Accelerating Silent Witness Storage

Mahadev Satyanarayanan, *Carnegie Mellon University, Pittsburgh, PA, 15213, USA*
Ziqiang Feng, *Google, Mountain View, CA, 94043, USA*
Shilpa George, Jan Harkes, Roger Iyengar, and Haithem Turki, *Carnegie Mellon University, Pittsburgh, PA, 15213, USA*
Padmanabhan Pillai, *Intel Labs, Santa Clara, CA, 95054, USA*

We propose hardware acceleration for a new edge computing abstraction called a Silent Witness. *This abstraction embodies a severe asymmetry in the ease of write versus read operations. Surveillance data from one or more video cameras are continuously encrypted and recorded, but the decrypting, processing, or transmission of that data only occurs under stringent privacy controls. For the new search workloads of such a system,* decode-enabled storage *alleviates the scalability bottleneck imposed by frequent decoding of data. Our experiments show throughput improvements up to 3.5× for typical search workloads of a Silent Witness.*

The tension between artificial intelligence (AI) and privacy is a topic of considerable public debate, especially regarding video surveillance for public safety. A major concern is that voluminous data collected for an ostensibly benign purpose may be subverted. In this article, we introduce a new computing abstraction that alleviates this concern by enforcing a strict "need to know" requirement on data processing. By construction, this abstraction implies severe runtime performance degradation. We show that application-specific integrated circuit (ASIC)-based hardware acceleration embedded in storage can greatly reduce this performance degradation.

Video surveillance is valuable in fighting crime. It can deter criminal activity,[1,2] give clues for solving crimes,[3] and provide forensic evidence for criminal trials.[4] For example, surveillance videos were crucial to discovering the Boston Marathon bombers in 2013,[5] and in successfully prosecuting them.[6] At the same time, there are legitimate public concerns about the privacy compromises implicit in widespread video surveillance. For example, in the aftermath of the Boston Marathon bombing, a spokesperson for the American Civil Liberties Union (ACLU) stated[7]:

> "What does trigger privacy concerns is the city of Boston installing a network of cameras, some in residential neighborhoods, that enable law enforcement to track individual people from the moment we leave our homes in the morning until the moment we return at night, seeing basically everywhere we went and everything we did."

WE PROPOSE A NEW EDGE COMPUTING ABSTRACTION CALLED SILENT WITNESS *THAT ENCRYPTS AND RECORDS DATA CONTINUOUSLY FROM ONE OR MORE VIDEO CAMERAS, BUT ONLY DECRYPTS, PROCESSES, OR TRANSMITS THAT DATA UNDER STRICT PRIVACY CONTROLS.*

Can we do better? Can we preserve the crime fighting benefits of video surveillance without compromising privacy? Toward this goal, we propose a new edge computing abstraction called *Silent Witness* that encrypts and records data continuously from one or more video

Digital Object Identifier 10.1109/MM.2022.3193048
Date of publication 21 July 2022; date of current version 28 October 2022.

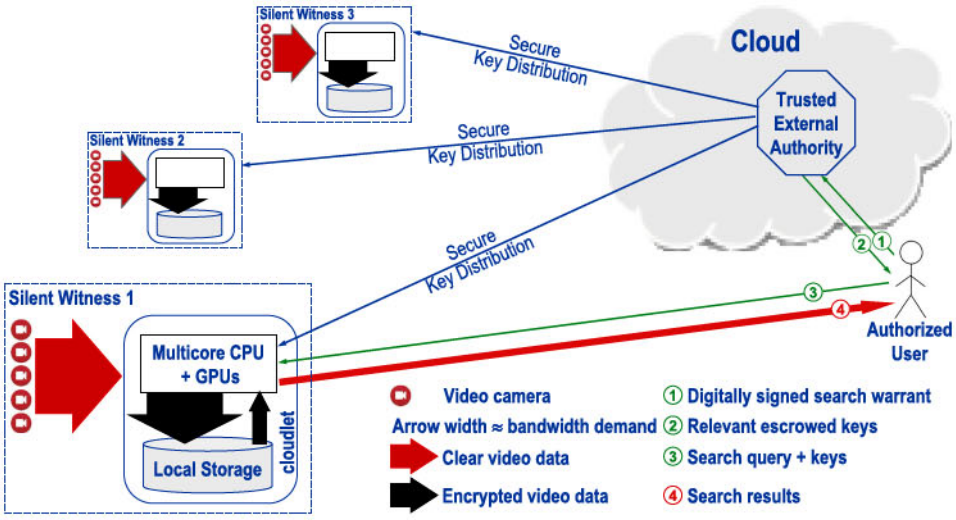

FIGURE 1. Silent Witness system architecture.

cameras, but only decrypts, processes, or transmits that data under strict privacy controls. We describe this new abstraction and its use as an edge node of a distributed system (the "Concept and System Architecture" section). The focus of this paper is hardware acceleration of new input/output (I/O) workloads that arise in this context (the "Forensic Discard-based Search" and "Search Workload Attributes" sections). Our analysis ("The Case For Decode-Enabled Storage" section) shows that decoding of data from storage is a scalability bottleneck for a Silent Witness. To overcome this bottleneck, we propose (the "Decode-Enabled Storage" section) and evaluate (the "Evaluation" section) *decode-enabled storage*. Our experiments show throughput improvements up to 3.5× for typical search workloads of a silent witness.

