VideoPipe: Building Video Stream Processing Pipelines at the Edge

Mohammad Salehe*
University of Toronto
Toronto, Canada
salehe@cs.toronto.edu

Zhiming Hu
Samsung AI Center
Toronto, Canada
zhiming.hu@samsung.com

Seyed Hossein Mortazavi*
University of Toronto
Toronto, Canada
mortazavi@cs.toronto.edu

Tim Capes
Samsung AI Center
Toronto, Canada
t.capes@samsung.com

Iqbal Mohomed
Samsung AI Center
Toronto, Canada
i.mohomed@samsung.com

Abstract

Real-time video processing in the home, with the benefits of low latency and strong privacy guarantees, enables virtual reality (VR) applications, augmented reality (AR) applications and other next-gen interactive applications. However, processing video feeds with computationally expensive machine learning algorithms may be impractical on a single device due to resource limitations. Fortunately, there are ubiquitous underutilized heterogeneous edge devices in the home. In this paper, we propose VideoPipe, a system that bridges the gap and runs flexible video processing pipelines on *multiple devices*. Towards this end, with inspirations from Function-as-a-Service (FaaS) architecture, we have unified the runtime environments of the edge devices. We do this by introducing *modules*, which are the basic units of a video processing pipeline and can be executed on any device. With the uniform design of input and output interfaces, we can easily connect any of the edge devices to form a video processing pipeline. Moreover, as some devices support containers, we further design and implement *stateless services* for more computationally expensive tasks such as object detection, pose detection and image classification. As they are stateless, they can be shared across pipelines and can be scaled easily if necessary. To evaluate the performance of our system, we design and implement a fitness application on three devices connected through Wi-Fi. We also implement a gesture-based Internet of Things (IoT) control application. Experimental results show the the promises of VideoPipe for efficient video analytics on the edge.

CCS Concepts • **Computing methodologies** → *Computer vision tasks*; • **Computer systems organization** → *Cloud computing*;

Keywords edge computing, video streaming, pipelining

ACM Reference Format:
Mohammad Salehe, Zhiming Hu, Seyed Hossein Mortazavi, Tim Capes, and Iqbal Mohomed. 2019. VideoPipe: Building Video Stream Processing Pipelines at the Edge. In *20th International Middleware Conference Industrial Track (Middleware Industry '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3366626.3368131

*Work done during internship at Samsung AI Center, Toronto.

Middleware Industry '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7041-7/19/12…$15.00
https://doi.org/10.1145/3366626.3368131

1 Introduction

Many user-oriented computer vision and AI applications require real-time live video processing with immediate data processing in order to deliver a better user experience. The home environment is where many of these applications are used. Examples include virtual reality (VR) and augmented reality (AR) [18, 29], activity detection for smart home [21, 31] and health applications [20, 30]. Such applications have also emerged in retail environments such as "cashierless" stores where users can checkout items by simply walking out with them and have a computer vision system detect and process the purchase [1], and AR smart mirrors that let users try on clothes, accessories or make-up.

While recent advancements in machine learning and computer vision have enabled these applications through the developments of deep neural networks, the computational requirements even for inference can be substantial. These models cannot run on smaller devices with limited battery and processing power. Significant research efforts have attempted to shrink models to mobile and embedded devices using techniques such as quantization, model compression, etc. But these approaches reduce accuracy and limit possible applications. Other approaches attempt to run the heaviest parts of the processing pipeline in the cloud [11, 32, 33] as resources may be provisioned in an on-demand fashion. However, real-time video processing in the cloud may be impractical because of latency requirements for interactive applications, bandwidth limitations and privacy restrictions. Recently, edge computing has emerged as a viable solution to the first and partially the second challenge, but doesn't concern privacy issues as data is still transmitted to other entities. Considering these challenges, the question is how to architect a system that utilizes available resources inside a house (or even on the edge) where heterogeneous devices exist.

Previous work has attempted to apply principles from service-oriented architectures, where computationally expensive AI computation is embodied in services (perhaps hosted on a powerful edge server), and the application resides on some other host, making calls to the services. Our experience has shown that this approach incurs significant overhead in terms of delays in data transfer between the caller and the service. There have even been notable efforts to build applications at the edge by deploying devices capable of running Function-as-a-Service platforms [3] and containers [4].

