Rhythmic Pixel Regions: Multi-resolution Visual Sensing System towards High-Precision Visual Computing at Low Power

Venkatesh Kodukula
Arizona State University
Tempe, AZ, USA
vkoduku1@asu.edu

Alexander Shearer
Arizona State University
Tempe, AZ, USA
acshear1@asu.edu

Van Nguyen
Arizona State University
Tempe, AZ, USA
vtnguy19@asu.edu

Srinivas Lingutla
Arizona State University
Tempe, AZ, USA
slingutl@asu.edu

Yifei Liu
Arizona State University
Tempe, AZ, USA
yliu740@asu.edu

Robert LiKamWa
Arizona State University
Tempe, AZ, USA
likamwa@asu.edu

ABSTRACT

High spatiotemporal resolution can offer high precision for vision applications, which is particularly useful to capture the nuances of visual features, such as for augmented reality. Unfortunately, capturing and processing high spatiotemporal visual frames generates energy-expensive memory traffic. On the other hand, low resolution frames can reduce pixel memory throughput, but reduce also the opportunities of high-precision visual sensing. However, our intuition is that not all parts of the scene need to be captured at a uniform resolution. Selectively and opportunistically reducing resolution for different regions of image frames can yield high-precision visual computing at energy-efficient memory data rates.

To this end, we develop a visual sensing pipeline architecture that flexibly allows application developers to dynamically adapt the spatial resolution and update rate of different "rhythmic pixel regions" in the scene. We develop a system that ingests pixel streams from commercial image sensors with their standard raster-scan pixel read-out patterns, but only encodes relevant pixels prior to storing them in the memory. We also present streaming hardware to decode the stored rhythmic pixel region stream into traditional frame-based representations to feed into standard computer vision algorithms. We integrate our encoding and decoding hardware modules into existing video pipelines. On top of this, we develop runtime support allowing developers to flexibly specify the region labels. Evaluating our system on a Xilinx FPGA platform over three vision workloads shows 43 – 64% reduction in interface traffic and memory footprint, while providing controllable task accuracy.

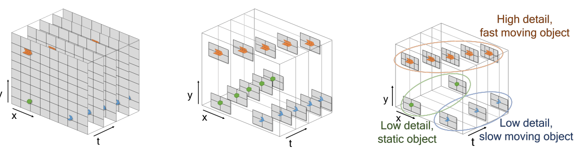
CCS CONCEPTS

• **Computer systems organization → Special purpose systems**.

KEYWORDS

visual computing, augmented reality, pixel discard

ACM Reference Format:
Venkatesh Kodukula, Alexander Shearer, Van Nguyen, Srinivas Lingutla, Yifei Liu, and Robert LiKamWa. 2021. Rhythmic Pixel Regions: Multi-resolution Visual Sensing System towards High-Precision Visual Computing at Low Power. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3445814.3446737



(a) Frame-based **(b) ROI-based** **(c) Rhythmic pixels**

Figure 1: Traditional frame-based computing captures and processes entire frames. ROI-based computing samples regions of interest, but at uniform spatial and temporal resolution. With rhythmic pixel regions, different regions are captured at different spatio-temporal resolutions.

1 INTRODUCTION

Through the lens of their cameras, mobile devices can visually observe a user's environment for detecting faces, understanding spatial scene geometry, and capturing images and videos. This has yielded a wide range of benefits, especially as image sensors have grown to support increasingly higher resolutions and frame rates. With such precision, it is now possible to position augmented reality (AR) overlays over users' faces or spatial living environments for social entertainment. Virtual AR media can also annotate physical environmental surfaces, including for navigational guidance for walking directions [14]. Wirelessly connected camera devices on the Internet-of-Things can provide home security, such as through video doorbells.

Unfortunately, visual systems on mobile systems are limited in their spatial precision, computational performance, and energy efficiency while performing continuous visual tasks. Mobile systems

Table 1: Opportunities for Rhythmic Pixel Regions

	Traditional uniform frame-based vision	**Rhythmic pixel region-based vision**
Spatial Resolution	If any part of the frame needs to be captured at a high resolution, e.g., to resolve complex texture or distant objects, the entire frame will need to be captured at a high resolution.	Regions with small, detailed, and/or distant features can be captured with the precision of high resolution. Frame regions with large, static, close visual features can be captured with the efficiency of low resolution.
Temporal Resolution	If any part of the frame needs to be captured at a high frame rate, e.g., to track substantial motion, the system will need to capture a sequence of entire frames of pixels at high frame rate.	Regions can be captured at different intervals. The entire frame can be scanned to update spatial understanding at a lower rate, e.g., 1 fps. Regions of moving objects/surfaces can be captured at higher rates, e.g., 60 fps.

are constrained by their small form factors with limited battery sizes and heat management requirements. To reduce the power consumption, recently proposed systems reduce the spatial resolution and frame rate of image capture [19, 22] to receive commensurate energy savings, especially by reducing the memory traffic of the DRAM-based frame buffers. Thus, resolution provides a tradeoff mechanism to dynamically configure systems for low power consumption or high visual task fidelity. However, downscaling or windowing a frame forces the application to reduce the resolution of the entire frame. Reducing the frame rate similarly reduces the temporal resolution of the entire frame stream. This reduced spatiotemporal resolution across the entire frame stream can lead to suboptimal visual precision.

This work aims to improve the capabilities of continuous mobile vision systems, based on a key insight: *The precision, performance, and efficiency of visual computing systems are limited by the current pattern of capturing and processing entire image frame streams at **uniform** spatial resolutions and **uniform** frame rates.* This assumption of frame-based computing (Fig. 1a) presents an inflexibly coarse granularity of tradeoff between task accuracy and energy efficiency. Most natural scenes do not have the same resolution needs across the entire image frame. Precise AR placement requires high spatial resolution for visual features on tracked surfaces, but would suffice with a relatively lower resolution for the rest of the frame. The detection and tracking of faces, hands, and objects could use a higher temporal resolution to capture quick motions, while the rest of a relatively static scene would suffice with a lower frame rate. Such tradeoffs are unavailable with the current model of frame-based computing.

To address this, we present a fundamental shift away from frame-based visual computing and towards ***rhythmic pixel regions*** (Fig. 1c), which we define as neighborhoods of pixels with region-specific spatiotemporal resolutions. Unlike visual computing based on a few Regions-of-Interest (ROIs) [21], rhythmic pixel regions leverage encoded data representations that scalably allow for the capture of hundreds of regions, with independently defined spatiotemporal resolutions. By supporting the simultaneous capture of a diversity of rhythmic pixel regions, our visual computing architecture allows the developer to selectively specify regions where higher spatiotemporal resolution is needed and where lower spatiotemporal resolution will suffice (Table 1), e.g., dynamically guided by the properties of the visual features. The fine-grained configurability will allow developers to extend their existing visual computing algorithms and applications for high energy efficiency. This creates the illusion of high spatiotemporal resolution capture at low power consumption by eliminating the wasteful DRAM traffic of unproductive pixels.

To design the rhythmic pixel region abstraction and the architecture to support it, we introduce two hardware sensor data interfaces: an encoder to selectively reduce sets of pixels before they are stored in memory, and a decoder to reconstruct the pixels from memory into traditional frame-based addressing for standard application use. Together, the rhythmic pixel region encoder and decoder work to reduce the significant DRAM traffic of writing and reading visual data, leading to system energy savings. These interfaces, which integrate into the System-on-Chip, support existing and future high-resolution image sensors, allowing for a revolutionary upgrade to evolutionary mass-market-scale image sensors.

To evaluate our system, we design our implementation on a Xilinx ZCU102 FPGA SoC platform. We support various visual workloads, including hardware-accelerated neural network processing for face and object tracking, and OpenCV-based visual simultaneous localization and mapping (V-SLAM). Through our evaluation, we demonstrate the opportunity of rhythmic pixel regions to decrease pixel memory traffic by 43 – 64% while only minimally degrading the visual task accuracy, e.g., only increasing absolute trajectory error of V-SLAM from 43 ± 1.5 mm to 51 ± 0.9 mm in our case study. Through reduction in memory traffic, rhythmic pixel region based techniques can significantly increase the energy-efficiency of battery-backed mobile systems.

In summary, we make the following contributions:

- To the best of our knowledge, we are the first to propose a visual computing paradigm where different parts of the scene are captured at different spatiotemporal resolutions – before the frame enters memory – for overall system energy-efficiency while respecting task needs.
- We develop two *lightweight and scalable* IP blocks – rhythmic pixel encoder and decoder – which decimates the incoming pixel stream while writing to memory and reconstructs the pixel stream on-the-fly while reading from memory.
- We develop a library and runtime to coordinate vision tasks with encoder/decoder operation.
- We augment our architecture and runtime support on top of an existing commercial mobile vision pipeline built around a FPGA platform. We evaluate the augmented architecture on a variety of vision tasks to demonstrate significant reduction in memory traffic with controllable accuracy loss.