SILENT WITNESS

Concept and System Architecture

Sensor data captured by a Silent Witness are transmitted over a local area network to a nearby cloudlet,[8,9] where it is immediately encrypted and written to storage. No content-based processing (such as indexing) is done. This represents an extreme point in the tradeoff space between privacy and runtime performance. For a skeptical public that is wary of experts, we conjecture that a blanket commitment to avoid any form of content-based processing at data ingest is much easier to understand than more sophisticated privacy policies. The brutal simplicity of the commitment, and its ease of verification by inspection of source code by trusted third parties, can help in establishing public trust in the Silent Witness concept. This approach runs counter to widely accepted practice in processing surveillance video today,[10,11,12] but it offers the best chance to thread a viable path through this sociolegal minefield.

We focus on video data because it is the most challenging. However, the concept of a Silent Witness is applicable to any kind of sensor data. Bandwidth scalability forces edge-based realization of this abstraction. A single 4K-resolution camera at 30 frames per second (FPS) has a bandwidth demand of roughly 30 Mbps (https://www.synopi.com/bandwidth-required-for-hd-fhd-4k-video). Continuously transmitting video from multiple cameras to the cloud is simply not scalable. After some retention period, old data on a cloudlet is overwritten by fresh data from video cameras. Most data are overwritten without ever being decrypted, viewed, or processed. Retention periods on the order of a few weeks to a few months are easily achievable today. Only a miniscule fraction of captured data is ever exported from a Silent Witness.

The key used to encrypt fresh incoming data is changed frequently (e.g., hourly), and only the most recent key is available locally. A trusted external authority (TEA) periodically delivers unique random keys to each Silent Witness. The TEA escrows the keys, and releases them only under very restrictive "need-to-know" conditions. For example, a law enforcement agent who obtains a search warrant will only be given the keys necessary to decrypt data from specific cameras and time periods. Obtaining a search warrant is a heavyweight process that typically involves judicial review in a democratic society. Since keys are changed frequently and only the latest key is retained locally, even physical capture of a Silent

Witness will only compromise the small subset of data encrypted with the current key. Older data remains secure, because only the TEA has the keys to decrypt that data. In addition, technologies such as Intel SGX (i.e., secure enclaves) can provide an additional level of security for the current key.

Figure 1 shows how multiple Silent Witnesses can be integrated with a cloud-based TEA. Transport layer security-based end-to-end security is assumed for all communication links. Secure key distribution channels from the TEA are used to periodically deliver encryption keys to cloudlets.

Forensic Discard-Based Search

By definition, a Silent Witness does not perform any processing of video data at capture time. Its value as a privacy-preserving abstraction relies on this assumption. Even when the TEA releases relevant keys for an authorized request, access is restricted to the narrowest scope of data that meets the "need to know" criterion. These privacy-motivated constraints violate the basic assumptions underlying cloud-based video processing today. The norm today is to process raw video data only at ingest, and to never reprocess it again.

Since indexing is forbidden, how does a solitary user search through many hours or days of raw video data to which she is granted keys by the TEA? Crowd-sourcing is not applicable here because of privacy restrictions—the right to view the data is only granted to specific individuals or teams. The only feasible approach under these constraints is *discard-based search*.[13–15] This human-in-the-loop approach uses *query-specific content-based computation* for interactive search of visual data. This approach has been extended to support *just-in-time indexing*[16] and deep neural networks (DNNs).[17] We very briefly summarize this approach here, as background for the rest of the article.