But in a home environment, a user may have smartphones and tablets running Android, TVs, watches and Smart Fridges that run a Linux based OS such as Tizen [9], and general purpose devices such as laptops and desktop computers. Some of these devices are constrained in that they cannot run container-based applications but can support a high-level language such as JavaScript that is particularly well-suited to being sandboxed within a virtual execution environment. Others devices in the home, such as laptops and desktop computers, are less constrained and can run container[1] based applications.

In this paper, we propose VideoPipe, a video processing framework for the home that sets up video processing pipelines on *multiple heterogeneous devices*. More specifically, inspired by service-oriented programming, the microservices architecture and the Function-as-a-Service architecture, VideoPipe runs user applications that are splitted into modules by connecting different devices to form a video processing pipeline where each device may only execute a part of the pipeline. VideoPipe is particularly useful when there is heterogeneity among devices and containers cannot be run on all devices; devices without containers can still contribute to the pipeline, by exposing native services and local I/O capabilities to modules, and the runtime automatically manages chaining the various modules together - on and off device. Running on heterogeneous devices allows applications to run in the home, on the edge or in a hybrid model.

To this end, we design and implement the same runtime environments and input/output interfaces in VideoPipe for heterogeneous devices in the home. With this feature, any processing units (*modules* in this paper) in the video processing pipeline can be executed on any device if it has enough resources and any of the two devices can be easily connected. This provides much more flexibility to build the video processing pipeline across multiple devices. In addition to *modules*, we also incorporate *stateless services* for computationally expensive tasks such as object detection, pose detection and image classification. As they are stateless, the *services* provide auto-scaling and reusability capabilities across different pipelines. In short, in VideoPipe, the video frames will flow through the *modules* on the devices that are involved in the video processing pipeline and *modules* may call the stateless *services* for heavy-weight processing.

We make three key contributions in this paper. (i) We describe the design and implementation of VideoPipe, a FaaS-Container Hybrid runtime platform that co-locates modules with the services they call in order to reduce round-trip delays. VideoPipe leverages the capabilities of both constrained and resourceful devices. Through our evaluations, we show the clear benefits of co-locating modules with the services they call. (ii) We have provided a uniform runtime with I/O interfaces on multiple devices to enable the chaining of processes on edge devices that create the pipeline. (iii) We describe various proof-of-concept applications that we implemented on the VideoPipe platform and we evaluated the performance of VideoPipe on real edge devices.

2 System Design

Inspired by functional programming [12, 17] and the microservices architecture [22, 28], VideoPipe can support video processing pipelines composed of smaller vision modules.

[1] e.g. Docker

In our design, each application is specified as a Directed Acyclic Graph (DAG) by the application developer. Each node in this graph is a module that defines the data flow logic. The edges are the data flow pipeline between the nodes that the system sets up. The modules can call corresponding machine learning services such as object detection, pose detection and image classification. All the communications among modules and services are supported by ZeroMQ for efficient data communications.

An example pipeline is shown in Fig. 1. In this figure, we can see that our video processing pipeline is distributed over three edge devices and the modules are connected with remote calls. The internal details of communications are masked from the pipeline developer. Each module can call the services available on the same device for computationally expensive machine learning services.

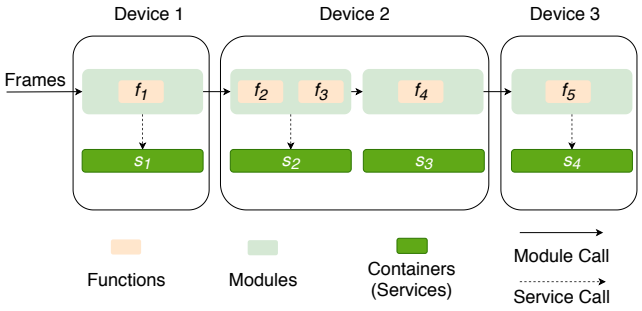

Figure 1. An example of video processing pipeline.

2.1 Modules

In our system, we use *modules*, which are self-contained units with encapsulated states, to control the flow of video frames inside the video processing pipeline. One of the core principles behind our design is simplicity. Each module implements an interface that is triggered by data arrival events. Depending on the device that they are being executed on, modules can use *stateless services* to do the heavy lifting on video processing and then forward the results to other modules. The APIs provided by these stateless services, greatly simplify the development of modules, which only care about the high-level logic workflow of each application.

