EdgeEcho: An Architecture for Echocardiology at the Edge

Aman Khalid
Computer Science
Saint Louis University

Flavio Esposito
Computer Science
Saint Louis University

Alessio Sacco
Control and Computer Engineering
Politecnico di Torino

Steven C. Smart
School of Medicine, Cardiology
Saint Louis University

***Abstract*—Edge computing technologies have improved delays and privacy of several applications, including in medical imaging and eHealth. In this paper, we consider ultrasound technology and echocardiology (echo) and empower it with edge computing. Despite the many advances that ultrasound technology has seen recently, e.g., it is possible to perform echo scans using wireless ultrasound probes, the use of Artificial Intelligence (AI) techniques is becoming a necessity, for faster and more accurate echo diagnosis (not limited to heart diseases). While a few proprietary solutions exist that embed AI within echo devices, none of them uses resource-intensive tasks on handheld devices, and none of them is open-source. To this end, we propose EdgeEcho, an architecture that captures ultrasound data originated from handheld ultrasound probes and tags it using semantic segmentation performed on edge cloud. Our prototype focuses on optimizing the management of edge resources to address the specific requirements of echocardiology and the challenges of serving AI algorithms responsively. As a use case, we focus on a ventricular volume detection operation. Our performance evaluation results show that EdgeEcho can support multiple parallel medical video processing streaming sessions for continuing medical education, demonstrating a promising edge computing application with life-saving potential.**

I. Introduction

Computational offloading and hardware virtualization techniques have empowered several resource-intensive medical applications [1] that require heterogeneous resources for specific tasks. Distributed systems have to provide latency-sensitive tasks at the network edge and compute-intensive tasks that have the option to be offloaded to the cloud if necessary. On the one hand, recent advances in virtualization techniques, such as edge computing, have made practical the use of virtual machines to build such systems, accounting for the dynamic nature of these applications. On the other hand, applications of Deep Learning to Echocardiography (echo) have improved significantly [2]. Aside from such improvements, one area of research that has been so far largely unexplored is the use of edge computing to process echocardiography imagery in real-time. Image processing in a remote cloud needs to be reinforced to address delays, security, and privacy concerns.

Generating ultrasound images is a latency-sensitive operation that requires converting the ultrasound waves gathered from a test subject to an electrical signal, parsing it, and converting the then generated echo-pulse to image data. A sequence of such images forms the ultrasound video. While several progress has been made on other diagnoses, e.g., using artificial intelligence, the vast majority of echo diagnoses still rely on human expertise and their data collection operations remain buggy and a time bottleneck, leading to an incorrect diagnosis. Advancements in medical imaging have made it possible to advance state of the art, e.g., by broadcasting raw ultrasound feeds to a display using wireless probes for pre or post processing [3]. These solutions are often expensive and not open-source. *In this paper, we design and provide a proof-of-concept implementation of EdgeEcho, a deep learning-based system able to perform echo image processing, e.g., heart video segmentation, in real-time using edge computing resources, minimizing network latency but still having access to the high-performance computing of a Cloud.*

Thanks to the edge capabilities and recent advances in deep learning [4], such processing can be achieved in real-time using virtualized hardware with GPUs. Keeping track of the dynamic resources at the edge-cloud interface poses, however, several challenges. Among those, the need to maintain optimal performance despite constant updates in the global state of the system. It is sub-optimal, e.g., to use predefined values to initialize the internal data structures that keep track of different aspects of our system like: content caching, load-balancing, and resource discovery. Our system offers a live service that can serve such imagery session requests continuously with no manual intervention. To cope with this challenge, we employ memory-efficient (probabilistic) data structures that result into acceptable performance despite the demand spikes in the data flow to reliably process such a medical imagery stream.

Our contribution. In summary, we design EdgeEcho, an edge computing-based system able to analyze the echo feeds originating from a set of wireless ultrasound systems, with the goal of enabling robust and performant tele-echocardiology sessions. EdgeEcho uses use Optimized Cuckoo Filters (OCF), a congestion-aware membership testing data structure that we recently published [5]. We implemented our EdgeEcho architecture using open-source echo and cloud solutions, and we tested it over a use case of human heart ventricular volume detection.