The central idea in this approach is *early discard*. This involves discovering as cheaply as possible in the pipeline from storage to user whether a data item is part of the result set for the current query. The content-based computation for early discard is composed of code components called *filters* that map to individual terms of a search query. Filters span a wide range from simple features such as color and texture to more complex features that are expressed as DNNs. Query execution can be optimized by dynamic filter reordering, using runtime measurements of filter execution cost and selectivity.[13]

Figure 1 shows a user obtaining keys from the TEA (steps ① and ②), and then presenting them to a Silent Witness along with a search query (③); the results from the query are streamed directly back to the user as they are generated (④). A user may abort query processing once she has seen a sufficient number of results to refine the query. This iterative process is referred to as *interactive data exploration*.[16]

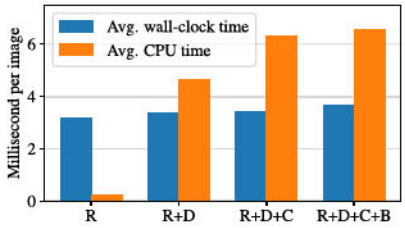

FIGURE 2. Impact of image decoding on scalability. R: Read files from disk. D: Decode JPEG to RGB. C: Color filter for redness. B: Bus detection with DNN. The experiments were run on a cloudlet consisting of a 4-core slice of a server with two Intel Xeon E5-2699v3 processors (2.30 GHz), an NVIDIA GTX 1080 Ti GPU, and a Seagate 4-TB disk drive (7200 RPM, SATAv3). This configuration reflects the per-drive resources of a typical 2-socket server with 8–12 direct-attached disks. CPU times are higher than wall-clock times because multiple (4) cores are used.

Search Workload Attributes

A Silent Witness continuously encrypts and records video received from cameras. This background workload can benefit from hardware acceleration for encryption, which is assumed without further discussion in this article.

One or more foreground workloads of discard-based search augment the background workload. Previous work[15] shows that discard-based search workloads have similarities and differences relative to well-known Map-Reduce workloads. The Reduce step involves no computation, but merely presents results to the user. Like the Map phase, discard-based search offers embarrassing parallelism. However, unlike Map-Reduce, a discard-based search is not a batch operation that runs to completion. Rather, it is an interactive operation that is aborted as soon as the user has gained sufficient insight to pose a better query. Unlike Map-Reduce, where a data item is processed exactly once, discard-based search involves repeated reprocessing of the same data items as the query evolves.

THE CASE FOR DECODE-ENABLED STORAGE

The first step of any early-discard filter pipeline is decoding the data item from its on-storage format (e.g., JPEG, PNG, MP4, etc.) to a bitmap representation.

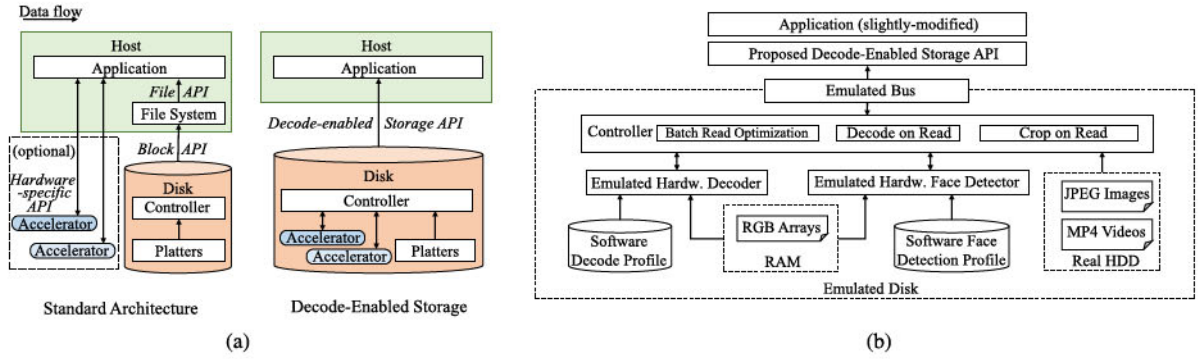

FIGURE 3. Decode-enabled storage. (a) Implementation. (b) Emulation.

Data volumes are usually too large to retain decoded data items in memory or on disk. Hence, decoding has to be done repeatedly: once for each data item, on each iteration of discard-based search. Decoding visual data thus becomes a frequent operation, offering the opportunity for performance optimization through hardware acceleration.

The performance impact of frequent decoding can be seen from a simple example. Consider the query "Show frames that contain red buses." This query is expressed using two filters: 1) a color filter tuned to a broad range of shades of red; and 2) a DNN that detects the class "bus." Detecting a red patch of color is computationally very cheap, while DNN execution is many orders of magnitude more expensive. Hence, the obvious early-discard tactic is to check for a red patch first, and only run the expensive DNN on the subset of frames with red. The human in the loop can eliminate false positives: i.e., frames with a red patch and a bus, but not a red bus.

The filter pipeline for this discard-based search consists of four main operations: (R) reading encoded images from disk, (D) decoding into pixel arrays, (C) execution of color filter, and (B) execution of the bus detection DNN. Figure 2 shows the wall-clock and central processing unit (CPU) times of these four steps in processing 50,000 images from the YFCC100 M dataset.[18] The small difference between "R+D+C" and "R+D+C+B" shows the power of early discard: the DNN is only applied to about 3% of the data items. Step (R) dominates wall-clock as a result of disk seeks. Step (D) consumes more than 75% of CPU time, suggesting decoding as the bottleneck once I/O is improved.