To ease the deployment of the modules, we also provide the same runtime environments even though the edge devices are heterogeneous in both hardware specifications and software stacks. With the same runtime environment, we can deploy the modules on any devices, which provides much more flexibility to the pipelines.

2.2 Services

The main video analytics are performed by stateless services accessible to modules. These services perform framewise video processing. Examples of these services include object detection, face detection, activity recognition, and object tracking. These services all receive needed data as input so they do not require saving state. This allows the services to be shared among different applications and also allows for horizontal scaling of these services, which results in faster processing of live streams and higher frame rates.

Modules can be deployed at almost all the edge devices as they only contain lightweight application code. However, we can only

Table 1. JavaScript interface to module code

Function	Description
init()	Callback for module initialization
event_received(message)	Callback for event arrival
call_service(service, message)	Requests container-based services
call_module(module, message)	Calls the next module

deploy the services on the devices that support containers as services will be running inside containers. Therefore, services are preinstalled on some edge devices and modules will be spawned on edge devices after we set up the pipeline.

2.3 Data Flow

Our goal is to support high frame rates and to avoid delays perceivable by the user. In the simple pipeline as shown in Fig. 1, the video source may push images at a high frame rate into the pipeline. Queuing the images anywhere inside the pipeline will introduce delays which are undesired in real-time applications and dropping frames inside the pipeline wastes computation resources if there is a bottleneck. We do not use any queues in our design. When the final module is done with its current data, it signals the source to send a new frame into the pipeline. This approach pushes frame dropping to the beginning of the pipeline and eliminates queuing delays inside the pipeline. A more intelligent signaling mechanism may also be utilized (e.g., by identifying the bottlenecks) to improve efficiency further.

3 Implementation

In VideoPipe, we implement an execution framework that supports running lightweight modules on different heterogeneous devices such as smart phones, TVs, etc. We show the architecture for VideoPipe in Fig. 2. As shown in the figure, we use Duktape [6], a lightweight embeddable JavaScript engine that executes JavaScript code on different underlying environments. Similar to other Function-as-a-Service platforms, modules in VideoPipe are triggered on events. These events are either data arrival events or calls from other modules. For each module of an application, a separate Duktape context is created to execute the module code that is written in JavaScript. Separate Duktape contexts are spawned inside a single Jave Virtual Machine (JVM) environment to provide isolation without compromising performance.

The API specifications for the interface to each module is shown in Table 1. The *init* function is called upon deployment on the device and *event_received* is called each time there is new message for the module. *call_service* is used by the module code to call on services on the data and *call_module* is to call other modules. To minimize data copying between different components, rather than copying the full image frames to the module, we pass on a reference id that identifies the frame. The module code can use that id to do the modifications on the image using the services and forward the frames to other modules.

```
// An Example of DAG Configuration for a Pipeline
modules : [
```

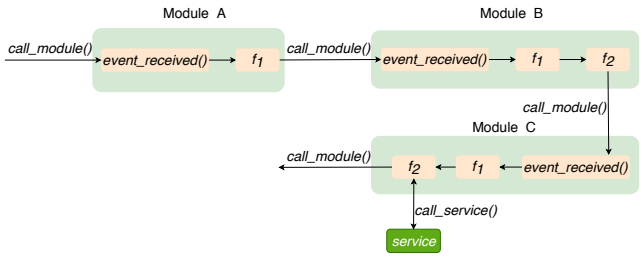

Figure 2. Implementation details of JavaScript modules. Each module is implemented in JavaScript and can have multiple functions and its own state. Each module is running on a separate JavaScript context.

```
{   name: pose_detector_module
    include ("./PoseDetectorModule.js")
    service: ['pose_detector']
    endpoint: ["bind#tcp://*:5861"]
    next_module: activity_detector_module }
{   name: activity_detector_module
    include ("./ActivityDetectorModule.js")
    service: ['activity_detector']
    endpoint: ["bind#tcp://*:5862"]
    next_module: [rep_counter_module,
         display_module] }
{   name: rep_counter_module
    include ("./RepCounterModule.js")
    service: ['rep_counter']
    endpoint: ["bind#tcp://*:5863"]
    next_module: display_module }
...
]
```

Listing 1. Sample pipeline configuration file. Some details elided to simplify presentation. Each service is embodied within a container spec. These can be references to running containers or Dockerfiles in our implementation.

