requested) already captured with previous resolution. Resolution reconfiguration in Banner is rapid and continuous, i.e., without any frame drop. The procedure for stopping the capture and closing the camera follows the same sequential procedure in V4L2 framework.

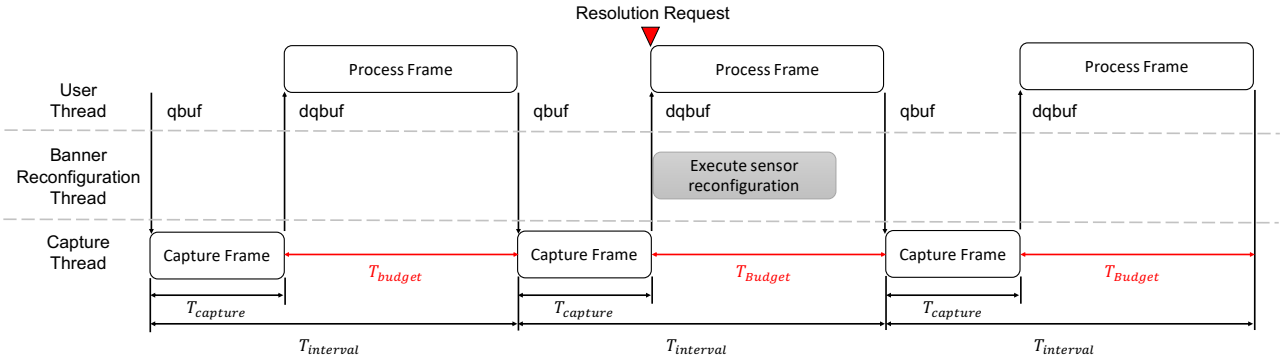


Figure 5: Banner reconfigures sensor resolution in parallel with application processing frames in the reconfiguration timing budget (a function of frame interval and capture time) such that reconfiguration latency can be hidden.

3.1 Parallel Reconfiguration

As we discussed in §2, resolution reconfiguration in the current V4L2 framework follows a strict sequential procedure. This sequential reconfiguration procedure introduces both a substantial end-to-end reconfiguration latency and a substantial frame-to-frame reconfiguration latency. In Banner, sensor resolution reconfiguration is completed in parallel while the application is processing frames. By doing so, the frame-to-frame resolution reconfiguration latency is fully hidden.

To achieve this, the parallel reconfiguration module is designed based on three considerations. First, the sensor is not always busy; there is an idle time between captures. Second, the reconfiguration thread cannot be interrupted, otherwise the end-to-end latency will be increased. Dequeuing a buffer signals that a capture is complete and queuing a buffer signals the next capture. The system should identify the right time to reconfigure sensor. Third, reconfiguration itself takes time, due to camera driver implementations and camera hardware limitations.

To resolve these considerations, *thread-level concurrency* can address the first and second considerations, while *a reconfiguration timing budget* can address the second and third considerations. Altogether, Banner can schedule the right time to reconfigure the sensor and trigger the next capture. The parallel reconfiguration strategy is illustrated in Figure 5.

3.1.1 Thread-level concurrency. The crux of the parallel reconfiguration is to utilize thread-level concurrency to reconfigure sensor resolution. In the current V4L2 framework, in addition to a main thread, there is a *capture thread* responsible for capturing frames. This capture thread is frozen until it is woken up by the application queueing a buffer for frame capture. The capture thread and the main application thread process in parallel. Although the sensor is busy capturing frames when the capture thread is awake, it is free for reconfiguration while the capture thread is frozen. For Banner, we design a *reconfiguration thread* that can work in parallel with the application thread. This thread processes reconfiguration requests while the application processes frames and the capture thread is frozen. We choose to create a reconfiguration thread, considering the latency penalty incurred by waking up the capture thread. Reconfiguration thread and the main thread are joined before they wake up the capture thread for the next capture. Banner uses atomic read/write to ensure thread safety.

3.1.2 Reconfiguration timing budget. The Banner reconfiguration thread cannot reconfigure the sensor when the capture thread is active. Therefore, a resolution reconfiguration timing budget needs to be defined for the reconfiguration thread to work with. We define resolution reconfiguration timing budget T_{budget} in Equation 1 as a function of frame interval $T_{interval}$ and capture time $T_{capture}$. $T_{interval}$ – the interval between consecutive frame captures – is defined by the application as the interval between two consecutive `ioctl(VIDIOC_QBUF)` calls from the application. $T_{interval}$ is typically held stable to ensure good user experience. $T_{capture}$ varies from frame to frame, influenced by the capture parameters such as the exposure time and resolution. T_{budget} is equal to frame interval $T_{interval}$ minus the required capture time $T_{capture}$, i.e.,

$$T_{budget} = T_{interval} - T_{capture} \quad (1)$$

It is important to ensure that sensor resolution reconfiguration is finished in the reconfiguration timing budget such that the reconfiguration thread is not interrupted by the wake up of the capture thread. Otherwise, capture at the new resolution will be delayed by another capture with the old resolution, which causes both end-to-end and frame-to-frame reconfiguration latency to be unpredictable.