Factoring both wall-clock time and CPU time into consideration, the results in Figure 2 suggest that big wins may be achievable by 1) reducing disk seeks and 2) integrating I/O with decoding. This, in turn, strongly suggests that acceleration is needed early in the processing pipeline where the greatest wins are available. It also requires cross-layer optimizations that span application (e.g., JPEG decoding), file system (e.g., block-to-file mapping), device driver (e.g., detection and parallelization of accelerators), and hardware (e.g., disk read head positions). To enable such complex and technology-dependent optimizations while preserving application simplicity and maintainability, we present and evaluate a new storage API in the rest of the article. We refer to any storage system that supports this API as *decode-enabled storage*.

DECODE-ENABLED STORAGE

Application Programming Interface

Our new storage application programming interface (API) extends the well-known object store API,[19] which operates on entire variable-size logical objects rather than blocks or sectors. Our extension consists of two new calls that are described in the rest of this section. The first operates on a single object, while the second iterates the first call over multiple objects. In these calls, each object is addressed by an integer `object_id`.

```
FetchAndDecodeObject(
        int64 object_id,
        int32 opcode, void *params,
        iovec *where_to_put_decoded_object,
        iovec *where_to_put_original_object)
```

The above uses the `opcode` field to indicate whether to obtain the on-storage encoded object, the decoded version, or both. The last is useful in a server context, because it avoids reencoding for network transmission after early-discard filtering on the server. The scatter–gather I/O vectors, `where_to_put_decoded_object` and `where_to_put_original_object` indicate the memory locations where the decoded and original objects are to be placed.

While decode acceleration is our primary motivation, we leave open the possibility of opcode indicating use of other forms of storage intelligence such as cropping around human faces. Input or output parameters for such operations can be provided through the params pointer. Since partial read or write of an object is not supported, very large videos (gigabytes per second to terabytes per second) are stored as a sequence of objects (one per short video segment), plus an additional directory object to store the mapping from entire video to short segments.

```
IterateCollection(
        int64 collection_id,
        int32 opcode,   void *params,
        int64 logical_index,  int64 *flags,
        int64 *returned_object_id,
        iovec *where_to_put_decoded_object,
         iovec *where_to_put_original_object)
```

The above iterates over the collection of objects specified by the first parameter. The storage system is free to deliver these objects in any order, thus enabling it to exploit internal optimizations such as disk seek minimization and caching. Each call retrieves one object, with returned_object_id identifying which object was fetched. The call mandates exactly-once semantics: exhaustive and nonrepeating access to all the objects listed in collection_id. The iterator cursor is logical_index. The flags parameter can be set to the value NEW_ITERATION to abort the current iteration (if any), and start a new iteration. It indicates COLLECTION_LAST when the last object is returned. The other parameters play the same roles as in FetchAndDecodeObject.

Implementation Considerations

Since visual data formats such as JPEG, PNG, and MP4 define long-lived standards, fixed-function ASICs suffice as hardware accelerators for decode-enabled storage. Their lack of programmability is not a handicap, and their superior energy efficiency is an advantage. ASICs for visual data decoding have been studied extensively.[20,21]

A well-known design principle for scalability from "Big Data" systems and database systems is to minimize data copying. Locating the decode–acceleration ASIC in a storage device aligns well with this principle. No new data copies are needed; rather, data are decoded in a streaming operation as it is read off the disk surface. This may seem to be a counterintuitive optimization at first glance, because early decoding greatly increases the bandwidth demand on the SATA interconnect from disk. However, this can be overcome by replacing SATA by the modern NVMe host-storage interconnect. NVMe was originally created to support the much higher bandwidth demand of solid state drives (SSDs). There is industry speculation that NVMe will become the unified interface for all storage types in the near future, including SSD and disks. By fortunate coincidence, this trend aligns well with our proposal to place decoding functionality on disks.

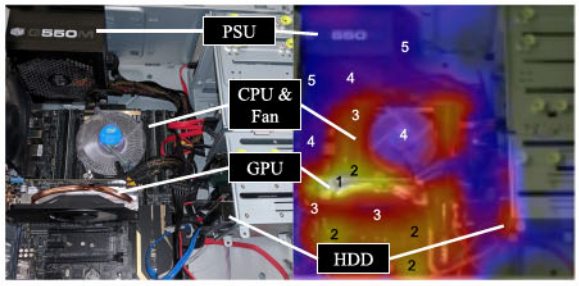

FIGURE 4. Thermal heatmap of a typical cloudlet. Numbers in heatmap indicate temperature (1 = hottest, 5 = coolest).