2 BACKGROUND AND RELATED WORK

A primer on vision pipelines: The ecosystem of visual computing sensors, devices, systems, and algorithms on mobile devices have rapidly evolved to provide high-performance platforms for multitude of vision tasks such as augmented reality on smartphones, tablets, and headsets [18].

Image sensors collect digital readings of visual pixels in a frame [38]. The sensor sends values over a streaming MIPI interface, which enacts a serial transmission over multiple lanes [30]. The MIPI receiver inside the SoC receives the frame information from camera. In the sensor or on the system-on-chip, there is often an image signal processor (ISP) inserted into the visual computing pipeline, performing image improvement operations, e.g., white balance, and format changes, e.g., YUV conversion. Regardless of the placement and operation of the ISP, the visual hardware pipeline eventually writes the frame into DRAM and signals to the operating system that a frame is ready for readout from the memory.

Data movement across the off-chip MIPI and DDR interfaces entails significant energy consumption [12]. While tasks such as AR could significantly benefit from high spatiotemporal resolutions, e.g., 4K at 60 fps, these resolutions generate high datarates across the camera and memory interfaces. For example, the system expends 2.8 nJ to move a pixel [17, 24, 27, 35, 36] across the DDR interface, whereas it expends only 4.6 pJ for performing a multiply-and-accumulate (MAC) operation [16] around that pixel.

For AR, the software processes the frame through visual computing frameworks, extracting visual features to feed into SLAM algorithms [31]. These algorithms form a spatial understanding of the scene to estimate the pose of the camera. The pose of the virtual camera is precisely updated to the estimated pose of the physical camera. This allows the system to overlay virtual objects over the physical environment, achieving the AR illusion.

Multi-ROI sensors: Many image sensors are capable of selecting a region-of-interest (ROI) for readout [49]. There are also sensors that allow for multiple ROIs to be read out [39]. As region selection is performed at the sensor level, it offers efficiency and speed by reducing sensor readout time. However, there are significant limitations to adopting these sensors for mobile systems. In particular, the expressiveness of sensor-based region selection is limited by the footprint of additional circuitry. For example, in one such sensor, the region selection is limited to 4 regions, regions cannot overlap, and only full resolution and frame rate are available [49]. In contrast, our support for rhythmic pixel regions provides extensive configurability and composability through the inherent scalability of the encoded data representation. This allows a much larger number of regions (hundreds), and grants each region independent resolution and rhythm/interval control. Moreover, we implement region selection in the SoC, which allows any conventional image sensor to employ multi-ROI benefits, including emerging high resolution and high framerate sensors.

Event-driven cameras: Event-driven cameras, also known as dynamic vision sensors, focus their sampling on pixels that change their value [11]. This allows for a significant reduction in sensor bit rate and allows microsecond time resolution. However, the circuitry is spatially expensive due to per-pixel motion detection module, reducing frame resolution, e.g., to 128 x 128 pixels. This limits the scalability of these sensors to support high resolutions. More fundamentally, the logic of deciding what pixels to read out is limited at hardware design time, disallowing high-level and/or semantic knowledge from governing the pixel selection process. Thus, while our work shares similar motivations and inspirations – reducing data rate for efficiency and performance – we uniquely allow the expressive ability to dynamically use knowledge of visual feature extraction needs to selectively sample pixels as needed.

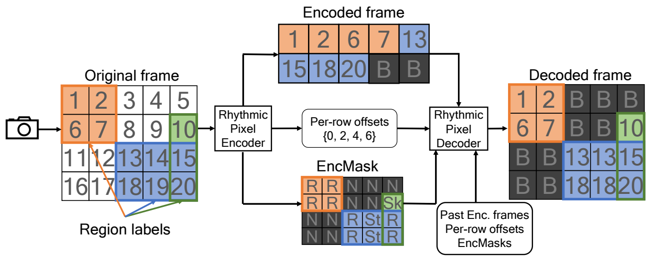
Image/Video Compression: Decades of work in image processing has gone towards compressing images and videos to reduce the bit rate of storing and transmitting media. Many of these techniques are inspiring to this work. For example, JPEG and other image compression standards reduce information at spatial frequencies with less perceptual priority [38]. MPEG-H Part 2 / HEVC / H.265 reduce redundant information by leveraging estimated motion information from frame-to-frame [20]. However, such entropy-coding compression techniques require the frame – or multiple copies of the frame – in memory before compression can be done. This incurs the memory overhead of visual computing that rhythmic pixel regions strives to avoid; we perform encoding *before* the frame enters the DRAM. Traditional video codecs also employ sophisticated techniques such as discrete cosine transform (DCT) and motion compensation for data reduction, increasing their design complexity. In contrast, we use simple pixel discard based strategy for data reduction.

Foveated Rendering: To improve graphical rendering capabilities under limited computing resources, foveated rendering focuses rendering effort where users are likely to notice. Foveated rendering [43] uses eye tracking to estimate user gaze and renders content near the gaze target at higher spatial resolutions. We apply similar motivation, increasing spatiotemporal resolution where it is needed, but with a distinctly different goal: to only capture necessary visual information. Among other differences, our work involves multiple simultaneous regions, as opposed to the singular region in foveated rendering.

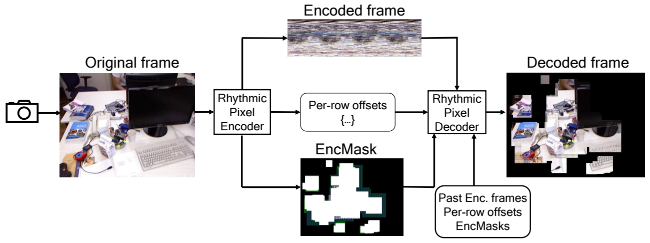
Flexible spatio-temporal sampling algorithms: Computer vision researchers propose different algorithms that modulate different regions in an image with different spatio-temporal resolutions for computational photography and video prediction use-cases. Flexible voxels [15] proposes an efficient space-time sampling scheme that enables per-pixel temporal modulation and a motion-aware reconstruction scheme for artifact-free videography. More advanced techniques [37] have been proposed for space-time sampling and interpolation for video compressive sensing and high-speed imaging.

More recently, researchers proposed neural networks [23] to generate high frame rate videos from low frame rate counterparts. These networks analyze spatiotemporal patterns across frames in a video to create interpolated frames. While we use similar spatiotemporal sampling mechanisms, we focus on providing the architecture and runtime support to enable spatiotemporal rhythmic capture on an embedded system.

Energy-efficient visual systems To reduce energy consumption and bandwidth utilization, some systems offload only interesting frames to the cloud, discarding the rest [6]. Determining which frames to discard itself is often an expensive task. To this end, other systems [32] implement a dedicated hardware/software subsystem using an array of gating sensors and imagers. Other



(a) Illustrative representation of dataflow



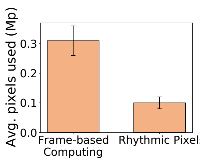
(b) Experimental example data with ORB-SLAM-based regions

Figure 2: The process of encoding and decoding. The encoder packs pixels within the regions from the original image, maintaining raster-scan order. The decoder reconstructs data from the encoded frame, per-row offsets, and encoding mask.

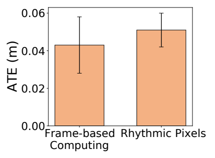
hardware-software co-design techniques reuse computation for energy-efficient visual computing. Euphrates [51] reuses the motion vector information from the ISP to extrapolate bounding box results for object detection to avoid extraneous vision processing on certain frames. EVA^2 [4] exploits temporal redundancy between frames and uses motion compensation techniques to approximate CNN results. ASV [10] applies stereo-specific algorithmic and computational optimizations to increase the performance with minimum error rate. Instead of approximating computation, our work primarily focuses on reducing the memory traffic of sensing by discarding pixels early in the vision pipeline. As such, rhythmic pixel regions can be complementary to the aforementioned visual processing techniques, with approximate computing filling in the visual gap of unsampled pixel data.

3 VISUAL COMPUTING WITH RHYTHMIC PIXEL REGIONS

The overarching concept of rhythmic pixel regions is to encode visual information captured by conventional sensors such that the encoded visual data: (*i*) meets real-time visual requirements specified by application developers, (*ii*) reduces the pixel memory throughput and footprint of pixel stream data into DRAM, and (*iii*) can be reconstructed into visual frames for application usage. Here, we introduce key data structures that specify the region labels, the encoded pixels, and associated meta-data, and how these data structures are used to perform pixel sampling and interpolation to satisfy visual needs. We also illustrate the effect of this new paradigm on a case study based on visual SLAM.



(a) Pixels captured



(b) Absolute Trajectory Error

Figure 3: For ORB-SLAM, rhythmic pixel regions can discard irrelevant pixels early in the pipeline for memory efficiency, while preserving relevant regions at sufficient resolutions for accuracy.