3.1 Pipeline Configuration

The list of services that the application can use is predefined and the application developer specifies the list of services it needs to use for each module through a configuration file. The user also determines how modules call each other. These settings are necessary for application deployment so the system can setup the data pipeline. VideoPipe prepares the required service stubs on each device and connects different components together.

An example of the pipeline configuration file containing three modules is included in Listing 1. The code file of the module along with the services it uses are specified in the two lines after the name of the module. The endpoint field indicates how this module can be reached. The next_module field determines the outgoing edges to other modules. VideoPipe's simple interface enables application developers to specify the application logic only by implementing functions in the modules code and describing connections between modules to create the video processing pipeline.

3.2 Message Transfer Protocol

Our pipelining system sets up the data path between modules upon deployment using the information about the DAG of the application provided by the developer. This pipeline is established using ZeroMQ [5] which is a high-performance asynchronous messaging library. This minimizes the delay between different application modules and improves flexibility of the deployment. Images that are passed between devices are encoded/decoded and transferred using ZeroMQ. When a module calls *call_module*, our system automatically forwards the data on the appropriate ZeroMQ socket for data transfer, and on the receiving end, data is passed by ZeroMQ to the corresponding module. While publish subscribe systems such as Kafka [24] or queue based system RabbitMQ have brokers in their systems, these brokers will incur extra data communication overheads because the data was first sent to the broker and then forwarded to the final destination.

4 Applications

In this section, we will describe two applications that we have built on top of VideoPipe and some other potential applications that we can support.

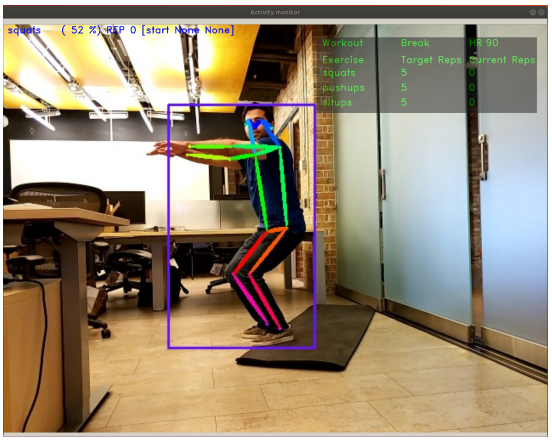

Figure 3. Output of the fitness application that is displayed on a 4K Television.

4.1 Fitness Application

The first application is a workout guidance system that tracks the progress of users' fitness routine as they perform various exercises in their living room. In this application, the user places their smartphone on a phone cradle mounted on the TV. The application involves running numerous computationally expensive algorithms on the camera feed and renders the output on the living room TV display. We show frames with rich information including the user skeleton and the number of exercise reps for each exercise. A screenshot of our fitness application is shown in Fig. 3.

For this system we have designed and applied pose detection, activity recognition and a rep counter. The overview of our system is shown in Fig. 4. As computational resources on the phone are not adequate for pose detection, we move this computation to a desktop. The devices in Fig. 4 are connected by our system to form a video processing pipeline.

As in Fig. 4, the top row shows the modules and the bottom row lists the services where blue boxes contain native services and green boxes represent remote services inside containers. We utilize ZeroMQ to interact with the remote services inside the containers. In the following sections, we will go through more details about the services.

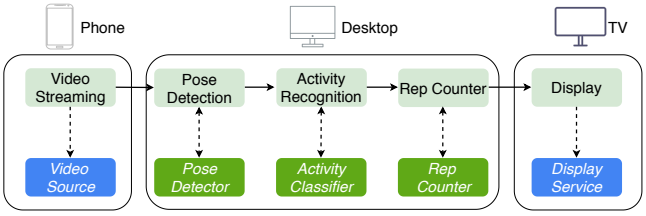

Figure 4. The overview of our application.

4.1.1 Pose Detection

The 2D pose detector first detects a human and places a bounding box around them. Within that bounding box, it detects 17 keypoints.

4.1.2 Activity Recognition

Our activity recognition system utilizes nearest neighbor on pose sequences. To feed nearest neighbors, we take a list of 15 consecutive frames that are provided by the calling module. We normalize the coordinates framewise so that (0,0) is located at the average of the left and right hips of the human in that frame. The algorithm is trained on all available labelled data except for a withheld test set. The test accuracy on a withheld test set was above 90%. This is higher than generally reported in the literature because our system has a standardized viewing distance and standardized viewing angle in our application.