The rest of the paper is structured as follows. In Section II we present the related work on edge computing orchestration and echocardiology image processing. In Section III we present the design of our EdgeEcho system design while in Section IV we describe with more details the implemented components. Then, we present a specific echocardiology use case in Section V and our performance evaluation results in Section VI. Finally, we conclude the paper in Section VII.

II. Related Work

In this section we discuss the present work on edge computing and echocardiography. We start focusing on the specific requirements of the edge network management and existing

978-3-903176-36-2 ©2021 IFIP

approaches, and then we describe applications related to our echocardiography use case. The edge cloud is particularly important for processing information close to the source, leading to reduced latencies. Examples of its usability have been shown in [6]–[8]. Among them, Clipper [9] is a low-latency online prediction serving system, which simplifies the deployment of a Machine Learning (ML) model across various frameworks and applications. Other projects similar to Clipper are LASER [10] and Velox [11], where the latter is considered as the solution providing the best performance. However, these deployments perform poorly when scaling over more complex ML models or larger datasets. A first attempt to address these scalability issues has been carried by Ray [12], a distributed framework for AI applications. It upgrades over the existing systems such as CIEL [13] by providing an option for distributed training and serving.

Alongside, ML has recently been applied to process echocardiographic to make cardiac imaging easier, faster, and more accurate. Some of these examples already validated are automated measurement features, including left ventricular ejection fraction, chamber dimensions, wall thickness, and Doppler measurements [14]. Avasalcai et al. [2] extends previous analysis on deep learning applicability to show that an improved CNN model can reliably identify local cardiac structures and anatomy, estimate volumetric measurements and metrics of cardiac function, and predict systemic human phenotypes that modify cardiovascular risk.

What makes our implementation different from the aforementioned technologies or their combination is the highly tailed nature of our system to support echocardiology. The sensitive nature of our use case required us to optimize at every step of the image generation. Moreover, the generalized solutions mentioned above do not address the requirement to manage or provision GPUs in real-time to perform semantic segmentation.

III. Architecture Overview and Workflow

The prime objective of EdgeEcho is to enable remote echocardiography to efficiently respond to medical requests from multiple users. Having this in mind, we build our EdgeEcho as a distributed system, as shown in Figure 1. Our solution comprises four main components: probes, stream processor, orchestrator, and analyzer node(s). *Probes* are wireless medical devices that emit an array of ultrasound data that is used to generate the video. The logic of digital beam-forming and image generation is offloaded to the *stream processor* module, which is located at the edge of the network for faster data transfer. This operation is offloaded for two reasons - to make handheld probes lightweight, and to retain the ability to spawn them at a location closest to the probe. Next are the *analyzer nodes* which are responsible for transforming raw ultrasound images into segmented video streams. Finally, the *orchestrator* is the core component of our architecture as it is responsible for several operations essential in the workflow, *e.g.*, scaling up and down network resources, GPU provisioning, and service discovery.

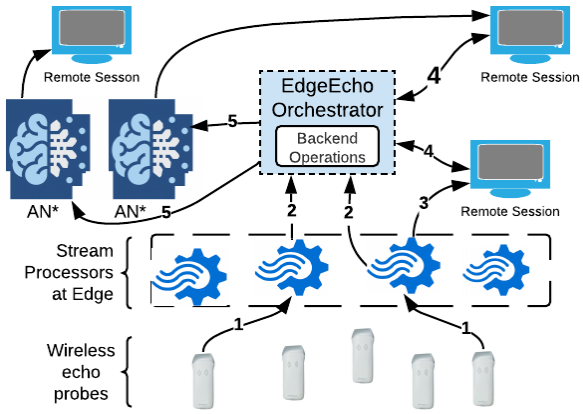


Fig. 1: EdgeEcho data flow overview, which depicts interactions when a new request is served

Echo live-stream workflow. At the arrival of a new user request, (1) the handheld probe sends a request to the nearest edge stream processor to start the image streaming. The probes can start communicating with the first idle Stream Processor (SP) that they discover. (2) Once the connection is established, the orchestrator component is notified, and the list of live sessions to multicast is updated. The (3) orchestrator itself exposes REST APIs that can be used by mobile applications or websites so that the user can select and view the type of session from a web interface. When a user selects a segmented stream operation, (4) the orchestrator component checks if a node is currently active and serves that request by forwarding it to one active node. In the case of no nodes available, the orchestrator creates a new node with enough resources to satisfy the session request, sends the quest, and publishes its IP address to the edge network so echo clients can subscribe to it.