3.1 Developer-Specified Region Labels

To mark the regions to be captured, as well as their spatiotemporal qualities, the system allows developers to specify a set of region labels (Fig. 2a) to define a capture workload. Each region label includes the following attributes:

- The coordinates of the top-left corner of the region
- The width and height of the region
- The stride resolution of the region, i.e., the density of pixels
- The skip rate, i.e., the time interval of consecutive sampling

We do not yet support movement, resize, or other temporal dynamics of regions in this work.

3.2 Encoded Frame

The encoder uses the region labels to selectively store a reduced set of pixels into memory, forming the encoded frame. Pixels within any of the region labels are stored in original raster-scan order, while omitting any pixels that do not fall within regions, or are not within the rhythm of the stride or skip. This results in a tightly packed encoded frame (Fig. 2a).

Notably, traditional ROI-based computing typically adopts a different memory representation, storing each region of pixels as a grouped sequence in memory. For the small number of regions that ROI-based computing typically supports, this may suffice, but when scaled to hundreds of regions, this creates unfavorable random access patterns into DRAM and/or buffering of large portions of the frame into local SRAM when writing pixels into memory. Furthermore, overlapping regions will duplicate the storage of pixels that appear in multiple regions. Instead, by preserving raster scan order, the rhythmic encoded frame representation instead retains sequential write patterns. This allows for highly efficient scalability to raised number of regions with minimal resource overhead.

3.3 Metadata: Per-Row Offset, Encoding Mask (EncMask)

To service pixel requests from the vision applications, the decoder will need to translate pixel addresses in the original frame into pixel offsets in the encoded frame before retrieving the pixel value. However, this would limit decoder scalability, as the complexity of the search operation quickly grows with additional regions.

Thus, instead of using region labels, we propose an alternative method that uses two forms of metadata for the decoder to reconstruct the original pixel stream (Fig. 2a).

First, a per-row offset counts the number of encoded pixels prior to that row in the image. Second, an encoding sequence bitmask (EncMask) helps the decoder to reconstruct a pixel stream without the need for region labels. For each pixel in the original (pre-encoding) frame, the EncMask uses a two-bit status that indicates how a pixel is sampled in space and time:

N (`00`): Non-regional pixel
St (`01`): Regional pixel but strided
Sk (`10`): Regional pixel but temporally skipped
R (`11`): Regional pixel

Together, the per-row offset and EncMask allow the decoder to find the relevant pixel address in the encoded frame, and access the appropriate values from the current encoded frame or previous encoded frames to decode the pixel. We provide more information about this process in Section 4.2.

3.4 A Case Study around ORB-SLAM

ORB-SLAM [31] is a popular real-time V-SLAM algorithm, highly centered around tracking visual "features", creating a map of localized features, and finding matches to previously mapped features for continual positioning. Visual features – key for scene understanding – are scattered throughout a scene and carry different spatiotemporal needs. While running ORB-SLAM for augmented reality, for example, the feature extractor routinely detects several hundreds of features (e.g., 1500 features in a 1080p frame) at varying distances from the camera. This motivates the need to sample hundreds of regions around hundreds of features with need-specific resolutions. Features can be grouped into a smaller number of regions, but this reduces task accuracy and memory efficiency, as we demonstrate in §6.

In the context of rhythmic pixel regions, ORB features will be located in the decoded pixel streams. Regions covering the entire frame can be captured at a slower skip rate to maintain coverage of features as the device moves around the space. In between full captures, detected features in both full and partial captures can guide region label selection for the subsequent frame, defining regions around feature locations and with properties based on feature characteristics. Specifically, our policy uses the feature's "size" attribute to guide the width and height of the region. Along similar lines, it uses other feature attributes such as "octave" for stride and feature movement between frames for temporal rate. The encoder uses these feature-based region labels to selectively store pixels of interest and metadata into memory. The decoder uses these to reconstruct the frames for the algorithm to access and use for ORB-SLAM and further region selection.

We evaluate the efficacy over the TUM dataset [40] of 480p videos, capturing full frames every 10 frames and feature-based regions for other frames. As shown in Fig. 3, we find that using rhythmic pixel regions can eliminate the memory storage of 66% of the pixels of the original stream, while only increasing absolute trajectory error from 43 ± 1.5 mm to 51 ± 0.9 mm. We further explore the applicability of rhythmic pixel regions to visual workloads in our evaluation.

4 DESIGN

The rhythmic pixel region architecture centers around the idea of: (*i*) encoding pixel streams to reduce the pixel data stored in memory,

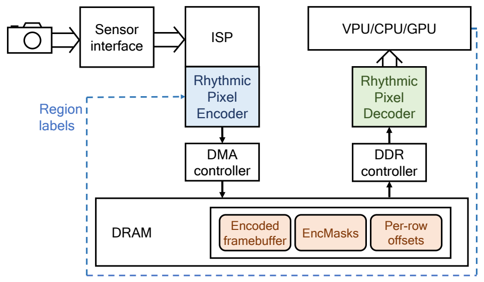


Figure 4: System design. The rhythmic encoder decimates the incoming pixel stream from camera and encodes only region-specified pixels into memory. The rhythmic decoder decodes the pixels for use with the vision algorithms.

and (*ii*) decoding the pixel streams for vision application usage. Here, we outline the design aspirations that guide our architecture. We then describe the design of two hardware units – a rhythmic pixel encoder and a rhythmic pixel decoder – and their integration with the existing mobile SoC vision pipeline, as shown in Fig. 4. We also discuss the software runtime for vision application developers to leverage these hardware extensions through policy specification.

Design aspirations: We set four goals that guide the design of our hardware and software extensions.

- Lightweight: Our encoding/decoding techniques should be lightweight (unlike traditional video codecs) so that they can be designed and integrated with minimal resource overhead.
- Scalable: Our system should scale to support the capture of many regions with minimal resource overhead.
- Memory friendly: Our hardware extensions should perform encoding/decoding tasks with efficient memory access patterns and minimal DRAM traffic
- Flexible: Our runtime should allow developers to flexibly and independently specify resolution and update rate needs on a per-region basis.

4.1 Rhythmic Pixel Encoder Architecture

The rhythmic pixel encoder module, shown in Fig. 5, intercepts the incoming pixel stream from an image sensor pipeline and uses developer-specified region labels to encode pixels into an encoded frame. The encoder also generates associated metadata (per-row offset is a count of the number of regional pixels in a row, and EncMask is the output of the region comparison process), which is stored with the encoded frame in DRAM. The app specifies region labels using the designed runtime support in Section 4.3.

We design our encoder as a fully streaming based module that avoids the need for partitions to store individually addressed regions. Instead, our encoder directly operates on a dense raster-scan based pixel stream, produces a sparse encoded stream based on the region labels, and writes the encoded stream to the DRAM, all with *on-the-fly* streaming operation.

4.1.1 Raster-Scan Optimized Sampling: The sampling block operates on the pixel stream and decides whether to sample a pixel in space and time based on whether the pixel is in any of the regions

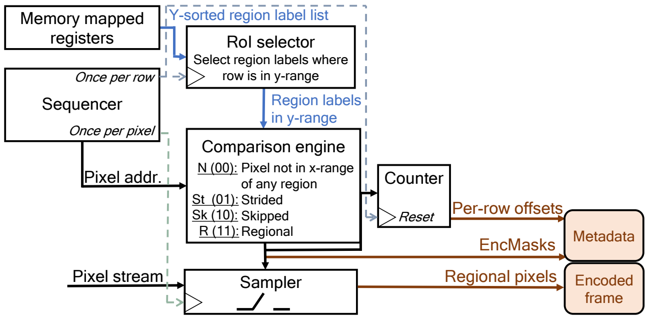


Figure 5: Encoder intercepts the incoming pixel stream and only forwards pixels that match the stride and skip specifications of any region.

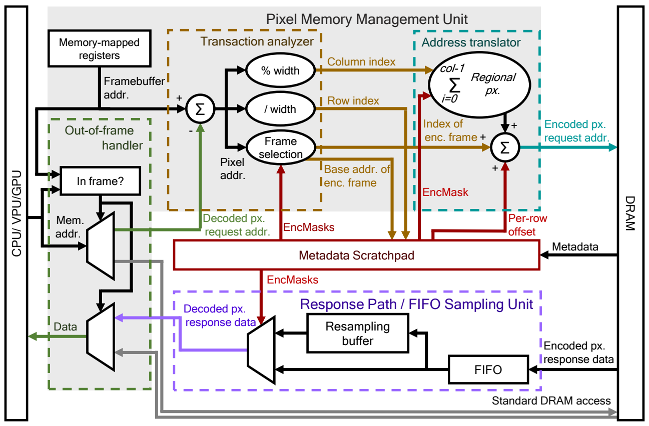


Figure 6: Decoder fulfills pixel requests from the vision app in two steps. (a) The pixel MMU performs address translation to fetch the right set of pixel regions from encoded frame. (b) The FIFO sampling unit reconstructs the original pixel regions from encoded pixel regions and metadata

or not. A naive approach could sequentially compare a pixel's location against every region label. This would be time consuming and would hinder pipeline performance. Although parallelizing the region label checking process would solve performance issues, we find that this exponentially increases the number of resources to support more regions, limiting encoder scalability.