In our implementation, we have observed that the reconfiguration timing budget is long enough that the reconfiguration latency can be completely hidden by the main application thread. That is, the frame-to-frame latency is eliminated. Seen from the application side, the frame rate is stable in Banner even between two frames at different resolutions. Based on our evaluation, we can maintain more than 30 FPS for an offloading application with only 10 ms reconfiguration timing budget. We note that different image sensors may require different amount of time to reconfigure the resolution.

That being said, theoretically, if an application operates at an unstably fast frame rate, then Banner could potentially delay the delivery of the frame after the resolution request. For example, this would be the case if the application performs a `memcpy` of the

frame to a memory location and immediately queues the buffer for a next capture. Still, in this case, Banner would improve the reconfiguration latency over the legacy V4L2 framework, which would delay frame delivery while executing the full reconfiguration procedure – complete with memory allocation.

3.2 Format-oblivious Memory Management

As explained in §2, the legacy V4L2 framework synchronizes frame buffer resolutions across all of its modules. Buffers are requested and mapped for a determined resolution before the camera can even start capturing. If the application requests another sensor resolution, the legacy V4L2 framework stops current streams, releases previous frame buffers, and allocates a fresh set of buffers. Thus, synchronizing the format can be very expensive for the resolution reconfiguration procedure.

We propose a format-oblivious memory management that removes resolution synchronization in the resolution reconfiguration procedure. Format-oblivious memory management reuses previously allocated buffers to store frames with different formats, as shown in Figure 6. This technique reduces the end-to-end reconfiguration latency and frame-to-frame reconfiguration latency by avoiding unnecessary system calls.

3.2.1 One-time buffer allocation. Instead of allocating frame buffers every time the application requests a new resolution, format-oblivious memory management only allocates buffers *once* when initializing the camera. To support all formats, the system can allocate for the highest supported resolution by the camera for reuse for any resolution.

Reusing buffers brings several benefits. First, repeated memory deallocation and allocation for different sensor resolutions are now completely avoided; `ioctl(VIDIOC_REQBUFS)` and `mmap()` are avoided. The system call `mmap()` is very time consuming as we discussed in §2. Second, current video streams are not discarded. The system calls for turning on and off the video streams – `ioctl(VIDIOC_STREAMON)` and `ioctl(VIDIOC_STREAMOFF)` respectively – are avoided. Both of them consume tens of milliseconds. Third, since there is only one format at the receiving end, the system doesn't need to set the output state of camera host driver for reconfiguration.

3.2.2 Format-oblivious frame delivery. Format-oblivious memory management delivers the frame to the application not based on the payload calculated by the sensor format, but based on how many bytes are used. When the application requests another resolution, Banner passes the format information to the camera driver and the camera host driver appropriately. As the system needs to maintain the current pipeline of frames, there will be a resolution discrepancy among the frames already captured and the frames to be captured in the new configuration.

Banner solves this problem easily by delivering the frames according to how many bytes are used. Banner and the frame itself will provide enough format information for the application to interpret frames. We argue that as long as the application and sensor know the format at the appropriate times, the frame can be correctly captured, delivered, and interpreted.

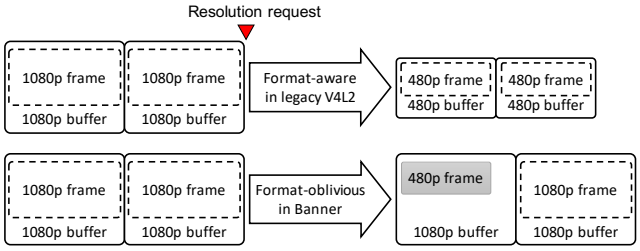


Figure 6: After a resolution request, format-oblivious memory management in Banner reuses buffers previously allocated and stores newly configured frames, despite potential format mismatch.

Format-oblivious memory management can be realized without any modification to the video buffer driver. The only potential limitation of this approach is that it allocates more memory than needed for low resolution configurations. For example, when configured for 480p resolutions, the frame buffer will occupy the memory footprint of a 1080p frame buffer (6 MB for 3 frames). However, on modern mobile systems, we do not foresee this as particularly problematic; most modern phones have at least 1 GB of RAM. More importantly, the additional buffer allocation does not increase system power, as DDR power consumption is governed by data rate, not allocation. We confirm this in our evaluation.

4 IMPLEMENTATION

Our Banner prototype is built by modifying the V4L2 media framework in the upstream NVIDIA Tegra TX2 Linux kernel 4.4, L4T 28.2.1. In this implementation, **the application reconfigures sensor resolution rapidly through only one `ioctl` system call**.

4.1 Parallel Reconfiguration

The goal of Banner's reconfiguration policy is to utilize idle time in kernel space to change the format of an image sensor. After capturing and processing a frame, the kernel camera host driver returns to an idle state until the next capture. Knowing that the kernel is idle, Banner can use this time to send commands that change the sensor's format and perform any state changing on the camera host driver side. This sufficiently performs the operations of reconfiguring the sensor resolution. Resolution reconfiguration is initialized by an `ioctl(VIDIOC_RECONFIGURE)` call, from the application which will set a sensor resolution format that is passed from user space to the camera host driver object. This system call will then immediately spawn a kernel thread to perform the reconfigure operation. We spawn a single thread to perform our reconfiguration as the overhead of spawning multiple times for more parallelism made reconfiguration slower overall. Setting camera host driver states is an immediate operation. The only part of the reconfiguration process that takes significant time is configuring the camera hardware.