IV. EdgeEcho Components Design and Offered Functionalities

Our EdgeEcho architecture is built using a Docker environment, which supports the four key components. In this section we discusses the details of each of these components and how they are organized.

A. Ultrasound Probe

The ultrasound probe is used to conduct the medical examination by placing it directly on the body of the patient. In this paper, we simulate a handheld probe with networking capabilities that imitate the state-of-the-art pulse-echo sequence. The probe acts as both the emitter and receiver of the ultrasound signal, and generates a bit array of pre-recorded ultrasound data.

B. Stream Processor

We design our EdgeEcho so that each probe is associated with a stream processor, a compute node that continuously listens to an incoming stream of bits originating from the probe. It runs the tasks offloaded by the probe, which is a reliable technique for offloading tasks at the edge [15]. The Stream Processor applies specified filters to the raw image

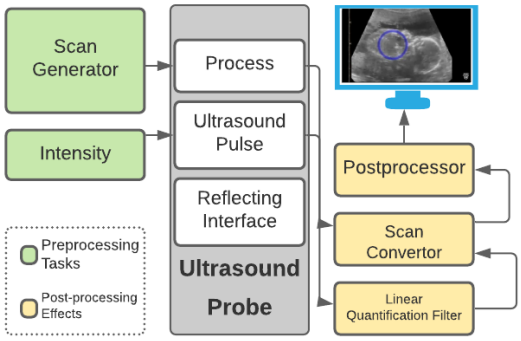


Fig. 2: Active tasks during stream processing. Our objective is to offload some computational intensive stream analysis tasks to the network edge.

data. The image is initially in the form of 8-bit intensity pixels. The final value of each pixel is calculated based on the offset of the RGB table and the filter being used,. Out of the various processes in Figure 2 only the ones in green and yellow can be offloaded. For this paper we re-implemented the processes marked in yellow. They collect information sent by wireless probes and form a raw ultrasound video, and apply some post-processing effects.

C. Image Analyzer

Image Analysis is performed by the Analyzer Nodes (ANs), which are containerized compute nodes requiring more resources than Stream Processor nodes and preferably a GPU for better performance, since serving the machine learning algorithm. We compared some real-time image processing algorithms that suited the requirements while considering the following constraints – sparse data availability, the requirement for high accuracy from a quite small training set. One example of image operation regards the mark of the ventricles of the human heart, which requires relatively low computing resources. In particular, we use a two-stage framework [16] that scans the images provided and generates proposals, and secondly classifies the proposals and generates bounding boxes and masks. This specific task requires manual annotation because the automatically generated proposals cant detect heart ventricles and instead detect the entire heart.

D. The Orchestrator

The orchestrator of our EdgeEcho is central to all operations within the system. It has the collective state of the entire system's resources, such as available CPU/GPUs, prepackaged segmentation applications, current and past sessions. Following is an overview of the logical components of the operator and how their interactions enable the desired features. The $session$ object stores all the necessary information about serving a request, including information of nodes involved. Moreover, the orchestrator stores the list of available wireless probes in a $probeList$ array. When a request for a session arrives at the orchestrator, it sets up a stream processor to start the ultrasound feed. One of the two operations is performed depending on the type of the request - if a raw feed is requested, it is sent to the client immediately; otherwise, it starts collecting the necessary information for segmentation. In the second case, the orchestrator checks the resources required to start the prepackaged segmentation applications corresponding to the request. Specifically, the orchestrator gathers the $currentSystemState$ object, which contains details regarding the available compute resources, and compares it to the quantities requested by the current request. If adequate resources are found, the orchestrator issues the command to start a new Analyser Node (AN) by calling a construction method. This method accepts two parameters: a reference to the stream processor that must be connected to the AN, and the ID image that can serve that request. While the Analyzer Node starts up, the orchestrator blocks the resources. Finally, the orchestrator shares the details of the live AN with the client.