Instead, our approach is to exploit the raster-scan patterns of the incoming pixel stream to reduce and reuse the work of the region search. The Sequencer keeps track of row and pixel location. For a given row, there is a smaller subset regions that are relevant – where the y-index of the pixel is inside of the y-range of the region and matches the vertical stride. The RoI Selector performs this search space reduction, i.e., converting from the region list to a sublist, once per row. Then, from pixel to pixel, the encoder's Comparison Engine only needs to check whether the x-index of the pixel is inside any of the regions in the sublist and matches the horizontal stride. As the sublist is much smaller than the original list, this design choice substantially reduces the level of region-based parallelism needed for the Comparison Engine, saving hardware complexity.

In addition, we perform further optimizations for the comparison engine based on spatial locality. Within a row, if we find that if a pixel belongs to a region, the Sampler can apply the same comparison result for the next *region width* number of pixels. We also simplify the process of finding relevant regions for a row through sorting the regions by their y-indices. This can be done by the rhythmic pixel region app runtime.

4.1.2 Integration with ISP Output. The placement of the encoder has implications on the efficiency of the visual capture. The most energy-efficient integration of the encoder would be on the sensor itself, reducing pixel traffic over the sensor interface and potentially reducing ISP computations. However, this would need a re-design of the ISP, which conventionally expects pixels in frame-based ordering in its algorithms. Thus, for this work, we instead integrate the encoder at the output of the ISP to seamlessly work with existing commercial ISPs.

As with typical ISP operation, the encoder collects a line of pixels before committing a burst DMA write to a framebuffer in the DRAM for efficient and performant memory transaction. Using a framebuffer allows asynchronous access to frame pixels. In the case of the rhythmic pixel architecture, the framebuffer also allows the system to collect multiple frames of data, which the decoder can use to extrapolate regional pixels over a sequence of frames for regions with temporal skip rate. The metadata (per-row offset and EncMask) of the frame is also stored alongside the framebuffer in DRAM. As the EncMask occupies 2 bits per pixel, we note that it occupies 8% of the original frame data (e.g., 500 KB for a 1080p frame). As explored in our evaluation, this amounts to a minimal memory overhead compared to the memory savings.

4.2 Rhythmic Pixel Decoder Architecture

The rhythmic pixel decoder fulfills pixel requests from the vision app, which seek groups of sequential pixels from a decoded framebuffer. While the app forms its request around pixel locations in the decoded address space, the decoder uses the metadata and encoded frames to translate decoded pixel addresses to the DRAM addresses of the encoded frame pixels. This request path is managed by a pixel memory management unit (PMMU), as described below and shown in Fig. 6.

The response path returns the pixel values to the vision application through a FIFO Sampling Unit. If required by spatial stride or temporal skip situations, the FIFO Sampling Unit interpolates encoded data to create decoded pixel values, which it provides to the requesting processor. Otherwise, the decoder returns the pixel value, or a black pixel, if the pixel was not within any of the specified region labels. As described below, the metadata is used to make such determinations.

4.2.1 Pixel Memory Management Unit for Pixel Address Translation: The PMMU works in the same spirit as a traditional memory management unit (MMU); while MMUs perform virtual to physical address conversion, our PMMU translates pixel transactions from decoded to encoded space. Similar to the exception handler of an MMU, the PMMU's Out-of-Frame Handler examines a memory request and determines if it is a valid pixel request, i.e., if the requested memory address is in the decoded framebuffer address space. The Out-of-Frame Handler forwards the transaction if it's a pixel-based one; otherwise, it will bypass for standard memory access.

Meanwhile, the Metadata Scratchpad loads the per-row offset and EncMask information pertaining to the transaction for the four most recent encoded frames. The Transaction Analyzer analyzes the EncMasks of the transaction and generates different sub-requests based on where the encoded pixels are present. Based on the stride (St) and skip (Sk) values of the EncMask bits, the pixel may be in the most recent encoded frame, or one of the recently stored encoded frames. Thus, similar to a virtual memory request, translation of these pixel requests creates sub-requests that are characterized by a base address (of the encoded frame), offset (row and column), and a tag index of which frame hosts the desired pixels.

These sub-requests are fed to a translator. For intra-frame requests, the base address remains the same, whereas for inter-frame requests, the translator modifies it to the appropriate base address. The per-row offset is read from the metadata. The column offset is the count of the number of full regional pixels from the start of the row until that pixel (The number of "11" entries in the EncMask). The translator sums this information to generate the new encoded pixel request corresponding to each sub-request, which will be sent to DRAM.

4.2.2 FIFO Sampling Unit: A FIFO buffers data packets received from a pixel-based DRAM transaction. To prepare a pixel value to service the original request, the engine either dequeues pixel data from the FIFO, re-samples the previous pixel (in the case of stride), or samples a black pixel, based on the EncMask. The unpacker also relays the RLAST control signal to indicate the last transfer in the transaction for proper handshake with the processing unit for the memory request.

4.2.3 Integration with DDR Controller. We integrate the decoder module with the existing DDR controller inside the SoC. By doing so, the decoder can intercept memory traffic coming from any processing element and service requests.

4.3 Developer Support for Rhythmic Pixel Regions

From the point of view of the processing unit – CPU, visual processing unit, or GPU – the rhythmic pixel region architecture preserves the addressing scheme of the original frame-based computing through the decoded framebuffer address. This allows fully transparent use of existing software libraries and hardware accelerators, with no modification needed.

We develop runtime support to allow the developers to flexibly specify region labels. This consists of a `RegionLabel` struct and a `SetRegionLabels()` function for developers to set a list of regions. Region label lists can be set on a per-frame basis or persist across frames. A runtime service receives these calls to send the region label list to the encoder.

```
struct RegionLabel {
   int x, y, w, h, stride, skip;
};
SetRegionLabels(list<RegionLabel>);
```

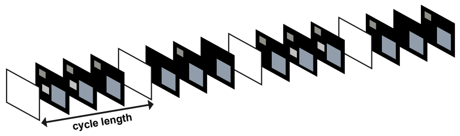


Figure 7: With rhythmic pixel regions, policies can define how different regions in the image can be captured at non-uniform spatial and temporal resolutions. In this illustrated policy, regions shaded in black are are not sampled. To track objects entering/leaving the scene, the system performs a full-frame capture on a "cycle length" periodicity.

4.3.1 Policy-Based Usage of Rhythmic Pixel Regions. Developers can build various policies that autonomously guide the region selection, in the similar spirit to issuing frame configurations with the Frankencamera API [2] or Android's Camera API [3]. Policies can incur different system overheads, leading to system trade-offs. A policy should predict region progression with time as well as a region quality requirements to maximize task performance. A feature-based policy (such as that in our case study of Section 3.4) can use proxies, such as feature scale and feature displacement to estimate spatial and temporal resolutions of regions. Developers can also introduce improved application-specific proxies with other prediction strategies, e.g., with Kalman filters [21].

The process of policy generation and modifying the app around that policy could be cumbersome for an app developer. To reduce the burden, we propose two tiers of developers

Policy Makers: The first tier of developers can specialize in policy development. They can write a wide variety of policies that employ a feature-based approach and/or sophisticated motion-vector based techniques, such as those found in Euphrates or EVA^2 to guide region selection.

Policy Users: The second tier of developers could select a policy directly from a pool based on their app needs.

This dichotomy of policy makers/users follows a typical stratified development paradigm, where high-level developers can simply employ domain-specific libraries (e.g., cuDNN, cuFFT, cuBLAS) that are developed to take advantage of architectural complexity (e.g., with CUDA) by a different set of low-level developers.

Example policy: As described in the case study, policies can center rhythmic pixel regions around features, ensuring that features are captured at sufficient resolution. We define an example policy (Fig. 7) around the concept of a *cycle length*, which is the number of consecutive frames between two full frame captures. These occasional full frame captures can provide contextual information of the entire scene at a slower temporal rate, while feature-based region tracking in the intermediate frames can provide continual coverage of important regions.

The visual features processed by the app – readily available in memory – can be used to determine the pixel regions for the next subsequent frame, centering around the features. For example, as discussed in §3.4, an ORB feature's "size" (or scale) attribute [34] can guide the width and height of its corresponding region, ensuring that the meaningful neighborhood of pixels around the feature is captured (with extra margin to allow for frame-to-frame feature displacement). Furthermore, the "octave"[34] attribute, which

Table 2: System components in the video pipeline

Component	Specification
Camera	Sony IMX274, 4K @ 60 fps
ISP	Demosaic and Gamma correction, 2 Pixels Per Clock
CPU	ARM Cortex-A53 quad-core
GPU	ARM Mali-400 MP2
NPU	Deephi DNN co-processor
DRAM	4-channel LPDDR4, 4 GB, 32-bit

Table 3: Vision tasks and benchmarks

Task	Algorithm	Resolution	Benchmark	#Frames
Visual SLAM	ORB-SLAM2 [31]	4K@ 30 fps	In-house dataset	6000
Pose estimation	PoseNet [5]	720p@ 30 fps	PoseTrack 2017	3792
Face detection	RetinaNet [1]	SVGA@ 30 fps	ChokePoint dataset	22099

describes the texture of the feature, can determine the stride (resolution) parameter for each region. Through the displacement of matched features from frame to frame and measuring the displacement, the system can estimate the movement of a region. The policy can use this feature velocity to determine the temporal rate parameter of pixel regions, sampling fast moving regions more frequently and slow-moving regions less frequently.