The ASIC could be host-attached like graphics processing units (GPUs) today [left side of Figure 3(a)]. However, that strategy suffers from at least two deficiencies. First, it requires some host mediation of the decoding process, incurring context-switching overheads. Second, memory bandwidth demand is high because encoded objects have to be placed in memory by the disk, then read by the ASIC, and finally the decoded objects have to be written to memory. Only the last of these costs is paid if the ASIC is colocated with storage [right side of Figure 3(a)].

ASIC placement is also guided by thermal considerations, as shown by Figure 4. Areas in white and yellow represent the hottest parts of a typical cloudlet, while it is processing a visual data pipeline. Adding the ASIC to these areas would make cooling more difficult. As the coolest parts of the system (red areas of Figure 4), storage devices can most easily accommodate the thermal load of additional processing.

EVALUATION

How much system-level performance improvement does decode-enabled storage offer to a Silent Witness? A rigorous answer would require a hardware implementation of this new abstraction. To gain confidence in the likely win before making this investment, we have implemented a timing-accurate software emulation of the new abstraction on top of existing storage hardware. Since our emulation is conservative, it likely understates the win achievable in a real deployment.

Timing-Accurate Emulation of Decode-Enabled Storage

We implement timing-accurate emulation of an NVMe-attached decode-enabled disk that allows execution of real application code and measurement of wall-clock time and real operating system (OS)-level statistics (e.g., CPU time, bandwidth). The emulation allows us to experiment with a wide range of parameters that are otherwise unavailable in existing hardware products. Event-driven simulation based on precomputation and modeling is used to emulate hardware decoders, while disk timing is based on a real disk.

HOW MUCH SYSTEM-LEVEL PERFORMANCE IMPROVEMENT DOES DECODE-ENABLED STORAGE OFFER TO A SILENT WITNESS? A RIGOROUS ANSWER WOULD REQUIRE A HARDWARE IMPLEMENTATION OF THIS NEW ABSTRACTION.

Figure 3(b) depicts our emulator. Application programs are largely unchanged except for the use of the new application programming interfaces (APIs) to fetch (decoded) data. The emulator includes critical components of a decode-enabled storage device. This includes the host-disk interconnect bus, potentially parallel hardware accelerators for decoding, and mechanical disk timings. The controller implements the logic to control data flow and coordinate different components in order to service a request from applications.

For each software-emulated component, we construct 1) a model to calculate the (content-dependent) completion time for operations, and 2) a mechanism to generate the actual results produced (e.g., the actual decoded pixel arrays). The latter mechanism must execute faster than the modeled time; this is achieved by serving precomputed results from memory. As this generally runs faster than needed, the system inserts a high-resolution sleep to achieve the modeled latency.

We use a discrete event-driven approach to model complex interactions between components. For example, requests to the decode ASICs are serialized through a priority queue sorted by the simulation timestamp. Thus, requests are ordered "correctly" even if generated out of order by the concurrently executing components. The simulation clock is continually updated to match the real world time. When a request's computed completion time is reached by the simulation, we send the response back to the application.

Prototypes ASIC- and field-programmable gate array (FPGA)-based image decoders[20,21] are typically characterized by a metric specified in MegaPixels per second (MPixel/s). Hence, we parameterize our emulated decoder with a targeted MPixel/s. With RGB format, 1 MPixel/s is equivalent to 3 MByte/s of output. In practice, decoding time may vary based on content and compression level of images. Our emulator accounts for such variability, while maintaining a target speed. We compute a global scaling factor that scales the average software decode time for an entire image dataset to the target rate; we apply this factor to the software decode time of each image to obtain its simulated HW decode time.

More concretely, suppose the dataset has N images, the software decode time of image i is t_i^{sw}, and its decoded size is r_i MPixel. When simulating an image decoder parameterized at M^{sim} MPixel/s, the simulated decode time of image i is calculated as

$$t_i^{\text{sim}} = \frac{\sum_{k=0}^{N} r_k}{\sum_{k=0}^{N} t_k^{sw}} \cdot \frac{t_i^{sw}}{M^{\text{sim}}}.$$

To emulate real-time hardware decode, we predecode all images and store them in a ramdisk. At run time, the decoded data are rapidly returned to the application. We emulate video decoding and face detection hardware in a similar fashion. For simplicity, we parameterize them using a target FPS, and scale individual elapsed times based on profiled software times.

We emulate mechanical disk timing factors, such as seeks, platter rotations, and block cache by reading files from a real magnetic hard disk (HDD). Although this will include overheads of the OS, filesystem, and bus, we find that these are small by performing similar tests on a fast SSD. We were careful to clear the OS page cache before each experiment.