4.1.1 Configure sensor device. The sensor configuration call changes the state of the camera device. The camera driver module then controls the image sensor directly by making I^2C or other bus calls. The time that configuring the sensor takes will vary from sensor

to sensor as each sensor will have a different protocol for setting sensor format.

4.1.2 Update camera host driver state. Updating the camera host driver's state will prepare it to capture frames at a new resolution. The camera host driver state must be updated immediately after the sensor is reconfigured, as the next captured frame will be at the sensor's new resolution. If it is not done, the next frame will be returned with the old resolution and be interpreted improperly at the application level. The next `ioctl(VIDIOC_QBUF)` operation will use the settings set here to capture a frame. This will also set the input for the frame size of buffers as well as the values required to calculate the size of buffer, so that the application knows how many bytes to read for the frame.

4.2 Format-oblivious Memory Management

An important optimization in Banner is to reuse memory buffers, as making `mmap()` and `munmap()` calls can take tens of milliseconds based on the frame size. When initializing the device, after calling `ioctl(VIDIOC_REQBUFS)`, the buffers returned should be allocated to the maximum size that will be used by the application. While reusing the buffers does consume extra memory when the frame size is smaller than maximum, it allows Banner to save reconfiguration latency; the `mmap()` and `munmap()` process does not need to be repeated.

`mmap()` allocates shared memory between the camera device and the application level. Shared memory allows the camera device driver to write frames into the buffer and the application to read from the same address in memory. The shared memory will contain information about the bytes used inside of the buffer, the state (if the buffer is ready to be read from the application level), and the raw frame data. The user application will use the buffer state to know the length of bytes to read out into its own buffer.

4.3 User Application Library

Banner exposes the sensor resolution reconfiguration API to the user application as a V4L2 system call. User applications can call the Banner API, just as they use V4L2 to start video streaming. We use the V4L2 capture example provided in [18] as a code base for our testing. The example code opens the camera device and initializes all memory needed for capture per the V4L2 specification. The example code then starts a capture loop that will run until a frame count has passed. This capture loop uses the `select` system call to wait until the video buffer driver signals that the buffers are ready for reading. The application takes ownership of the buffer by calling `ioctl(VIDIOC_DQBUF)` and then copies the shared memory to an application buffer before returning it with `ioctl(VIDIOC_QBUF)`.

Our modifications to the example code were minimal.

(1) When the application initializes the camera, it counts the number of frame buffers allocated. This count is saved for future reference, as it is equal to the number of frames in any given pipeline.
(2) Immediately after a `select`, on any frame, the application can call `ioctl(VIDIOC_RECONFIGURE)` with the reconfigure resolution target.
(3) After a reconfiguration call, the application starts counting frames returned in the main-loop until the captured frames at previous resolution are read out; at this point, the application's resolution is reconfigured to the new resolution.
(4) From this frame onward the frames returned by the driver will be the new resolution.

4.3.1 OpenCV hook. When working with Banner in OpenCV, we take our raw frames from the V4L2 capture. OpenCV requires frames to be in BGR format, but the V4L2 camera returns UYVY. To convert frames into a format that OpenCV can manipulate, we use a modified CUDA function from NVIDIA. The function originally converted YUYV to RGB, but we manipulated it to convert UYVY to BGR by reordering the image input and output planes Once we have our BGR frame, it is a 1 dimensional array and still not in a form for OpenCV to work with. To fix this we call the constructor for `Mat`, OpenCV's basic object for handling images. We take care to use the correct parameters for resolution, pixel size, plane count, and our frame data. From there, we can use any OpenCV function to operate on the image, such as `resize`, `imshow`, and `BFMatcher`.

5 EVALUATION

We evaluate Banner within the V4L2 framework on an NVIDIA Jetson TX2 board with an ON Semiconductor AR0330 sensor. This Jetson TX2 board has a Quad ARM A57 processor. It is one of the most popular embedded computing devices.

The evaluation answers three questions: *(i)* How much reconfiguration latency did Banner reduce when reconfiguring sensor resolution? *(ii)* How much power efficiency can be gained by reconfiguring sensor resolution dynamically and rapidly with Banner? *(iii)* What does fast sensor resolution reconfiguration mean to computer vision applications?

5.1 Evaluation Methodology

Workloads To evaluate and validate the effectiveness of Banner for reconfiguring sensor resolution in a variety of vision tasks, we choose three applications integrated with OpenCV. The first application only displays frames at 25 FPS. This application gives us preliminary results of the effectiveness of Banner. The second application offloads frames to a desktop server through a direct connection at 15 FPS. This application demonstrates Banner's usage in cloud-based vision applications. The third application implements the same marker-based pose estimation as we described in §1.1, running at 15 FPS. It verifies that Banner is effective for our target application (Augmented Reality).

All three applications cycle through a set of supported resolutions: 1920x1080, 1280x720, and 640x480. To compare Banner reconfiguration against computational downsampling, we use OpenCV `resize()` function to downscale 1080p frames to 480p (represented as 1080p↓480p). The frame rate is set to be constant across different resolutions in all applications, bounded by the frame rate in the highest resolution, with the help from dynamic CPU and GPU clock scaling.