We use and adapt this example policy to various visual workloads in our implementation and evaluation. We evaluate different cycle lengths, finding that as the cycle length increases, system efficiency improves, but the errors due to tracking inaccuracy also accumulate, and vice-versa. Cycle length thus becomes an important parameter to govern the tradeoff to meet application needs. The cycle length could also be adaptive, for example, by using the motion in the frame or other semantics to guide the need for more frequent or less frequent full captures. We envision that future policies developed by a wider community of policy makers could substantially improve the opportunities of rhythmic pixel regions.

5 IMPLEMENTATION

5.1 FPGA-Based Encoder and Decoder Integration

Platform: We use Xilinx's reVISION stack platform [45] which implements a end-to-end video pipeline that emulates a standard mobile embedded vision pipeline. As shown in Table 2, we use Sony's IMX274 camera, a mobile class imager used in many smartphones for high fidelity capture, i.e., 4K @ 60fps. For the ISP, we use Xilinx's IP modules for performing demosaicing, gamma correction, and color-space conversion. These ISP blocks operate at a throughput of 2 pixels per clock for real-time performance. The processing subsystem comprises heterogeneous set of computing elements, similar to a mobile SoC. Specifically, the device contains an ARM Cortex-A53 quad-core CPU and an ARM Mali-400 MP2 graphics processing unit (GPU). In addition, we integrate a Deephi DNN co-processor [48] into the FPGA fabric to emulate a neural network accelerator. Finally, the entire system is provisioned with 4 GB LPDDR4 DRAM in the processing subsystem, which we partially leverage for frame buffer capacity. The fully functional pipeline delivers real-time performance of up to 60 fps for video pass-through and up to 30 fps for certain vision tasks, such as face detection. We build our system around Xilinx's reVISION platform [45], which implements an end-to-end video pipeline with major components shown in Table 2.

Encoder and Decoder: We design our encoder IP module with Vivado HLS as a fully-streaming block with AXI-stream interfaces. In our encoder, input/output buffers are FIFO structures with a depth of 16. We find that this depth is enough to meet the 2 pixel-per-clock performance to match ISP performance and to avoid any pipeline stalls.

We also design our decoder with Vivado HLS, utilizing AXI memory-mapped interfaces on the input and output for integration with controllers and processing units. Our decoder operates within the timing budget without introducing extra latency. Both encoder and decoder functionally work in HLS simulations and the entire video pipeline passes Vivado FPGA post-layout timing. In addition to the hardware decoder, we design a software decoder using C++ and OpenCV. The software decoder runs in real-time for a 1080p video stream.

As Xilinx packages the system DDR controller as a part of its Zynq CPU IP module, we could not integrate the decoder between the DDR controller and the xPU as shown in Fig. 4. Instead, we integrate our decoder as a memory-mapped peripheral slave to Zynq SoC. In this integration, the system passes the pointers to the encoded frame and metadata to the decoder to reconstruct the original frame.

5.2 Runtime

Our runtime comprises a standard software stack with a user-space API, a kernel-space driver, and a set of low-level physical registers. We implement region parameters as registers in the encoder/decoder modules inside the SoC. Upon invoking any setter function from the application, the user-space API passes parameters to the kernel-space driver. The driver then writes these parameters to the appropriate registers in the hardware units over an AXI-lite interface.

5.3 Workloads

We study three widely-used vision tasks: (i) Visual SLAM, determining camera position with respect to its surroundings while constructing the map of surroundings. (ii) Human pose estimation, tracking person movement. (iii) Face detection, tracking faces over time. For each task, as shown in Table 3, we choose state-of-the-art algorithms with different input, memory, and computational requirements.

Notably real-time performance of the V-SLAM workload is not attainable on the FPGA's CPU due to the compute-intensive nature of ORB-SLAM2. Therefore, we resort to a simulation-based approach where we run the V-SLAM workload on a desktop computer, generate the region labels, and feed them to the encoder. That said, there are commercial V-SLAM IP cores [9, 41] available in the market that offer real-time performance. We plan to integrate them into our FPGA platform for future studies.

Benchmarks: We use popular publicly available benchmarks for each of these tasks to evaluate their accuracy. Each of these datasets comprise videos with different visual attributes, including illumination variation and occlusion that mimic the real-time scenarios in the wild. In addition, these videos cover a wide range of settings, e.g., indoor/outdoor, crowded/dispersed, and fast/slow motion of objects making them realistic candidates for evaluating rhythmic pixel regions.

To evaluate the potential of rhythmic pixel regions on high-precision visual computing, we also evaluate visual SLAM on a 4K dataset. Since there are no ready-made 4K datasets available for visual SLAM, we create a dataset of 6000 frames spanning a total of 7 indoor video sequences with varying user movement. For portability, we use the 4K camera on a Microsoft Azure Kinect DK [29]. We use an HTC Vive tracker setup to obtain the ground truth pose. The obtained ground truth pose has an offset compared to the original pose since the tracker is at a different location from the camera. We use the information from Kinect's IMU sensor to correct the offset.

For human pose estimation, we use the famous PoseTrack [26] with 3792 frames across all video sequences. Finally, for face detection, we use the ChokePoint [33] dataset which comprise 20 video sequences and 22,099 face images.

Baselines: We tested the workloads against the following baselines. (a) Frame-based computing: The system captures frames at high resolution (FCH: 4K for V-SLAM) or low resolution (FCL: 1080p for V-SLAM) (b) Rhythmic pixel regions (RPx): The system implements rhythmic pixel regions of cycle length "x" (c) Multi-ROI cameras: The system simulates off-the-shelf multi-ROI cameras (d) H.264: The system performs H.264 video compression with the "Baseline" profile and the "5.2 (2160p60)" level for the codec.

As far as we've seen, commercial multi-ROI cameras only support up to 16 regions, likely due to architectural unscalability. For workloads that use more regions, we combine smaller regions into 16 larger regions through k-means clustering. To accurately represent the capabilities of multi-ROI cameras, we do not implement stride or skip adaptations. As an H.264 compression implementation is inaccessible on our FPGA board, we instead use a codec datasheet to form estimations [44]. As compression needs multiple frames to be stored in the memory, the pixel memory footprint and throughput scale accordingly.

5.3.1 Metrics: Here we discuss about different evaluation metrics.
Task Accuracy: We choose standard accuracy metrics from computer vision literature for all of our tasks. For visual SLAM, we use absolute trajectory error and relative pose error metrics as discussed in §3.4. For human pose estimation/face detection, we use the intersection over union (IoU) score as the metric. IoU measures the amount of overlap between the predicted and ground truth bounding boxes. A detection is a true positive (TP) if the IoU score is greater than a certain threshold; otherwise, it is considered as a false positive (FP). Final detection accuracy is the number of true positives among all detections, i.e., TP/(TP + FP), across all the frames, which is known as mean average precision (mAP).

Datarate and Memory Footprint: We build a throughput simulator which takes the region label specification per frame from the application and uses it to generate the memory access patterns of pixel traffic. The simulator counts the number of pixel transactions and directly reports the read/write pixel throughput in bytes/sec. For memory footprint, we measure the size of encoded frame buffers over time.

Table 4: Observed statistics of task and benchmark

Task	Avg. Number of regions	Region size	Stride	Rate
Visual SLAM	973	Min: 70x70 Max: 230x230	Min: 1 Max: 4	Min: 100 ms Max: 33 ms
Face detection	3	Min: 70x63 Max: 270x228	Min: 1 Max: 2	Min: 67 ms Max: 33 ms
Human Pose estimation	4	Min: 161x248 Max: 324x512	Min: 2 Max: 4	Min: 100 ms Max: 33 ms

Overhead: We use Xilinx Vivado [46] to determine the area and power overhead of our encoder and decoder modules. We use the resource utilization from the post-layout design as a proxy to report the area overhead; For power overhead, we use the numbers from Vivado power analysis tool.

5.3.2 Policy/Parameter Choices: We outline our choices below.
Region selection: As outlined in §4.3, we use feature characteristics to guide the selection of region labels to evaluate our tasks. For V-SLAM, region size is derived directly from the feature size attribute, while spatial and temporal resolutions are derived from the octave attribute and feature displacement respectively. We use face trajectory for face detection and skeletal pose joints for human pose estimation for determining the regions. Spatial and temporal resolutions are calculated based on the region's size and motion, respectively.