Along with these tasks, our orchestrator performs operations that are not directly responsible for serving a user's request, but aim to provide resources that enable them. In particular, three backend operations are run: resource discovery, content caching, packet routing.

Resource discovery. This module is used to track the available resources. The orchestrator component of our EdgeEcho system tracks the available resources in the system using our Optimized Cuckoo Filter (OCF) [5]. This module of the orchestrator serves two purposes. First, it creates virtual machine or containerized images and ensures that the number of machine images does not exceed a predefined value. Additionally, it tracks the maximum amount of resources by maintaining a separate OCF, which throws an error when capacity is reached.

Content Caching. Our system also enables serving segmented media streams in real-time. The content of the video can change depending on the type of segmentation algorithm being used during a stream. The streams are stored temporarily in a Least Recently Used (LRU) cache that can scale out even on a different machine. In such a way, it can be extended or flushed as the size of the OCF shrinks or expands.

Packet Routing. The last operation entails the routing of packets. One or more nodes serve a user's request in our system. This subset of nodes is assigned to a network bridge that (i) enables the communication between these nodes by routing the packets appropriately, and (ii) connects the nodes to the internet. Using packet sniffing, we monitor the packets entering and leaving these virtual network bridges, saving the metric in our OCF.

E. Optimized Cuckoo Filters for fast look-ups

Fast lookup is a primary requirement for our system and is needed for all three backend functions of EdgeEcho. Traditional linear search algorithms are not optimal for querying large key-stores in a dynamic distributed setting. They occupy a considerable amount of space as the internal key-store grows large and does not have a constant lookup time. For this reason, we implemented Membership Testing (MT) [17]

via Optimized Cuckoo Filters (OCF) [5], which has constant lookup time and has specific merits. For example, MT allows us to check the existence of a key in a datastore very rapidly. This data structure stores the hashes of keys currently present in the datastore, which makes it lightweight. Every hashed value is mapped to a key using a hash function, which is the reason why lookup time is constant.

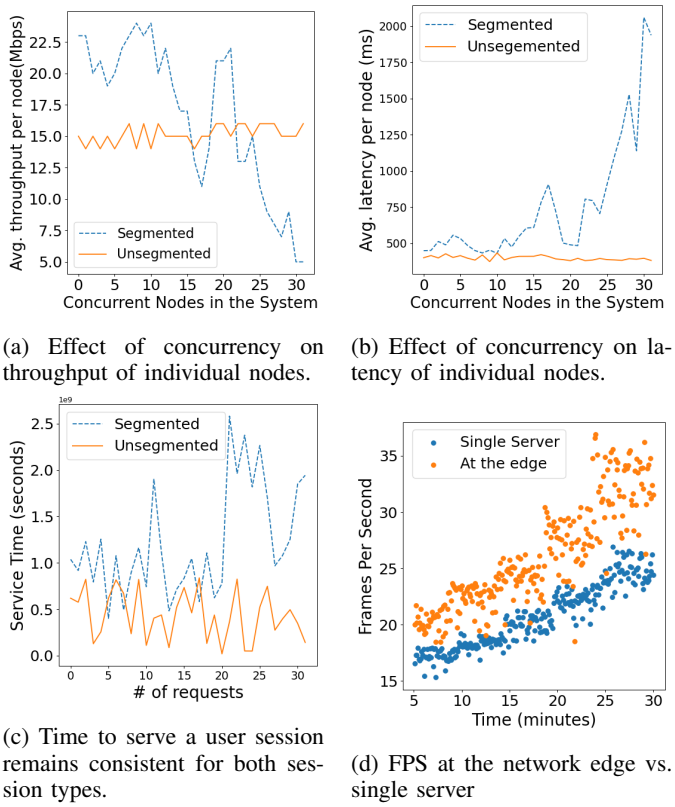
V. Use Case: Ventricular Volume Detection at the Edge

We use a convolutional deep learning model [18] that can perform real-time instance segmentation. This model delivers a decent framerate with an average of 30 fps. This method of segmentation divides its tasks into two parallel subtasks - (1) generating a set of prototype masks and (2) predicting per-instance mask coefficients. Splitting a more complex task into smaller individual tasks helps us towards our aim of optimizing resource usage. We trained the aforementioned model using the EchoNet-Dynamic dataset [19]: a dataset of echocardiography videos and has labeled measurements of features necessary for ventricular volume detection.