Experimental Setup

We ran experiments on a cloudlet with two Intel Xeon processors (totaling 36 cores@2.3 GHz), 128-GB DRAM, and an NVIDIA GTX 1080 Ti GPU. Decode-enabled storage was provided by the emulator described above (see the "Timing-Accurate Emulation of Decode-Enabled Storage" section). Emulated disk to host connectivity was NVMe (16 Gbps).

We used two datasets in our experiments. The first was the same dataset that was used in Figure 2 (random sampling of 50,000 images from the YFCC100 M dataset). The corresponding decoded data totals 51 GB, fitting comfortably in the memory of emulated storage. The second dataset consisted of six videos

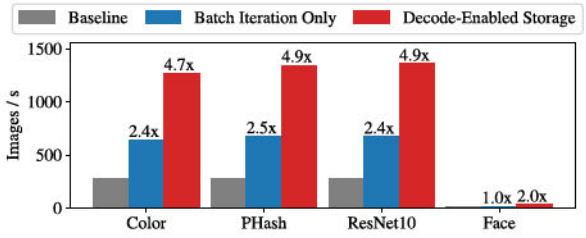

FIGURE 5. Effectiveness of individual optimizations.

from the VIRAT Release 2.0 Ground dataset,[22] which are encoded in H.264 format at 1080p/720p @ 30 FPS. Based on NVMe bandwidth, the emulation was configured with five JPEG decoders at 140 Mbps each.

Results

Full details of our evaluation, including microbenchmarks and application-level pipelines, can be found in a recent technical report.[23] Due to limited space, we only present a summary of results here. We first evaluate a series of microbenchmarks of early-discard filters of partial analytics pipelines: 1) `Color` finds images with many red pixels; 2) `PHash` calculates an image's perceptual hash value; 3) `ResNet10` is a tiny DNN with 65×65 input and 10 layers; and, 4) `Face` detects faces in an image, annotates the bounding boxes, and drops images with no faces.

Figure 5 reports the benchmark throughputs (image/s) for three systems: 1) Baseline: uses standard SATA-connected disk and software decode; 2) Batch Iteration Only: optimizes multiobject batch read order on a standard disk; 3) Decode-enabled Storage: combines batch iteration, on-disk decode, and NVMe. The data labels show improvement factors relative to baseline. We see that batch iteration alone is effective (up to 2.5× improvement) by improving I/O efficiency, but is not responsible for all performance gains. Adding on-disk decode HW and NVMe achieves up to 4.9× improvement over baseline. `Face` is much slower than the others, because face detection is computationally expensive. However, even a single face detection chip at 30 FPS delivers 2× gain over software detection on the four CPU cores used in this experiment.

Next, we consider four full analytics pipelines that are representative of Silent Witness workloads:

- *RedBus* finds red buses in the YFCC100 M dataset. It first runs a redness color filter, and then an SSD-MobileNet DNN running on a GPU to detect buses.
- *RedBus-fast* trades off accuracy for speed by replacing object detection with image classification using a MobileNet DNN. Classification is faster, but may miss images where the bus is not the dominant object.
- *Obama* searches for Barack Obama in YFCC100 M. It first runs face detection to discard images without faces, and then runs face recognition on the face patches.
- *Pedestrian* detects humans in videos from the VIRAT dataset. It performs frame sampling and uses image difference to filter sampled frames. It then runs the candidate frames through a Faster R-CNN DNN to detect humans. We evaluate this pipeline with two frame-sampling rates: 10% and 50%.

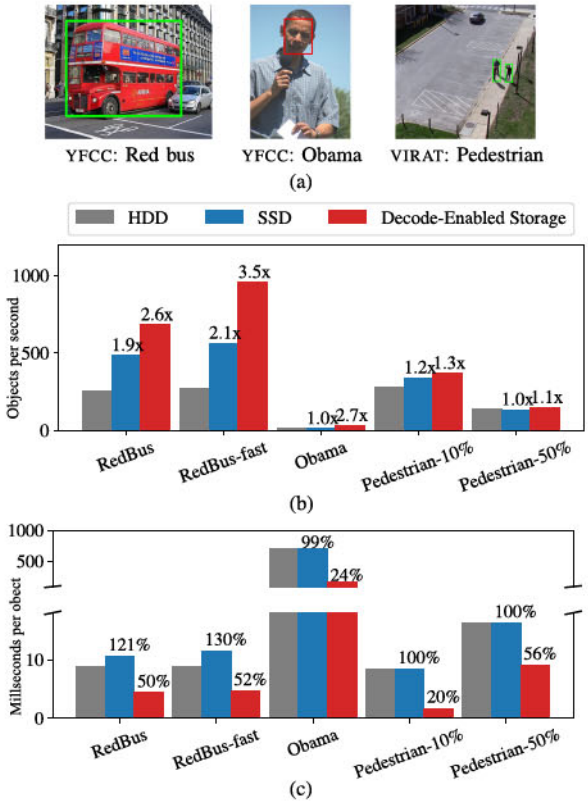

FIGURE 6. Full end-to-end visual pipeline performance. (a) Example results. (b) Pipeline throughput. (c) CPU time.