Cycle length: We evaluate the effectiveness of the example cycle-based policy with cycle lengths of CL=5, 10, and 15.

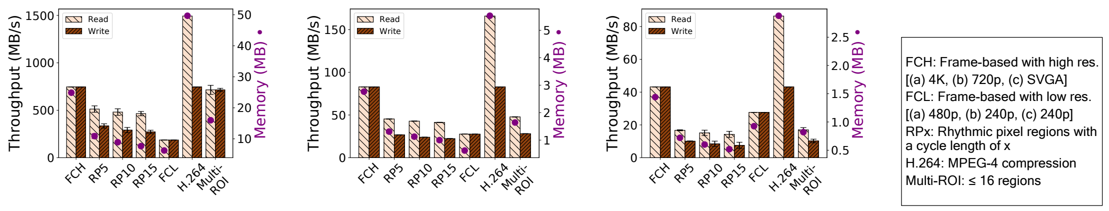
6 EVALUATION

6.1 The Use of Rhythmic Pixel Regions Is *Flexible*

Our runtime successfully allows apps to express region labels without any restrictions on the number, size, and resolution of the regions, as shown in Table 4. The size of these regions vary based on the semantics, e.g., nearness/farness of a face with respect to the camera. These regions are also sampled at different spatio-temporal resolutions based on their content.

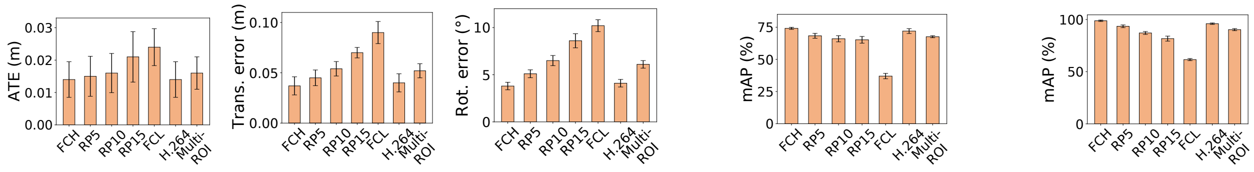
Algorithms/Apps are still *reliable*: With flexible region specification, the app now deals with only the pixels within the regions as opposed to the pixels in the entire frame. As shown in Fig. 9, we find that apps can still reliably perform their tasks with a slight accuracy loss compared to frame-based computing at high resolutions on uncompressed (FCH) or compressed (H.264) frames. Comparatively, frame-based computing at low resolutions (FCL) performs poorly, with significantly raised errors for all of the visual workloads.

For our workloads, we observe a trade-off between cycle length and accuracy. As shown in Fig. 9, while higher cycle lengths help discard more pixels, they also take a toll on the task accuracy. Moderate cycle lengths, e.g., CL=10, strike a reasonable balance between energy savings and task accuracy.



(a) Visual SLAM **(b) Human pose estimation** **(c) Face detection**

Figure 8: Rhythmic pixel regions reduces pixel memory traffic by generating sparser pixel streams and reduces the memory footprint by generating smaller frame buffers. The reduction is more with higher cycle lengths.



(i) Trajectory Error **(ii) Translational error** **(iii) Rotational Error** **(i) mAP** **(i) mAP**

(a) Visual SLAM **(b) Human Pose Estimation** **(c) Face Detection**

Figure 9: There is a trade-off between cycle length and task accuracy. In V-SLAM, we observe high standard deviation. This indicates future opportunity to adaptively reduce cycle length to improve accuracy for scenes with high motion.

This raises a question: How much accuracy loss is acceptable? There is no set standard in the vision community, as it depends on the app context. To that end, the flexibility of policy choice allows developers to heuristically set accuracy expectations for their specific apps. Our evaluated workloads result in roughly 5% accuracy loss for moderate cycle lengths, i.e., CL=10. Future investigation into region selection policies and adapted algorithms could further improve accuracy.

We can also observe a higher standard deviation for raised cycle lengths, especially for V-SLAM. This indicates scene-based variability in accuracy loss. We analyze the scenes to find that the scenes reporting low accuracy loss are fairly static in nature, whereas the scenes with high accuracy loss have rapid scene motion. Other scenes in our benchmark resulted in an average accuracy loss. Ideally, in future integrations, this information could be leveraged, e.g., using accelerometer data and/or scene knowledge to guide the region selection policy.

6.2 The Use of Rhythmic Pixel Regions Is *Memory Friendly*

Discarding pixels relieves memory interfaces: Rhythmic pixel regions substantially reduce the data traffic across DDR interfaces, as shown in Fig. 8. With higher cycle lengths, the system discards more pixels, leading to further reduction in memory bandwidth. Specifically, we find that memory traffic decreases by 5-10% with every 5 step increase in cycle length.

Notably, we find that the work saving techniques such as search-space reduction in our encoder works for a broad spectrum of videos Our evaluated datasets, which contain different videos captured in the wild, comprise images with regions spread across the entire image as well as images with regions confined to a few areas within the image. The encoder saves work in both cases. In the former case, our encoder saves work by reducing the number of region comparisons for each row. In the latter case, the encoder saves work by skipping region comparison entirely for those rows where there are no regions.

The pixel memory throughput for the multi-ROI baseline is larger than that of rhythmic pixel regions for face detection and pose estimation workloads and substantially higher for visual SLAM. Video compression generates a substantially higher amount of memory traffic since it operates on multiple frames.

Discarding pixels reduces memory footprint: Rhythmic pixel regions not only relieve the memory interfaces but also helps reduce the pixel memory footprint of the vision pipeline, with similar trends, also shown in Fig.8. Specifically, with our paradigm, the average frame buffer size reduces by roughly 50% compared to frame-based computing. Frame buffers are storage intensive components in the vision pipeline; with smaller frame buffers, a system can not only reduce storage energy, but also potentially afford to store buffers locally inside the SoC itself, thereby reducing reliance on DRAM. Metadata produced by the encoder incurs minimal memory overhead. Specifically, EncMask and row offsets amount to 8% of pixel buffer storage for a 1080p frame.

Energy efficiency implications of rhythmic pixel regions: Memory efficiency begets energy efficiency; as is widely acknowledged in the computer architecture community, systems expend significant energy to move data in and out of memory. Based on first order modeling (see Appendix), with an assumption of 300 pJ to read a pixel and 400 pJ to write a pixel, the reduced interface traffic of rhythmic pixel regions reduces energy consumption by 18 mJ per frame for RP10 on V-SLAM at 4K and 30 fps. This reduces power consumption by 550 mW. The precise energy savings will depend on system characteristics that are highly specific to the device, operating system, and application usage pattern. We leverage the coarse model to contextualize the benefits of reducing pixel memory throughput.

Further energy reduction could result from deeper investigation in tuning vision algorithms and architectures to reduce their computational workload on lower resolution workloads, e.g., avoiding computation on zeroed pixels. Power reduction through sparse computing has been shown to be effective strategy [7, 13, 50], which is a focus of future work.

6.3 The Hardware Extensions Are *Lightweight and Scalable*

Since real-estate is a precious resource on the SoC, our hardware extensions need to be lightweight and scalable to support a multitude of regions without taking too many resources.

Encoder scales well with number of regions: As shown in Table 5, the encoder is able to support additional regions without needing significantly more transistors. This is because our encoder utilizes a hybrid architecture that uses the processor for pre-sorting at the OS level and specialized architecture for shortlisting regions. This significantly reduces the number of comparisons needed, thereby favoring scalability. On the other hand, resource footprint for a fully-parallel based encoder design substantially increases with more regions to such an extent that they cause synthesis issues on the FPGA.

Decoder is agnostic to number of regions: As the decoder uses bitmasks (EncMask) instead of region labels, our decoder design is agnostic to the number of regions. Specifically, it needs 699 LUTS, 1082 FFs, and 2 BRAMs (18Kb) for 1080p decoding, regardless of the number of supported regions.

Encoder/decoder are performant: Encoder and decoder designs do not affect the pipeline performance. Our end-to-end system with encoder and decoder runs in real-time; our encoder meets the 2 PPC constraint of the capture pipeline. For the decoder, since it intercepts every pixel transaction and appropriately modifies the transaction response, it will add a few clock cycles of delay when returning the response. We find that this delay is the order of a few 10s of ns and is negligible compared to the frame compute time – typically 10s of ms – of vision workloads.

Our alternative software decoder also runs in real time, consuming a few ms of CPU time for a 1080p frame where 30% of the pixels are regional pixels. The software decoder linearly scales in time to the amount of regional pixels.

Encoder/decoder are power-efficient: Our hardware extensions need to be power-efficient so as not to outweigh potential energy savings. Our encoder consumes 45 mW for supporting 1600 regions, which entails less than 7% of standard mobile ISP chip power (650 mW). Our decoder consumes < 1 mW of power. These power consumption estimates are based off of FPGA targets; power-efficiency improves for ASIC targets, which we will study as future work.