4.1.3 Rep Counter

Our rep counting system relies on the fact that all exercises start and return to an initial position to start and end a rep. As such, we feed frame by frame from the same 2D pose detector as in the activity recognition system. We use k-means with $k = 2$ to classify the frames into a cluster that occurs near the start of the exercise and a cluster that occurs near the end of an exercise. To avoid issues with boundary cases, we require 4 frames to have transitioned to count a state transition, which avoids counting alternating 0's and 1's that sometimes occur near the cluster boundary. We count a state transition from and back to the initial state as a single rep. On our withheld test set, 83.3% accuracy is achieved.

4.2 Gesture Control for IoT Applications

The second application pipeline that we built on top of VideoPipe is a video based gesture control system for IoT applications. With the same pose detector service, we use a similar activity classifier to support activities, such as 'waving' and 'clapping'. The activity classifier can be trained with custom actions that trigger custom behaviours. Two examples are using 'clapping' to toggle the light in the living room and using 'waving' to toggle a doorbell camera. In this application, we have the video streaming module, pose detection module and activity recognition module and use the pose detector and activity classifier services.

4.3 Other Applications

In addition to the above two applications, we also implement a fall detection application pipeline with VideoPipe. Real-time video analytics consisting of hand detection/tracking, face detection/tracking and pose detection/tracking, can create ample opportunities for new user interfaces with IoT devices and new multiple device experiences in the home environment.

5 Evaluation

In this section, we demonstrate the benefits of VideoPipe's design. In particular, we compare VideoPipe's performance with a baseline approach where frames are sent to a server for processing. We also highlight the advantage of re-using services for multiple pipelines.

5.1 Experimental Setup

We evaluate two pipelines. One is the fitness application pipeline as shown in Fig. 4 and the other is for the gesture control introduced in Sec. 4.2. The phone is one of the flagship Android phones in 2018 with 6GB of main memory and 128 GB of storage. Devices on the system are connected to each other over a Wi-Fi network.

Metrics We evaluate our system based on the frame rate of the pipelines and latency of different modules in the pipeline.

Baseline We compare our approach with a baseline architecture inspired by EdgeEye [25], where all the application logic (modules) is in one device and the modules may call the services in remote servers through API calls, which is shown in Fig. 5. While in our case, our modules are deployed in a way that they are co-located with the corresponding services available on the devices, as shown in Fig. 4.

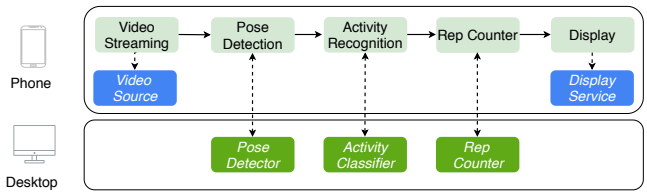

Figure 5. The architecture of the baseline approach where three modules make API calls to a remote server.

5.2 Experimental Results

We aim to answer the following questions: (I) What is the performance of VideoPipe comparing with the baseline? (II) How the performance will change if multiple pipelines are sharing the services?

5.2.1 Frame Rates of Pipelines

We first depict the latency of different modules in the fitness application in Fig. 6. In this figure, we can see that VideoPipe always has the lowest latency compared with the baseline. Specifically, VideoPipe has less latency for pose detection, which contributes most of the improvements.

We also record the end-to-end frame rate shown in Table. 2. In the second and third column, both VideoPipe and the baseline are running the fitness pipeline while the FPS in the video source changed from 5 to 60. As illustrated in Table. 2 co-locating the modules with the services clearly improves the frame rate compared to the baseline approach. Since the pose detector is the bottleneck of the pipeline, the highest frame rate that we have obtained in the experiments, is about 11 regardless of higher source frame rates.