VI. Evaluation

In this section we report the results of experiments to evaluate the benefits brought by our system to enhance medical ultrasound using edge computing. We first test how the throughput for individual nodes of our EdgeEcho system is affected when the number of concurrent nodes in the system is increased. We consider two different options in the system: a *segmented* version, where the segmentation processing is applied over the collected images, and *unsegmented* version, which does not enforce the image processing. Figure 3a depicts the average throughput of the system as increases number of concurrent nodes. We can observe that, for more users, the average throughput is cut down for the segmented version, thus reducing the availability of bandwidth per node. Similar conclusions can also be observed by considering the latency, as shown in Figure 3b. The latency rises significantly for nodes serving a segmented session, while in the unsegmented session, the latency of the system is not affected by the number of concurrent users in the system. This is due to the fact that, for the additional processing as in segmentation algorithms, the data to transmit among services is much higher than compared to nodes running plain stream. These results are extremely important to us to determine the set of resources that are required to run our EdgeEcho without incurring in a considerable downgrade of performance. We can thus conclude that our current deployment supports around 20/25 nodes and that, for more nodes, we need to increase also the available bandwidth capacity.

Aside from system side metrics, to observe metrics from the user's perspective, we evaluate the time it takes for EdgeEcho to serve an incoming request. We define the service time as the time taken to serve the first byte since the request arrives. We report this serve time in Figure 3c, studying its evolution for an increasing number of requests. A session is considered served



(a) Effect of concurrency on throughput of individual nodes.

(b) Effect of concurrency on latency of individual nodes.

(c) Time to serve a user session remains consistent for both session types.

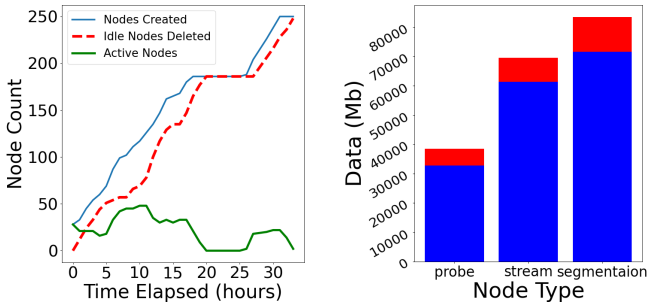
(d) FPS at the network edge vs. single server

Fig. 3: (a) Throughput, (b) latency, and (c) service time evolution when more nodes join the system. These results suggest the minimum set of resources required to smoothly run our EdgeEcho, i.e., 20/25 nodes in segmented sessions. (d) Frames per second obtained by running the same model at edge vs. on a single server. Edge computing clearly exhibits advantages in real-time video streaming.

when the broadcast node is live. It must be noted that for a segmented request, the system has to create both a streaming and a segmentation server, and connect them to an output. As observed in the graph, the service time remains constant for the first few requests for both types of requests. When the number of requests increases, the service time of unsegmented is constant, while it increases for the segmented because the resources are occupied by previous requests. Hence, similar conclusions to the previous pair of graphs hold and we can confirm our previous findings.

To assess the benefits brought by edge computing, we then run the same semantic segmentation algorithm on a traditional client-server application and on EdgeEcho. In Figure 3d we compare the frames per second (FPS) of the segmented feed in the scenarios, and we can clearly observe how running the application at the edge leads to an increased FPS. This is due to the location of the processing, which is closer to the source of the streaming process, along with our optimized edge management.

Additionally, EdgeEcho uses a smart cleanup mechanism to remove idle nodes that have served their purpose, to save resources for future requests. Once all the users have



(a) System nodes creation-disruption evolution over time (b) Throughput vs. Goodput during a test run

Fig. 4: (a) Nodes are created and destroyed to release resources. Orchestrator maintains optimal resource utilization by retiring nodes whose session has been terminated. (b) Data efficiency in terms of goodput/throughput ratio of the system for a period of 40 hours with consistent requests of both types.

disconnected from a session, the nodes responsible for that session become idle, and the orchestrator removes those nodes to free up system resources. We record the lifespan of nodes in our system for a 40-hour session, and we report the results in Figure 4a. As it can be observed, the number of created nodes follows the number of incoming requests and grows over time. However, since the system cleans up idle resources, we can also observe how the nodes are destroyed when the request is satisfied. This leads to a number of active nodes that is constantly limited, demonstrating that the system assures that only necessary nodes are kept alive.