7 FUTURE DIRECTIONS

DRAM-less Computing: The paradigm of rhythmic pixel regions significantly reduces the average size of the frame buffer.

Table 5: Resource utilization for different encoder designs

Type	#Regions	Resources		
		#LUTs	#FFs	#BRAMs
Parallel	100	4644	5935	6
Parallel	200	8635	10935	6
Parallel	400	16251	20685	6
Parallel	1600	No Synth	No Synth	No Synth
Hybrid	100	942	1189	11
Hybrid	200	949	1190	11
Hybrid	400	944	1191	11
Hybrid	1600	952	1186	11

This presents an opportunity to store frame buffers in the local SoC memory when not dealing with full frame captures. By doing so, the system would be less dependent on DRAM, thereby significantly reducing data movement. In the future, we will study such an integrated memory-compute system and its effectiveness in reducing memory datarate.

Rhythmic Pixel Camera: In this work, we place our encoder after the camera capture inside the SoC. However, the MIPI interface which sends the pixels from camera to SoC is still burdened with data movement. To this end, we plan to study an integration of our encoder inside the camera module to reduce MIPI interface traffic for further energy savings.

Pixel Region Selection Policies: We will further explore potential policy tradeoffs of rhythmic pixel regions. We will study motion estimation techniques in guiding the region label selection. Machine learning, e.g., reinforcement learning, could also autonomously guide region selection. We believe these will unlock the potential of rhythmic pixel regions for high-precision visual computing with efficient memory use.

8 CONCLUSION

Frame-based computing is limited by fixed resolutions and frame rates. We propose an architecture to sample regions with different spatiotemporal rhythms, and a runtime to allow region selection control. We efficiently encode regions before writing them to memory. We decode the compact data on-the-fly before feeding it to the vision apps for seamless use. This relieves interface traffic and memory footprint, towards system energy-efficiency. With our hardware/software interfaces, we take early steps to transform the norm of imaging/vision trends towards precise, performant, and efficient vision for AR/VR systems.

ACKNOWLEDGMENTS

The authors are grateful for comments made by anonymous reviewers and the paper shepherd Dr. Vijay Janapa Reddi. This material is based upon work supported by the National Science Foundation under grants 1942844 and 1909663.

APPENDIX

A.1 Rhythmic Pixel Regions in Action

In Figs. 10-15, we show the progression of frames for different workloads in action.

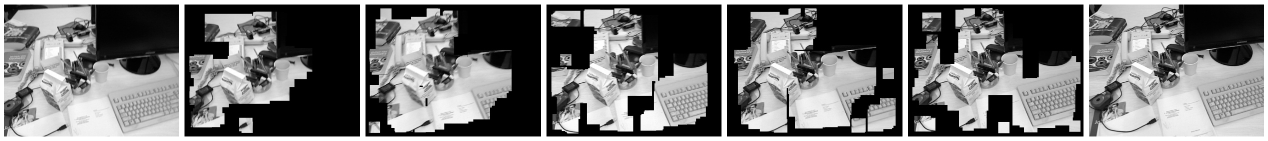

(a) Frame 1 (100%) (b) Frame 2 (37%) (c) Frame 3 (31%) (d) Frame 4 (34%) (e) Frame 5 (27%) (f) Frame 6 (35%) (g) Frame 7 (100%)

Figure 10: Task: Visual SLAM; Benchmark: TUM freiburg1-xyz

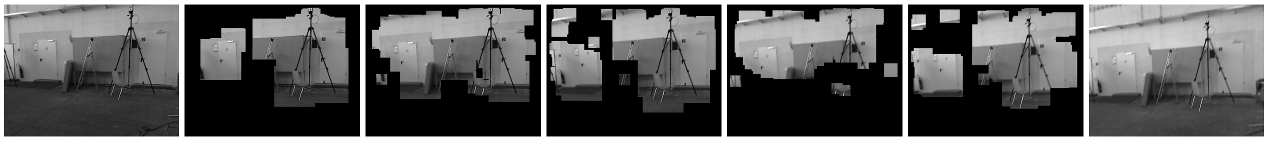

(a) Frame 1 (100%) (b) Frame 2 (42%) (c) Frame 3 (36%) (d) Frame 4 (39%) (e) Frame 5 (33%) (f) Frame 6 (40%) (g) Frame 7 (100%)

Figure 11: Task: Visual SLAM; Benchmark: TUM freiburg1-floor

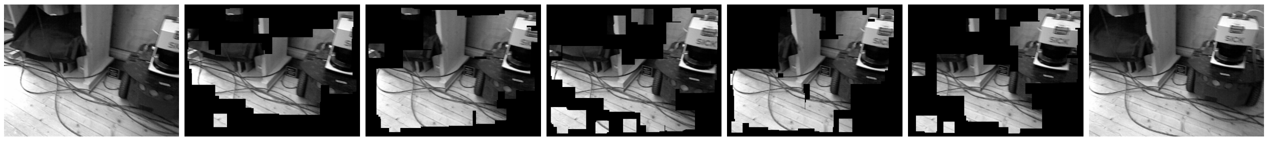

(a) Frame 1 (100%) (b) Frame 2 (44%) (c) Frame 3 (41%) (d) Frame 4 (37%) (e) Frame 5 (32%) (f) Frame 6 (35%) (g) Frame 7 (100%)

Figure 12: Task: Visual SLAM; Benchmark: TUM freiburg2-360-kidnap-secret

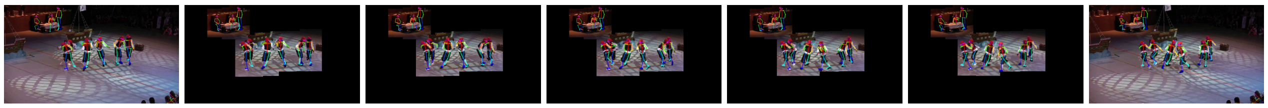

(a) Frame 1 (100%) (b) Frame 2 (31%) (c) Frame 3 (24%) (d) Frame 4 (29%) (e) Frame 5 (23%) (f) Frame 6 (33%) (g) Frame 7 (100%)

Figure 13: Task: Human pose estimation; Benchmark: PoseTrack 2017 024575-mpii

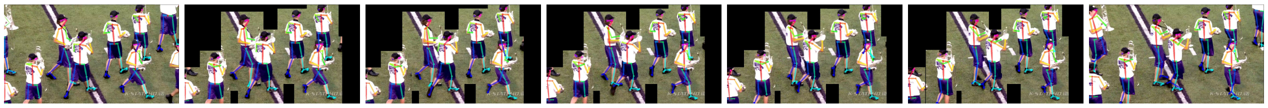

(a) Frame 1 (100%) (b) Frame 2 (39%) (c) Frame 3 (23%) (d) Frame 4 (36%) (e) Frame 5 (21%) (f) Frame 6 (41%) (g) Frame 7 (100%)

Figure 14: Task: Human pose estimation; Benchmark: PoseTrack 2017 015301-mpii

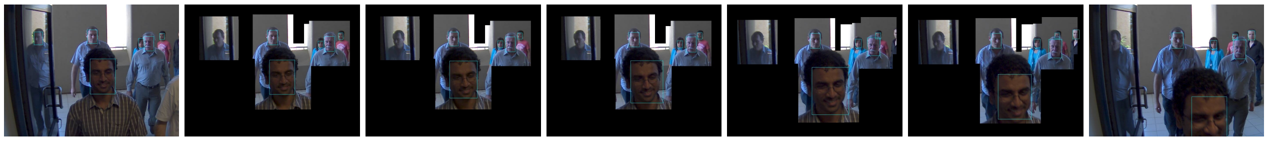

(a) Frame 1 (100%) (b) Frame 2 (28%) (c) Frame 3 (26%) (d) Frame 4 (22%) (e) Frame 5 (37%) (f) Frame 6 (31%) (g) Frame 7 (100%)

Figure 15: Task: Face detection; Benchmark: Chokepoint dataset P2E-S5

A.2 Energy Model

The power measurements directly taken off from the Xilinx Zynq FPGA board, which consumes power on the order of tens of watts, are not representative of mobile systems, which consume only a few watts of power for typical vision tasks. Therefore, we construct a coarse energy model based on reported numbers in component datasheets and computer architecture literature to estimate system

Table 6: Energy-per-pixel of various components in the vision pipeline. As is widely acknowledged, communication cost is atleast three orders of magnitude more than compute cost.

Component	Energy (pJ/pixel)
Sensing	595 [8, 22]
Communication (SoC - DRAM)	2800 [17, 24, 27, 35, 36, 47]
Storage	677 [12, 25, 28, 42]
Computation (per MAC)	4.6 [16]

power. Precise energy savings will depend on system characteristics that are highly specific to the device, operating system, and application usage pattern. We construct the first-order linear model to approximate power consumption so as to contextualize the benefits of reducing pixel memory throughput in a mobile system.