We measure the metrics described below:

As we defined in §1, resolution reconfiguration latency includes *end-to-end reconfiguration latency* which describes how long it takes for the application to receive the first new frame after a resolution

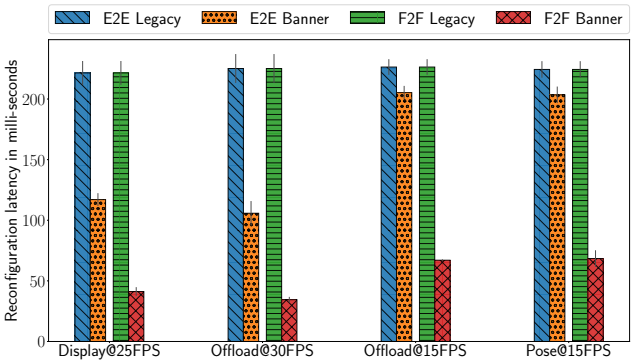


Figure 7: Banner reduces end-to-end (E2E) resolution reconfiguration latency and removes frame-to-frame (F2F) latency in all three workloads comparing with legacy in V4L2 framework.

is requested, and *frame-to-frame latency* which indicates the interval during which an application receiving no frames after a resolution is requested. Both latencies are measured by retrieving system timestamps at the application level. In each application, they are measured and averaged across 99 samples, i.e., 99 resolution reconfigurations.

Power consumption is monitored by retrieving the power rail system files on the Jetson TX2 board. These files include SYS_SoC which monitors the power consumption of the main Tegra core, SYS_DDR which monitors the power consumption of the LPDDR4, SYS_CPU which monitors the power consumption of the ARM processor, and SYS_GPU which monitors the power consumption of the Pascal GPU. Power rail system files are written automatically by the system at 400 KHz. In our evaluation, we measured the power after the application ran at a steady state, in order to minimize the variance. In each measurement, we acquired and averaged 600 readings for each power rail system file.

5.2 Resolution Reconfiguration Latency Reduced by Banner

With parallel reconfiguration and format-oblivious memory management, **Banner completely eliminates the frame-to-frame latency in all three workloads and is able to halve the end-to-end reconfiguration latency**, as shown in Figure 7.

In the display workload (at 25 FPS), the average end-to-end reconfiguration latency is reduced by 47% (from 222 ms to 117 ms) and the average frame-to-frame latency is reduced by 82% (from 222 ms to 41 ms). In the slower offloading workload (at 15 FPS), the average end-to-end reconfiguration latency is reduced by 9% (from 226 ms to 205 ms) and the average frame-to-frame latency is reduced by 70% (from 226 ms to 67 ms). In the pose estimation workload (at 15 FPS), the average end-to-end reconfiguration latency is reduced by 10% (from 225 ms to 203 ms) and the average frame-to-frame latency is reduced by 70% (from 225 ms to 67 ms). In addition, in the faster offloading sub-workload (at 30 FPS), the average end-to-end reconfiguration latency is reduced by 54% (from 226 ms to 105 ms) and the average frame-to-frame latency is reduced by 85% (from 226 ms to 34 ms).

We have several observations from these results. First, end-to-end reconfiguration latency and frame-to-frame latency are equivalent in the legacy V4L2 framework. This is because the frames stored in the capture queue are abandoned once there is a new resolution request. If those frames need to be read out and processed before the start of resolution reconfiguration – as they are in the Android OS [15] – the end-to-end reconfiguration latency can be even larger.

Second, the average end-to-end reconfiguration latency and frame-to-frame the latency in legacy V4L2 framework are stable across all three workloads because they all go through the same procedure, though they still have larger standard deviation compared to Banner.

Third, end-to-end reconfiguration latency and frame-to-frame latency are predictable in Banner because they depend on the frame rate. In Banner, the first frame at new resolution will be received after N frame intervals, where N is the number of frames already captured and stored in the buffer queue for the previous resolution (discussed in §2.3). In our case, there are three buffers (N) requested and thus, three captures are stored in the buffer queue whenever a new resolution is requested by the application. Therefore, the end-to-end reconfiguration latency in Banner is around three frame intervals. These intervals cause the end-to-end reconfiguration latency to be around 120 ms in Display@25FPS and 200 ms in Pose@15FPS, as shown in Figure 7. Frame-to-frame latency in Banner is equal to the inverse of the processing frame rate. The application will receive continuous frames at the same frame rate without noticing the resolution reconfiguration procedure. In other words, Banner eliminates frame drops.

5.3 Power Efficiency Improved by Banner

Table 2 demonstrates that **rapid sensor resolution reconfiguration with Banner enables a substantial power efficiency improvement**. We note the following observations.

First, the resolution-based power efficiency improvement is generic to vision tasks and components across the system (e.g., SoC, DDR, CPU, GPU). In all three evaluated workloads, the choice of sensor resolution influences the power consumption of all stages in the image processing pipeline, including data movement, storage, and processing. The results in Table 2 show that SoC, GPU, DDR, and CPU all benefit from processing lower resolution frames in terms of power savings.

Second, the power efficiency improvement is substantial as the sensor resolution drops. In legacy V4L2, the combined power consumption is reduced by 62%, 60%, and 42% as sensor resolution is reduced from 1080p to 480p in display, offload, and pose estimation workloads respectively. Thus, the power efficiency of a mobile vision task can be significantly improved if the system allows dynamic sensor reconfiguration when it can sacrifice resolution.