Lastly, we evaluate the number of errors encountered by our EdgeEcho throughout our experiments which can occur due either to the unavailability of GPU resources or application malfunctions. Whenever a request is unsuccessful during operation, data is wasted, and nodes need to be restarted. Our engine takes care to monitor the status of the requests and, when necessary, restart the process, always assuring that the request is performed. In Figure 4b we summarize the amount of data wasted (in megabytes) through a 40-hour operation of our system. We can observe how the wasted data is a very small portion compared to the goodput and that the additional control traffic is negligible.

VII. Conclusion

This paper presented EdgeEcho, a system that enables remote echocardiology, with segmentation capabilities and the ability to serve parallel requests. We discussed our implementation and optimizations that enable us to serve segmented ultrasound streams at a decent time to serve. We demonstrated the gain in FPS by running the same semantic segmentation algorithm on our system vs. a monolithic client-server setup. Finally, we described other performance aspects of our system like downscaling, throughput, and latency. In the future, we plan to evaluate some static aspects of our system, e.g., the exact amount of resources needed for a particular session type. Further, we will improve EdgeEcho by making the system more fault-tolerant by decentralizing the backend operations of the orchestrator.

Acknowledgement

This work has been partially supported by NSF Awards CNS-1836906 and CNS-1908574.

References

[1] A. Sacco, F. Esposito, G. Marchetto, G. Kolar, and K. Schwetye, "On edge computing for remote pathology consultations and computations," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 9, pp. 2523–2534, 2020.

[2] A. Ghorbani, D. Ouyang, A. Abid, B. He, J. H. Chen, R. A. Harrington, D. H. Liang, E. A. Ashley, and J. Y. Zou, "Deep learning interpretation of echocardiograms," *NPJ digital medicine*, 2020.

[3] D. Lertsilp, S. Umchid, U. Techavipoo, and P. Thajchayapong, "Improvements in ultrasound elastography using dynamic focusing," in *The 4th 2011 Biomedical Engineering International Conference*, 2012.

[4] M. Thoma, "A survey of semantic segmentation," *arXiv preprint arXiv:1602.06541*, 2016.

[5] A. Khalid and F. Esposito, "Optimized cuckoo filters for efficient distributed sdn and nfv applications," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 77–83, 2020.

[6] H. Cao, M. Wachowicz, and S. Cha, "Developing an edge computing platform for real-time descriptive analytics," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 4546–4554, IEEE, 2017.

[7] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 373–377, IEEE, 2018.

[8] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "Sustainable task offloading in uav networks via multi-agent reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 5003–5015, 2021.

[9] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 613–627, 2017.

[10] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang, "Laser: A scalable response prediction platform for online advertising," in *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 173–182, 2014.

[11] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan, "The missing piece in complex analytics: Low latency, scalable model management and serving with velox," *arXiv preprint arXiv:1409.3809*, 2014.

[12] P. Moritz *et al.*, "Ray: A distributed framework for emerging ai applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 561–577, 2018.

[13] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand, "Ciel," in *Proc. 8th ACM/USENIX Symposium on Networked Systems Design and Implementation*, pp. 113–126, 2011.

[14] D. Medvedofsky and V. Mor-Avi, "Three-dimensional echocardiographic quantification of the left-heart chambers using an automated adaptive analytics algorithm," *European Heart Journal-Cardiovascular Imaging*, 2018.

[15] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[17] D. Gupta and S. Batra, "A short survey on bloom filter and its variants," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 1086–1092, 2017.

[18] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact: Real-time instance segmentation," 2019.

[19] D. Ouyang, B. He, A. Ghorbani, N. Yuan, J. Ebinger, C. P. Langlotz, P. A. Heidenreich, R. A. Harrington, D. H. Liang, E. A. Ashley, *et al.*, "Video-based ai for beat-to-beat assessment of cardiac function," *Nature*, vol. 580, no. 7802, pp. 252–256, 2020.