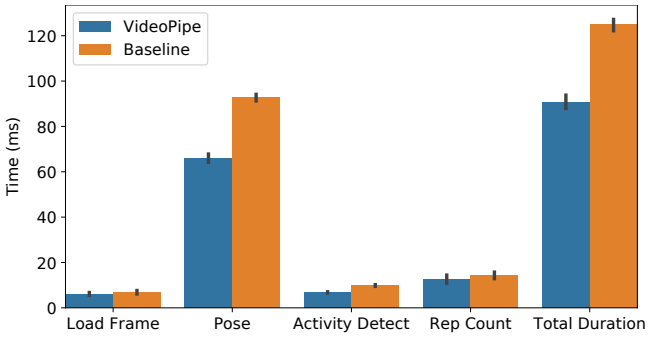

Figure 6. VideoPipe achieves lower latency for loading frames, pose detection, activity detection, rep counter and the pipeline. Among which, the delay for the pose detection is much lower than the remote API calls in the baseline as we call the pose detection service on the same machine.

Table 2. We show the end-to-end frame rates if we change the frame rate per second (FPS) in the video source. When comparing with the baseline, we achieve higher frame rates in different settings.

Source FPS	VideoPipe	Baseline	Two Pipelines
5	4.53	4.52	(4.56, 4.56)
10	8.21	7.79	(7.83, 7.83)
20	11.00	8.25	(9.44, 9.41)
30	10.72	8.33	-
60	11.03	8.01	-

5.2.2 Sharing the Services Across Pipelines

We show the re-usability of services across pipelines by executing the fitness application and the gesture control application simultaneously. These two pipelines share the pose detector service.

The results are shown in the fourth column of Table 2. The first and second numbers are the frame rates for the fitness pipeline and the gesture control pipeline. The performance of the fitness pipeline remains almost the same for frame rates less than 20 and adding the second pipeline did not significantly affect performance, which reflects on the flexibility of VideoPipe.

After the frame rate reaches 20, the end-to-end frame rate is decreasing, which indicates that we may have reached the limit of the shared pose detector service. It also implies that we should scale the services at this point, which is convenient in our design as the services are stateless.

6 Related Work

MediaPipe [26] is a framework for building audio/video processing pipelines. Similar framework is proposed in DeepStream SDK [7] where it provides APIs for TensorRT, video coding/decoding and visualization, which are optimized for Nvidia GPUs. However, both of the approaches only focus on running the pipeline on one device

instead of multiple devices. While serverless processing architectures have been proposed for the edge [14, 27], these systems do not deal with setting up a data pipeline for heterogeneous devices within a home.

In EdgeEye [25], a real-time video analytics service is proposed for the edge where applications can send requests to use services running on high performance servers. In this model, Deep Neural Networks run on these servers and applications can send frames to the servers for further processing. This system builds the pipelines by making service calls in the remote server while our approach will call the services by the modules on the same device directly. Similar systems [11, 32, 33] all send data to datacenters where the user has no control over and privacy concerns for how the data is managed and kept exists.

Furthermore, studies such as Vigil [33], VideoEdge [19] and Chameleon [23] focus on aggregating video analytics from a geographically distributed set of sources to improve the overall accuracy while reducing the computational overhead. However, we focus on how to do video analytics for a single video source in a video processing pipeline. Their work are orthogonal to ours. GStreamer [8] can also be used to build video processing pipelines as shown in [19]. However, GStreamer is mainly designed for video editing instead of streaming [26]. Moreover, it did not support video processing pipelines across multiple devices.

Finally, other pipeline related data processing frameworks are proposed in [2, 10, 13]. However, all of the systems focus on the batch data processing instead of real-time video processing in our paper. Batch processing and real time data processing have very different design considerations. For instance, in our real time video processing case, frame rates and lag are more important objectives. The concept of having a distributed operating system for home devices has been argued for and proposed by Dixon et al. [15, 16]. They propose HomeOS a multi-layer operating system for the home where PC-like abstractions for network devices are provided to users and developers. However, in our architecture, applications are deployed as a pipeline of modules suited for streaming applications.

7 Concluding Remarks

In this paper, we propose VideoPipe as a framework for efficient and flexible video processing pipelines in the home. With the design of *modules*, we can execute application code on any of the heterogeneous edge devices as they have the same runtime environment even though the hardware specifications may differ. We have designed and implemented *services* for heavy machine learning workloads. Services can be accessed from the modules locally and shared across multiple modules transparently. To validate the performance of our approach, we have created two applications on top of our framework and compared VideoPipe with the state-of-the-art approaches. Experimental results show that VideoPipe can substantially improve the performance of the video processing pipelines. For future work, we aim to include automatic deployment, scheduling and monitoring components to VideoPipe and also scale up services automatically based on workload.