Third, reconfiguring sensor resolution physically is much more power efficient than other alternatives, i.e., computational downsampling. 1080p↓480p@FPS in Table 2 shows the power consumption of downsampling 1080p frames to 480p in the legacy V4L2 framework. Comparing with reconfiguring sensor resolution physically to 480p in Banner, downsampling consumes 43%, 49%, and 16% more power in display, offload, and pose estimation accordingly.

Workload	Resolution@FPS	Legacy V4L2					Banner				
		SoC	GPU	DDR	CPU	Total	SoC	GPU	DDR	CPU	Total
Display-only	1920x1080@25	1149	910	1869	2460	6388	1149	918	1839	2190	6096
	1280x720@25	1073	611	1727	1076	4487	1073	559	1704	1100	4436
	640x480@25	617	306	826	691	2440	669	306	860	681	2516
	1080p↓480p@25	1078	613	1690	1035	4416	N/A	N/A	N/A	N/A	N/A
Cloud-based offloading	1920x1080@15	1073	536	1629	1967	5205	1146	544	1638	2027	5355
	1280x720@15	688	230	863	617	2398	700	230	856	630	2416
	640x480@15	617	230	702	540	2089	630	230	687	554	2101
	1080p↓480p@15	1073	382	1606	1032	4093	N/A	N/A	N/A	N/A	N/A
Marker-based pose estimation	1920x1080@15	1281	1701	1940	2524	7446	1149	1448	1875	2527	6999
	1280x720@15	1230	1058	1814	1271	5373	1225	987	1795	1203	5210
	640x480@15	1210	589	1635	928	4362	1150	540	1637	916	4243
	1080p↓480p@15	1227	850	1727	1260	5064	N/A	N/A	N/A	N/A	N/A

Table 2: Total system power consumption (mW) is reduced by 62%, 60%, and 42% as sensor resolution is reduced from 1080p (1920x1080 in legacy V4L2) to 480p (640x480 in legacy V4L2), in each workload accordingly. In addition, physically reconfiguring sensor resolution to 480p (640x480 in Banner) consumes 43%, 49%, and 16% less total system power than downsampling 1080p to 480p (1080p↓480p in legacy V4L2), in each workload accordingly.

Resolution-Framework	SoC	GPU	DDR	CPU	Total
1920x1080-V4L2	803	230	951	879	2863
1280x720-V4L2	637	230	686	693	2246
640x480-V4L2	612	230	618	613	2073
99x-reconf.-V4L2	659	230	719	691	2299
99x-reconf.-Banner	620	230	644	600	2094

Table 3: Reconfiguring sensor resolution dynamically (99x-reconf.-Banner) can reduce 27% of the combined system power consumption (mW) comparing with constantly working at 1080p (1920x1080-V4L2), measured with our CPU-based cloud-based offloading workload working at 30 FPS.

5.3.1 Effectiveness of Dynamic Reconfiguration. To demonstrate the power efficiency improvement brought by Banner for reconfiguring sensor resolution dynamically, we conduct a simple experiment in which the sensor resolution cycles through 1080p, 720p, and 480p every 10 frames (a randomly chosen number) in a total of 1000 frames. This results in 99 resolution reconfigurations. We run this pattern with our CPU-based cloud-based offloading workload working at 30 FPS. The results in Table 3 show that even in legacy V4L2 framework, reconfiguring sensor resolution dynamically (99x-reconf.-V4L2) can reduce 20% of the combined system power consumption comparing with constantly working at 1080p – notably, there are substantial frame drops with each reconfiguration in the legacy V4L2 system.

Meanwhile, reconfiguring sensor resolution with Banner (99x-reconf.-Banner) can further reduce total power consumption by 9% and without the frame drop penalty, compared with 99x-reconf.-V4L2. These power savings come from the use of fewer operations to reconfigure the sensor format and no repeated memory allocation in Banner. The power consumption of 99x-reconf.-Banner is roughly about the same as constantly working at 480p.

5.3.2 Power overhead of Banner. As shown in Table 2, comparing between Banner and legacy V4L2 framework, there is no obvious power overhead. Specifically, Banner does not consume more DDR power despite its allocation of more memory than the active resolution requires. This is because DDR power consumption is based on data rate, not buffer size [36].

5.4 Implications

Banner enables rapid sensor resolution reconfiguration by eliminating frame-to-frame latency and halving the end-to-end reconfiguration latency. This unlocks a more than 49% system power consumption reduction by reconfiguring the image sensor resolution from 1080p to 480p comparing with computationally downsampling to 1080p↓480p. As we mentioned in our motivation, in a variety of vision tasks, the image resolution needs to be configured dynamically to adapt to environmental changes in order to maximize the power efficiency. For example, the required sensor resolution can be determined dynamically based on the continuously changing distance between the image sensor and the marker in a marker-based pose estimation application. Our evaluation in the marker-based pose estimation application on the Jetson/AR0330 system reveals that the estimated pose accuracy can be maintained (±0.1 cm MSE translation vector error) even if the image resolution is reconfigured from 1080p to 720p and then to 480p as the distance between the image sensor and the marker is reduced from 40 cm to 20 cm. This results in a 28% power consumption reduction between 1080p and 720p and a 42% power consumption reduction between 1080p and 480p.