Figure 6(a) shows an example search result from each pipeline. The selectivity of the pipelines (i.e., fraction of objects that contain the search target) is as follows: 0.01% for *RedBus* and *RedBus-fast*, 0.004% for *Obama*, and 2.45% for *Pedestrian*.

Our results report two different metrics: pipeline throughput [see Figure 6(b)] and CPU cost per object [see Figure 6(c)]. We consider three configurations: 1) [gray] a standard SATA magnetic disk (HDD); 2) [blue] a standard SATA flash drive (SSD); and 3) [red]

emulated decode-enabled storage with NVMe interconnect). The numeric annotations on blue and red bars are relative to the corresponding gray bar.

These results show significant wins for decode-enabled storage. In all cases, throughput is increased. The CPU cost per object is lowered, enabling greater parallelism and hence better scaling for concurrent workloads. The win can be as high as 3.5× relative to HDD, and 2.7× relative to SSD. These are big wins for a storage-centric workload. For two reasons, we expect that HDDs rather than SSDs will be the storage technology of choice for Silent Witnesses. First, their higher capacity enables a longer retention period for data. Second, their lower cost-per-bit makes Silent Witnesses more affordable for widespread deployment.

CONCLUSION

Preprocessing visual data to create an index is the standard way to gain orders-of-magnitude speedup for interactive search. When preprocessing is impossible, as in the case of a Silent Witness, there is no option but to use discard-based search. This is an inherently slower process because the user is actually conducting two interleaved searches concurrently: one to converge on the right query to pose, and the other to examine results from the current query. In this article, we have shown throughput speedups up to 3.5× for a typical processing pipeline in such a system. That is a considerable increase in productivity for a law enforcement official who is trying to solve a crime. Instead of spending an entire workday on a single case, she only needs to devote a few hours. This speedup from decode-enabled storage is crucial to making the Silent Witness abstraction a practical tool.

THE SILENT WITNESS ABSTRACTION ENABLES THE VALUABLE CRIME-FIGHTING TOOL OF VIDEO SURVEILLANCE TO BE USED WITHOUT COMPROMISING PRIVACY.

Much-hyped "AI at the Edge" will ultimately be judged by how it improves human lives. The Silent Witness abstraction enables the valuable crime-fighting tool of video surveillance to be used without compromising privacy. The primary beneficiaries will be the inhabitants of high-crime areas, who are often the most economically challenged segments of society.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) under Grant CNS-2106862. Additional support was provided by ARM, AutoDesk, CableLabs, Crown Castle, Deutsche Telekom, Intel, InterDigital, Meta, Microsoft, Seagate, VMware, Vodafone, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

REFERENCES

1. R. G. Jones, "The crime rate drops, and a city credits its embrace of surveillance technology," *New York Times*, May 2007. [Online]. Available: https://www.nytimes.com/2007/05/29/nyregion/29east.html
2. J. B. Kuhns, K. R. Blevins, S. Z. Lee, A. Sawyers, and B. Miller, "Understanding decisions to burglarize from the offender's perspective," Dept. Criminal Justice Criminol., Univ. North Carolina at Charlotte, Charlotte, NC, USA, Tech. Rep., Dec. 2012, doi: 10.13140/2.1.2664.4168.
3. J. Xiao, S. Li, and Q. Xu, "Video-based evidence analysis and extraction in digital forensic investigation," *IEEE Access*, vol. 7, pp. 55432–55442, 2019.
4. N. G. L. Vigne, S. S. Lowry, J. A. Markman, and A. M. Dwyer, "Evaluating the use of public surveillance cameras for crime control and prevention," Urban Inst., Justice Policy Center, Tech. Rep., 2011. [Online]. Available: https://www.urban.org/research/publication/evaluating-use-public-surveillance-cameras-crime-control-and-prevention
5. T. Soper, "Security cameras helped catch boston marathon bombers; what's your take on public surveillance?," Apr. 2013. [Online]. Available: https://www.geekwire.com/2013/security-cameras-helped-catch-boston-marathon-bombers-public-surveillance/
6. M. McIntyre, "Video evidence and vital nonhumans," *J. Multimodal Rhetorics*, vol. 2, no. 2, Fall 2018, Art. no. 87.
7. C. Nikisch, "Boston police use marathon security to expand video surveillance network," Apr. 2015. [Online]. Available: https://www.wbur.org/news/2015/04/17/boston-marathon-surveillance
8. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
9. M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.
10. M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68–75, Feb. 2006.