References

[1] 2019. Amazon Go: Amazon Cashierless Stores. https://www.amazon.com/b?ie=UTF8&node=16008589011 Accessed: 2019-09-12.

[2] 2019. Apache Beam: An Advanced Unified Programming Model. http://zeromq.org/ Accessed: 2019-08-28.

[3] 2019. AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/index.html Accessed: 2019-09-12.

[4] 2019. Azure IoT Edge. https://azure.microsoft.com/en-us/services/iot-edge/ Accessed: 2019-09-12.

[5] 2019. Distributed Messaging - zeromq. http://zeromq.org/ Accessed: 2019-08-09.

[6] 2019. Duktape: An Embeddable JavaScript Engine. https://duktape.org/ Accessed: 2019-09-09.

[7] 2019. NVIDIA DeepStream SDK. https://developer.nvidia.com/deepstream-sdk Accessed: 2019-08-28.

[8] 2019. The GStreamer Library. https://gstreamer.freedesktop.org/ Accessed: 2019-08-28.

[9] 2019. Tizen: a Linux-based Mobile Operating System. https://www.tizen.org/ Accessed: 2019-09-09.

[10] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *Proceedings of the VLDB Endowment* 8, 12 (2015).

[11] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *IEEE Computer* 50, 10 (2017), 58–67.

[12] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*. Springer, 1–20.

[13] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[14] Eyal de Lara, Carolina S Gomes, Steve Langridge, S Hossein Mortazavi, and Meysam Roodi. 2016. Hierarchical Serverless Computing for the Mobile Edge. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 109–110.

[15] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. 2012. An Operating System for the Home. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 25–25.

[16] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Victor Bahl. 2010. The Home Needs an Operating System (and an App Store). In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 18.

[17] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2017. Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research. *arXiv preprint arXiv:1708.08028* (2017).

[18] Anne-Cecilie Haugstvedt and John Krogstie. 2012. Mobile Augmented Reality for Cultural Heritage: A Technology Acceptance Study. In *2012 IEEE international symposium on mixed*

and augmented reality (ISMAR). IEEE, 247–255.

[19] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing Camera Streams using Hierarchical Clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 115–131.

[20] Ahmad Jalal, Shaharyar Kamal, and Daijin Kim. 2014. A Depth Video Sensor-based Life-Logging Human Activity Recognition System for Elderly Care in Smart Indoor Environments. *Sensors* 14, 7 (2014), 11735–11759.

[21] Ahmad Jalal, Shaharyar Kamal, and Daijin Kim. 2015. Shape and Motion Features Approach for Activity Tracking and Recognition from Kinect Video Camera. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 445–450.

[22] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The Journey so Far and Challenges Ahead. *IEEE Software* 35, 3 (2018), 24–35.

[23] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 253–266.

[24] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the NetDB*. 1–7.

[25] Peng Liu, Bozhao Qi, and Suman Banerjee. 2018. Edgeeye: An Edge Service Framework for Real-time Intelligent Video Analytics. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 1–6.

[26] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. 2019. MediaPipe: A Framework for Building Perception Pipelines. *arXiv preprint arXiv:1906.08172* (2019).

[27] Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, and Eyal de Lara. 2017. Cloudpath: a Multi-tier Cloud Computing Framework. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 20.

[28] Sam Newman. 2015. *Building Microservices: Designing Fine-Grained Systems*. " O'Reilly Media, Inc.".

[29] Thomas Olsson, Tuula Kärkkäinen, Else Lagerstam, and Leena Ventä-Olkkonen. 2012. User Evaluation of Mobile Augmented Reality Scenarios. *Journal of Ambient Intelligence and Smart Environments* 4, 1 (2012), 29–47.

[30] Sana Tonekaboni, Mjaye Mazwi, Peter Laussen, Danny Eytan, Robert Greer, Sebastian D Goodfellow, Andrew Goodwin, Michael Brudno, and Anna Goldenberg. 2018. Prediction of Cardiac Arrest from Physiological Signals in the Pediatric ICU. In *Machine Learning for Healthcare Conference*. 534–550.

[31] Chenyang Zhang and Yingli Tian. 2012. RGB-D Camera-Based Daily Living Activity Recognition. *Journal of computer vision and image processing* 2, 4 (2012), 12.

[32] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proc. of 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 377–392.

[33] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The Design and Implementation of a Wireless Video Surveillance System. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 426–438.