6 DISCUSSION AND FUTURE WORK

Driver modification: Banner does not modify the camera driver, which is often a sensitive proprietary piece of software. Banner also does not add any dependencies to the video buffer driver. Banner only involves changes in the camera host driver and exposes itself as a system call to the application developers, as the V4L2 driver

does. Thus, developers can use Banner just as they use the V4L2 framework for video streaming. If developers want to use other media frameworks with the Banner-based camera host driver, they also need to modify the media frameworks' library. In addition, as we mentioned in §3.1, different images sensors take different amount of time to reconfigure resolution. Banner can be further optimized if image sensor manufactures are able to optimize the camera driver implementation together with the camera hardware for faster sensor resolution reconfiguration.

Memory management: Banner allocates more memory than is needed for smaller resolution. However, allocating three buffers for 1920x1080 frames with 8 bits per pixel only requires 6 MB memory. Storing 640x480 frames in them will cause memory waste but only about 5 MB. Unused memory does not incur power overhead, as observed from our evaluation. If the application still wants to utilize that memory, Banner's memory management is able to provide enough information. We did not implement or evaluate this fine-grained memory management in this version of Banner. We assume resolution change is frequent in some vision applications who want to utilize resolution-based energy and accuracy tradeoff. Thus, the memory management could face an interesting but complex situation that requires frequently reallocating memory and/or losing access to memory. We will investigate more sophisticated memory management in future work.

Other system support: Banner will benefit almost all systems that build upon the Linux system, including the Android OS. However, we were unable to implement and test Banner in the Android environment because of the closed source camera host and device drivers. In particular, the Android OS requires captured frames to be read out before reconfiguring the sensor resolution which allows Banner to fit in. We suggest the following three policies for integrating Banner to the Android OS. (i) Direct the resolution request to Banner API through the Android frameworks and camera HAL. (ii) Issue the resolution request directly in current camera preview and capture session. The Android OS requires captured frames with previous resolution to be readout before reconfiguring the sensor (a function called `waitUtilIdle()`) which perfectly match the parallel reconfiguration component in Banner. (iii) Update the preview size based on frame metadata right after `waitUtilIdle()` finishes. In addition, the concepts of parallel reconfiguration and format-oblivious memory management in Banner are also generic to other media frameworks, not only the V4L2 framework.

Machine learning support: Machine learning based vision applications also have resolution-based tradeoffs [26, 31, 33, 39]. Banner as a media framework is generic to upper level applications. However, based on our understanding, most of the neural networks often work on fixed-size images. The structural influence of Banner to neural networks still needs to be explored. Machine learning developers are welcome to use Banner to find opportunities relying on resolution-based performance and accuracy tradeoff.

Optimal resolution selection algorithm: When we evaluated Banner with the OpenCV marker-based pose estimation, we triggered the resolution change based on the physical distance between the maker and the sensor. We noticed that the frequency of resolution reconfiguration definitely affects the amount of power savings. Maximizing power saving with minimum performance drop can be determined by many factors according to different applications. We are targeting on a more sophisticated resolution selection algorithm as our future work, such as teaching the machine to decide when to change the sensor resolution in a machine learning environment.

Limitations: As we discussed in the evaluation, the reduction in end-to-end reconfiguration latency could vary depending on the frame rate. If the frame rate is low, the reduction in end-to-end reconfiguration latency can be alleviated, as shown in the slower cloud-based offloading and the pose estimation workloads. Another limitation is that sensor resolution reconfiguration takes time in Banner, even though it can be completely hidden behind application process. Thus, if the application process is very fast, then the frame rate for that specific frame during resolution reconfiguration will be limited. Improvements in camera design and camera driver implementation are required to further reduce the sensor resolution reconfiguration latency.

7 RELATED WORK

Resolution-based energy, performance and accuracy tradeoff: Ha et al. [11] and Hu et al. [15] verified the resolution-based energy, performance and accuracy tradeoff on mobile devices. LiKamWa et al. [22] demonstrated that image sensor energy consumption can be proportional to frame rate and resolution. Sundaram [35] downsampled the images for faster frame rate. Kumar et al. [21] cropped the images for faster training. Haris et al. [12] and Růžička et al. [33] increased the resolution for more accurate object detection. YOLO3 [31] and Lin et al. [26] explored the resolution-based performance and accuracy tradeoff in machine learning based vision applications. Lin et al. [25] even demonstrated that energy, performance and accuracy tradeoff exists in other types of sensors. However, no frame-critical application is able to change resolution at runtime because resolution change incurs long latency penalties in current operating systems as we described in the previous sections. Banner provides rapid and no-frame-drop sensor reconfiguration, unlocking the ability for these vision tasks to use resolution-based energy, performance, and accuracy tradeoff in real-time.

Consecutive captures with different settings: Photography can blend several frames captured with different sensor settings for improving image quality. High Dynamic Range imaging rapidly takes three photos with different exposure settings [2, 38]. Similarly, Samsung's [34] multi-frame image processing utilized several consecutive captures to reduce the blur. Frankencamera [1] and KHRONOS Group [10] recognized the significance of programmability for camera operation. Sensor register change can be effective in the next capture [27, 28] such that sensor settings, e.g., exposure, can be tailored for each frame. Android has provided system support for streamlined reconfiguration. Unfortunately, this does not work for resolution reconfiguration, primarily because of the sequential reconfiguration patterns and memory management issues we discussed in this work. Banner enables tailoring each frame for different resolutions for a variety of vision tasks that need resolution-based energy efficiency tradeoffs.