11. K. Muhammad, R. Hamza, J. Ahmad, J. Lloret, H. Wang, and S. W. Baik, "Secure surveillance framework for IoT systems using probabilistic image encryption," *IEEE Trans. Ind. Inform.*, vol. 14, no. 8, pp. 3679–3689, Aug. 2018.
12. Z. Shao, J. Cai, and Z. Wang, "Smart monitoring cameras driven intelligent processing to big surveillance video data," *IEEE Trans. Big Data*, vol. 4, no. 1, pp. 105–116, Mar. 2018.
13. L. Huston *et al.*, "Diamond: A storage architecture for early discard in interactive search," in *Proc. USENIX Conf. File Storage Technol.*, 2004, pp. 73–86.
14. M. Satyanarayanan *et al.*, "Searching complex data without an index," *Int. J. Next- Gener. Comput.*, vol. 1, no. 2, 2010.
15. M. Satyanarayanan, R. Sukthankar, L. Mummert, A. Goode, J. Harkes, and S. Schlosser, "The unique strengths and storage access characteristics of discard-based search," *J. Internet Serv. Appl.*, vol. 1, pp. 31–44, 2010.
16. M. Satyanarayanan, P. Gibbons, L. Mummert, P. Pillai, P. Simoens, and R. Sukthankar, "Cloudlet-based just-in-time indexing of IoT video," in *Proc. IEEE Glob. IoT Summit*, Geneva, Switzerland, 2017, pp. 1–8.
17. Z. Feng, S. George, J. Harkes, P. Pillai, R. Klatzky, and M. Satyanarayanan, "Edge-based discovery of training data for machine learning," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 145–158.
18. B. Thomee *et al.*, "YFCC100 M: The new data in multimedia research," *Commun. ACM*, vol. 59, no. 2, pp. 64–73, 2016.
19. M. Mesnier, G. R. Ganger, and E. Riedel, "Object-based storage," *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 84–90, Aug. 2003.
20. J. Ahmad, K. Raza, M. Ebrahim, and U. Talha, "FPGA based implementation of baseline JPEG decoder," in *Proc. 7th Int. Conf. Frontier Inf. Technol.*, 2009, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1838002.1838035
21. G. Kyrtsakas and R. Muscedere, "An FPGA implementation of a custom JPEG image decoder SoC module," in *Proc. IEEE 30th Can. Conf. Elect. Comput. Eng.*, 2017, pp. 1–4.
22. S. Oh *et al.*, "A large-scale benchmark dataset for event recognition in surveillance video," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 3153–3160.
23. Z. Feng *et al.*, "Improving edge elasticity via decode offload," Schl. Comput. Sci., Carnegie Mellon Univ., Tech. Rep. CMU-CS-21-139, Sep. 2021. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/2021/CMU-CS-21-139.pdf

MAHADEV SATYANARAYANAN is the Carnegie Group University Professor of Computer Science at Carnegie Mellon University (CMU), Pittsburgh, PA, 15213, USA. His multidecade research career has focused on the challenges of performance, scalability, availability, and trust in information systems that reach from the cloud to the mobile edge of the Internet. Satyanarayan received a Ph.D. degree in computer science from CMU. He is a Fellow of ACM and IEEE. Contact him at satya@cs.cmu.edu.

ZIQIANG FENG is with Google, Menlo Park, CA, 94025, USA. His research interests include machine learning, edge computing, and distributed systems. Feng received a Ph.D. degree in computer science from Carnegie Mellon University. Contact him at csfzq2009@gmail.com.

SHILPA GEORGE is currently working toward a Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA, 15213, USA. Her research is at the intersection of edge computing, distributed systems, machine learning and computer vision. Contact her at shilpag@cs.cmu.edu.

JAN HARKES is a principal project scientist in the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 15213, USA. His research interests include distributed systems, storage, security, Internet of Things, and edge computing. Harkes received an M.Sc. degree in computer science from the Vrije Universiteit, Amsterdam. Contact him at jaharkes@cs.cmu.edu.

ROGER IYENGAR is currently working toward a Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA, 15213, USA. His research interests include mobile systems and edge computing. Contact him at raiyenga@cs.cmu.edu.

HAITHEM TURKI is currently working toward a Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA, 15213, USA. His current research interests lie at the intersection of edge computing and machine learning. Contact him at hturki@cs.cmu.edu.

PADMANABHAN PILLAI is currently a senior research scientist with Intel Labs, Santa Clara, CA, 95054, USA. His research interests include the use of edge computing to support immersive interactive experiences, and perform real-time understanding of the world. Pillai received a Ph.D. degree in computer science from the University of Michigan. He is a Member of IEEE. Contact him at padmanabhan.s.pillai@intel.com.