In particular, we break down the system energy into four components: sensing, computation, storage, and communication, as shown in Table 6. Sensing requires an energy of roughly 600 pJ/pixel [8, 22], mostly drawn from three components: pixel array, read-out circuits, and analog signal chain. DRAM storage on standard mobile-class memory chips (8 Gb, 32-bit LPDDR4) draws 677 pJ/pixel for writing and reading a pixel value [28]. This roughly divides into 300 pJ/pixel for reading and 400 pJ/pixel for writing [12, 25, 42]. Communication over DDR interfaces incur 4 nJ/pixel, mostly due to operational amplifiers on transmitters and receivers. We measure the interface power dissipation [47] on 4-lane CSI interfaces and LPDDR4 interfaces by inputting several datarates. From this information, we construct a linear-regression model to estimate the energy per pixel to be 1 nJ/pixel over CSI and 3 nJ/pixel [17, 24, 27, 35, 36] over DDR. We use 5 pJ per MAC operation [16] to estimate compute energy.

We use FPGA measurements of pixel memory read/write throughput and computation patterns to apply the runtime behavior of the system to our energy model. For communication and storage, we take memory traffic/footprint values from Fig 7 and apply them to the model. We obtain sensing energy by applying frame resolution from Table 3 on our model. For compute energy, we take the number of MAC operations for our CNN workloads and apply them to the model.

REFERENCES

[1] 1996scarlet. RetinaNet. https://github.com/1996scarlet/faster-mobile-retinaface.

[2] Andrew Adams, Eino-Ville Talvala, Sung Hee Park, David E Jacobs, Boris Ajdin, Natasha Gelfand, Jennifer Dolson, Daniel Vaquero, Jongmin Baek, Marius Tico, et al. The frankencamera: an experimental platform for computational photography. In *ACM SIGGRAPH*. 2010.

[3] Android. Android Camera API documentation. https://developer.android.com/guide/topics/media/camera.

[4] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. Eva^2: Exploiting temporal redundancy in live computer vision. In *ACM/IEEE 45th Annual Int. Symp on Computer Architecture (ISCA)*, 2018.

[5] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Real-time multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[6] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proc. of the 13th ACM Conf. on Embedded Networked Sensor Systems*, 2015.

[7] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[8] Jaehyuk Choi, Seokjun Park, Jihyun Cho, and Euisik Yoon. An energy/illumination-adaptive CMOS image sensor with reconfigurable modes of operations. *IEEE Journal of Solid-State Circuits*, 2015.

[9] EETimes. Tensilica's New Vision/AI DSP Guns for SLAM. https://www.eetimes.com/tensilicas-new-vision-ai-dsp-guns-for-slam.

[10] Yu Feng, Paul Whatmough, and Yuhao Zhu. Asv: accelerated stereo vision system. In *Proc. of the 52nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2019.

[11] Gallego, Guillermo and Delbruck, Tobi and Orchard, Garrick and Bartolozzi, Chiara and Taba, Brian and Censi, Andrea and Leutenegger, Stefan and Davison, Andrew and Conradt, Jörg and Daniilidis, Kostas and others. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019.

[12] Saugata Ghose, Abdullah Giray Yaglikçi, Raghav Gupta, Donghyuk Lee, Kais Kudrolli, William X Liu, Hasan Hassan, Kevin K Chang, Niladrish Chatterjee, Aditya Agrawal, Mike Connor, and Onur Mutlu. What your dram power models are not telling you: Lessons from a detailed experimental study. *Proc. of the ACM on Measurement and Analysis of Computing Systems*, 2018.

[13] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and TN Vijaykumar. Sparten: A sparse tensor accelerator for convolutional neural networks. In *Proc. of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[14] Google. Take off to your next destination with Google Maps. https://www.blog.google/products/maps/take-your-next-destination-google-maps.

[15] Mohit Gupta, Amit Agrawal, Ashok Veeraraghavan, and Srinivasa G Narasimhan. Flexible voxels for motion-aware videography. In *European Conference on Computer Vision*, 2010.

[16] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proc. of the 37th annual intl. symp on Computer architecture*, 2010.

[17] Ron Ho, Kenneth W Mai, and Mark A Horowitz. The future of wires. *Proc. of the IEEE*, 2001.

[18] Tobias Höllerer and Steve Feiner. Mobile augmented reality. *Telegeoinformatics: Location-based computing and services*, 21, 2004.

[19] Jinhan Hu, Jianan Yang, Vraj Delhivala, and Robert LiKamWa. Characterizing the reconfiguration latency of image sensor resolution on android devices. In *Proc. of the 19th International Workshop on Mobile Computing Systems & Applications*, 2018.

[20] Iaian Richardson. *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia.* 2004.

[21] Odrika Iqbal, Saquib Siddiqui, Joshua Martin, Sameeksha Katoch, Andreas Spanias, Daniel Bliss, Suren Jayasuriya, and SenSIP Center. Design and fpga implementation of an adaptive video subsampling algorithm for energy-efficient single object tracking. In *IEEE Int Conf on Image Processing*, 2020.

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

[23] Chaochao Lu, Michael Hirsch, and Bernhard Scholkopf. Flexible spatio-temporal networks for video prediction. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2017.

[24] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir. Interconnect-power dissipation in a microprocessor. In *Proc. of the 2004 international workshop on System level interconnect prediction*, 2004.

[25] Krishna T Malladi, Frank A Nothaft, Karthika Periyathambi, Benjamin C Lee, Christos Kozyrakis, and Mark Horowitz. Towards energy-proportional datacenter memory with mobile dram. In *39th Annual Int. Symp. on Computer Architecture (ISCA)*. IEEE, 2012.

[26] Max Planck Institute for Informatics, University of Bonn. PoseTrack Dataset and Benchmark. https://posetrack.net/.

[27] James D Meindl, Jeffrey A Davis, Payman Zarkesh-Ha, Chirag S Patel, Kevin P Martin, and Paul A Kohl. Interconnect opportunities for gigascale integration. *IBM journal of research and development*, 2002.

[28] Micron technologies. Micron system power calculators. https://www.micron.com/support/tools-and-utilities/power-calc.

[29] Microsoft. Azure Kinect DK. https://www.microsoft.com/en-us/p/azure-kinect-dk/8pp5vxmd9nhq?activetab=pivot%3aoverviewtab.

[30] MIPI Alliance. MIPI Camera Serial Interface 2 (MIPI CSI-2). https://www.mipi.org/specifications/csi-2.

[31] Mur-Artal, Raúl, Montiel, J. M. M. and Tardós, Juan D. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Trans. on Robotics*, 2015.

[32] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *Proc. of the 15th Annual Int. Conf. on Mobile Systems, Applications, and Services*, 2017.

[33] NICTA. ChokePoint Dataset. http://arma.sourceforge.net/chokepoint/.

[34] OpenCV. OpenCV KeyPoint Class Reference. https://docs.opencv.org/3.4/d2/d29/classcv_1_1KeyPoint.html.

[35] Dhinakaran Pandiyan and Carole-Jean Wu. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *2014 IEEE Int. Syump. on Workload Characterization (IISWC)*. IEEE, 2014.

[36] Vijay Raghunathan, Mani B Srivastava, and Rajesh K Gupta. A survey of techniques for energy efficient on-chip communication. In *ACM Proc. of the 40th annual Design Automation Conference*, 2003.

[37] Dikpal Reddy, Ashok Veeraraghavan, and Rama Chellappa. P2c2: Programmable pixel compressive camera for high speed imaging. In *CVPR 2011*.

[38] Richard Szeliski. *Computer Vision: Algorithms and Applications 1st ed.* 2010.

[39] Stemmer Imaging. Teledyne DALSA Piranha4 - Dual-Line-CMOS line camera. https://www.stemmer-imaging.com/en/products/series/teledyne-dalsa-piranha4.

[40] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012.

[41] Synopsys. Augmenting Your Reality with Deep Learning. https://www.synopsys.com/designware-ip/technical-bulletin/augmenting-your-reality-dwtb_q318.html.

[42] Thomas Vogelsang. Understanding the energy consumption of dynamic random access memories. In *43rd Annual IEEE/ACM Int Symp. on Microarchitecture*. IEEE, 2010.

[43] Wikipedia. Foveated Rendering. https://en.wikipedia.org/wiki/Foveated_rendering.

[44] Xilinx. H.264/H.265 Video Codec Unit v1.2. https://www.xilinx.com/support/documentation/ip_documentation/vcu/v1_2/pg252-vcu.pdf.

[45] Xilinx. reVISION Getting Started Guide 2018.3 (UG1265). https://github.com/Xilinx/reVISION-Getting-Started-Guide.

[46] Xilinx. Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html.

[47] Xilinx. Xilinx Power Estimator. https://www.xilinx.com/products/technology/power/xpe.html.

[48] Xilinx. Zynq DPU v3.2. https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_2/pg338-dpu.pdf.

[49] ximea. Multiple ROI cameras. https://www.ximea.com/support/wiki/allprod/Multiple_ROI.

[50] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE Int. Symp. on High Performance Computer Architecture (HPCA)*, 2020.

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