Towards energy efficient continuous sensing: Rigel introduced flexible multi-rate image processing pipeline to improve vision tasks' performance with three orders of magnitudes lower energy consumption [14]. Darkroom [13] eased developers burden to utilize specialized image signal processors for higher energy efficiency.

Roy et al. [32] introduced an energy-efficiency context recognition framework for multi-modal sensing. LittleRock [30] [29] proposed dedicated low-level processing hardware to reduce sensing power. Reflex [24] helped developers to leverage low-power processors on mobile devices. Buckler et al. [6] abandoned image signal processing hardware to reduce energy consumption. Chu et al. [8] proposed a tool to classify sensor data for efficient multimodal sensing on mobile devices. SeeMon [17] helped applications to efficiently understand context from numerous sensors. Glimpse [7] proposed an energy efficient continuous object detection system that balanced the accuracy and energy tradeoff between local and offload computing. Starfish [23] enabled resource sharing among computer vision applications to improve energy efficiency. In addition to these, Banner enables rapid sensor resolution-based energy tradeoff at the operating system level to improve continuous sensing efficiency.

8 CONCLUSION

We observe a substantial image sensor resolution reconfiguration latency caused by the sequential reconfiguration procedure in current operating systems. This long reconfiguration latency gives vision applications a perception of losing frames which impedes the adoption of otherwise beneficial resolution-energy tradeoff mechanisms. In this paper, we propose Banner as a system solution for providing rapid and seamless image sensor resolution reconfiguration. Evaluated in three different OpenCV workloads including display-only, cloud-based offloading, and marker-based pose estimation running on a Jetson TX2 board, Banner is able to halve the end-to-end reconfiguration latency and completely remove the frame-to-frame latency, i.e., no frame drop during sensor resolution reconfiguration even for workloads working at 30 FPS. This allows a more than 49% system power consumption reduction comparing reconfiguring the sensor resolution from 1080p to 480p with downsampling 1080p↓480p. Banner unlocks a variety of mobile vision tasks to dynamically reconfigure sensor resolutions to adapt to the environmental change and then maximize the energy efficiency.

Acknowledgement We sincerely thank Jeremy C. Andrus for shepherding the final version of this paper and all the valuable reviews given by the anonymous reviewers. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1657602. The work was also supported by Samsung Mobile Processor Innovation Lab.

REFERENCES

[1] Andrew Adams, Eino-Ville Talvala, Sung Hee Park, David E. Jacobs, Boris Ajdin, Natasha Gelfand, Jennifer Dolson, Daniel Vaquero, Jongmin Baek, Marius Tico, Hendrik P. A. Lensch, Wojciech Matusik, Kari Pulli, Mark Horowitz, and Marc Levoy. 2010. The Frankencamera: An Experimental Platform for Computational Photography. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM. https://doi.org/10.1145/1833349.1778766

[2] Apple. 2018. Use HDR on your iPhone, iPad, and iPod touch. (2018). https://support.apple.com/en-us/HT207470

[3] Apple. 2019. ARKit. (2019). https://developer.apple.com/arkit//

[4] Saul Berenbaum. 2013. Google Glass Explorer Edition has a 30-minute battery life while shooting video. (2013). https://www.digitaltrends.com/mobile/google-glass-30-minute-videobattery/

[5] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. 2018. EVA2: Exploiting Temporal Redundancy in Live Computer Vision. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18)*. IEEE Press, Piscataway, NJ, USA, 533–546. https://doi.org/10.1109/ISCA.2018.00051

[6] Mark Buckler, Suren Jayasuriya, and Adrian Sampson. 2017. Reconfiguring the Imaging Pipeline for Computer Vision. In *The IEEE International Conference on Computer Vision (ICCV)*.

[7] Tiffany Chen, Hari Balakrishnan, Lenin Ravindranath, and Paramvir Bahl. 2016. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. *GetMobile: Mobile Computing and Communications* 20 (07 2016), 26–29. https://doi.org/10.1145/2972413.2972423

[8] David Chu, Nicholas D. Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. 2011. Balancing Energy, Latency and Accuracy for Mobile Sensor Data Classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*. ACM. https://doi.org/10.1145/2070942.2070949

[9] Google. 2019. ARcore. (2019). https://developers.google.com/ar//

[10] KHRONOS Group. 2013. Camera BOF. (2013). https://www.khronos.org/assets/uploads/developers/library/2013-siggraph-camera-bof/Camera-BOF_SIGGRAPH-2013.pdf

[11] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards Wearable Cognitive Assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '14)*. ACM.

[12] Muhammad Haris, Greg Shakhnarovich, and Norimichi Ukita. 2018. Task-Driven Super Resolution: Object Detection in Low-resolution Images. *CoRR* abs/1803.11316 (2018). arXiv:1803.11316 http://arxiv.org/abs/1803.11316

[13] James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. 2014. Darkroom: Compiling High-level Image Processing Code into Hardware Pipelines. *ACM Trans. Graph.* 33, 4, Article 144 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601174

[14] James Hegarty, Ross Daly, Zachary DeVito, Jonathan Ragan-Kelley, Mark Horowitz, and Pat Hanrahan. 2016. Rigel: Flexible Multi-rate Image Processing Hardware. *ACM Trans. Graph.* 35, 4, Article 85 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925892

[15] Jinhan Hu, Jianan Yang, Vraj Delhivala, and Robert LiKamWa. 2018. Characterizing the Reconfiguration Latency of Image Sensor Resolution on Android Devices. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications (HotMobile '18)*. ACM. https://doi.org/10.1145/3177102.3177109

[16] Rizwan Mohamed Ibrahim. 2019. Camera. (2019). https://github.com/rizwankce/Camera/

[17] Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. 2008. SeeMon: Scalable and Energy-efficient Context Monitoring Framework for Sensor-rich Mobile Environments. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08)*. ACM. https://doi.org/10.1145/1378600.1378630

[18] The kernel development community. V4L2 video capture example.

[19] The kernel development community. 2019. ioctl VIDIOC_REQBUFS. (2019). https://linuxtv.org/downloads/v4l-dvb-apis/uapi/v4l/vidioc-reqbufs.html

[20] Venkatesh Kodukula, Sai Bharadwaj Medapuram, Britton Jones, and Robert LiKamWa. 2018. A Case for Temperature-Driven Task Migration to Balance Energy Efficiency and Image Quality of Vision Processing Workloads. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications (HotMobile '18)*. ACM. https://doi.org/10.1145/3177102.3177111

[21] Athindran Ramesh Kumar, Balaraman Ravindran, and Anand Raghunathan. 2018. Pack and Detect: Fast Object Detection in Videos Using Region-of- Interest Packing. *ArXiv e-prints*, Article arXiv:1809.01701 (Sept. 2018), arXiv:1809.01701 pages. arXiv:cs.CV/1809.01701

[22] 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 (MobiSys '13)*. ACM. https://doi.org/10.1145/2462456.2464448

[23] Robert LiKamWa and Lin Zhong. 2015. Starfish: Efficient Concurrency Support for Computer Vision Applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '15*. ACM. https://doi.org/10.1145/2742647.2742663

[24] Felix Xiaozhu Lin, Zhen Wang, Robert LiKamWa, and Lin Zhong. 2012. Reflex: Using Low-power Processors in Smartphones Without Knowing Them. *SIGPLAN Not.* 47, 4 (March 2012), 13–24. https://doi.org/10.1145/2248487.2150979

[25] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. 2010. Energy-accuracy Trade-off for Continuous Mobile Device Location. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. ACM. https://doi.org/10.1145/1814433.1814462

[26] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. *CoRR* abs/1708.02002 (2017). arXiv:1708.02002 http://arxiv.org/abs/1708.02002

[27] ON Semiconductor 2017. *AR0330 1/3-inch CMOS Digital Image Sensor.* ON Semiconductor. Rev. 18.

[28] ON Semiconductor 2017. *MT9P031 1/2.5-Inch 5 Mp CMOS Digital Image Sensor.* ON Semiconductor. Rev. 10.

[29] Bodhi Priyantha, Dimitrios Lymberopoulos, and Jie Liu. EERS: Energy Efficient Responsive Sleeping on Mobile Phones.
[30] B. Priyantha, D. Lymberopoulos, and J. Liu. 2011. LittleRock: Enabling Energy-Efficient Continuous Sensing on Mobile Phones. *IEEE Pervasive Computing* 10, 2 (April 2011), 12–15. https://doi.org/10.1109/MPRV.2011.28
[31] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767 (2018). arXiv:1804.02767 http://arxiv.org/abs/1804.02767
[32] N. Roy, A. Misra, C. Julien, S. K. Das, and J. Biswas. 2011. An energy-efficient quality adaptive framework for multi-modal sensor context recognition. In *2011 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. https://doi.org/10.1109/PERCOM.2011.5767596
[33] Vít Ruzicka and Franz Franchetti. 2018. Fast and accurate object detection in high resolution 4K and 8K video using GPUs. *CoRR* abs/1810.10551 (2018). arXiv:1810.10551 http://arxiv.org/abs/1810.10551
[34] Samsung. 2017. [In-Depth Look] Fast, Fun and In-Focus: The Galaxy S8 Camera. (2017). https://news.samsung.com/global/in-depth-look-fast-fun-and-in-focus-the-galaxy-s8-camera
[35] Narayanan Sundaram. 2012. *Making computer vision computationally efficient*. Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-106.html
[36] Micron Technology. Calculating Memory System Power for DDR. https://www.micron.com/~/media/Documents/Products/Technical%20Note/DRAM/TN4603.pdf
[37] Vuforia. 2019. Innovate With Industrial Augmented Reality. (2019). https://www.ptc.com/en/products/augmented-reality/
[38] Wikipedia. 2018. High-dynamic-range imaging. (2018).
[39] Boyu Zhang, Azadeh Davoodi, and Yu-Hen Hu. 2018. Exploring Energy and Accuracy Tradeoff in Structure Simplification of Trained Deep Neural Networks. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASPDAC '18)*. IEEE Press. http://dl.acm.org/citation.cfm?id=3201607.3201693