RhythmEdge: Enabling Contactless Heart Rate Estimation on the Edge

Zahid Hasan†, Emon Dey†, Sreenivasan Ramasamy Ramamurthy†, Nirmalya Roy†, Archan Misra§
†*Mobile Pervasive & Sensor Computing Lab, Center for Real-time Distributed Sensing and Autonomy (CARDS)*
†*Department of Information Systems, University of Maryland, Baltimore County, United States*
§*School of Computing & Information Systems, Singapore Management University, Singapore*
{zhasan3, edey1, rsreeni1, nroy}@umbc.edu, archanm@smu.edu.sg

Abstract—**The primary contribution of this paper is designing and prototyping a *real-time edge computing system*, *RhythmEdge*, that is capable of detecting changes in blood volume from facial videos (Remote Photoplethysmography; rPPG), enabling cardio-vascular health assessment instantly. The benefits of *RhythmEdge* include non-invasive measurement of cardiovascular activity, real-time system operation, inexpensive sensing components, and computing. *RhythmEdge* captures a short video of the skin using a camera and extracts rPPG features to estimate the Photoplethysmography (PPG) signal using a multi-task learning framework while offloading the edge computation. In addition, we intelligently apply a transfer learning approach to the multi-task learning framework to mitigate sensor heterogeneities to scale the *RhythmEdge* prototype to work with a range of commercially available sensing and computing devices. Besides, to further adapt the software stack for resource-constrained devices, we postulate novel pruning and quantization techniques (Quantization: FP32, FP16; Pruned-Quantized: FP32, FP16) that efficiently optimize the deep feature learning while minimizing the runtime, latency, memory, and power usage. We benchmark *RhythmEdge* prototype for three different cameras and edge computing platforms while evaluating it on three publicly available datasets and an in-house dataset collected under challenging environmental circumstances. Our analysis indicates that *RhythmEdge* performs on par with the existing contactless heart rate monitoring systems while utilizing only half of its available resources. Furthermore, we perform an ablation study with and without pruning and quantization to report the model size (87%) vs. inference time (70%) reduction. We attested the efficacy of *RhythmEdge* prototype with a maximum power of $8W$ and a memory usage of $290MB$, with a minimal latency of 0.0625 seconds and a runtime of 0.64 seconds per 30 frames.**

Index Terms—**rPPG, Edge Computing, System Prototyping**

I. Introduction

Heart rate (HR), the number of heartbeats (contractions of the ventricles) per minute, is a crucial and versatile biomarker, whose anomalous pattern can indicate various physiological conditions. For instance, Bradycardia (slow HR) and Tachycardia (fast HR) could indicate a possibility of a heart attack, infections, fever, asthma or breathing problems, anemia, sick sinus syndrome, high blood potassium, or under-active thyroid gland [1].

Usually, HR can be acquired by using a medical-grade Electrocardiograph (ECG/EKG) or Photoplethysmography (PPG). Although ECG provides the most accurate measurement, it requires careful placement of multiple lead-type ECG electrodes on the body surface in a clinical setup that *severely restricts its application for ubiquitous applications*. PPG provides a non-invasive, low-cost HR measurement in ambulatory settings as regular HR check-ups and primary screening for thorough ECG. It is an optical technique to detect volumetric changes in blood in the peripheral circulation. ***Wearable devices*** or ***Video-based Remote Sensing*** enables us to leverage the PPG mechanism to estimate HR. The ubiquitous wearables are predominant as regular HR, Heart Rate Variability monitoring system [2]. However, they require close skin contact with special optical sensors [3] to measure PPG. Alternative to wearables, ***Video-based Remote Sensing*** obtains remote Photoplethysmography (rPPG) in contact-less manner by capturing the variations in the light reflection off the human skin using video sensors [4]. Because of the pandemic-driven and growing deployment of camera sensors, rPPG has potential as a ubiquitous HR monitoring system providing enhanced safety to healthcare professionals, and the general public. However, rPPG input video suffers from low SNR and high variability in PPG estimation due to sensor-subject angles, distances, relative velocity, exposed light types [5], camera sensor heterogeneity (sensor properties, resolution, sensitivity, frames per second (fps)), inherent video compression mechanism. Moreover, the expensive computation of video processing and corresponding PPG estimation severely hinders its application as ubiquitous HR monitoring system.

We encounter a set of technical and scientific research challenges towards developing a ubiquitous rPPG application system. The technical parts deal with appropriate settings to capture PPG from the subjects, facilitate off-line and real-time operations, and devise rPPG systems compatible with a range of available low-cost resource constraint edge platforms and available cameras. The scientific challenges encompass the development of a robust model to extract PPG and techniques to effectively scale down the model for resource-efficient operation inside edge devices [6]. In this research study, we address these practical challenges in developing a portable rPPG system and postulate the following *contributions*.

- **Model Compression.** We propose domain-specific compression techniques to compress existing deep learning-based rPPG models while maintaining the baseline model performance (0.15 RMSE). Our approach reduces the model size (10 times), power consumption, memory usage (25%),

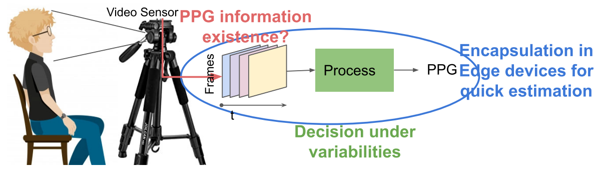

Video Sensor
PPG information existence?
Frames
t
Process
PPG
Encapsulation in Edge devices for quick estimation
Decision under variabilities

Figure 1: rPPG System Overview and Challenges.

and latency to enable edge platforms deployment. We validate and ablate our approach on three public datasets by performing empirical and statistical similarity analyses.

- **rPPG System Development.** We prototype a contact-less system, *RhythmEdge*, for real-time and offline rPPG estimation. Our prototype is compatible with heterogeneous cameras (differing sensitivity, resolution, fps, etc.) and is executable on diverse resource-constrained edge devices (different architectures). Particularly, *RhythmEdge* is deployed on three platforms (*NVIDIA Jetson Nano, Google Coral Development Board, Raspberry Pi*) and facilitates three commercially available cameras (*web camera, action camera* and *DSLR*) with an option of storing and notifying longitudinal rPPG.
- **Evaluation and Benchmarking.** We evaluate *RhythmEdge* on three public datasets and our collected realistic data for both off-line and real-time operation. Further, we test *RhythmEdge* thoroughly for robustness, device overheating, time complexity, and memory overheads to ensure stable operation and provide system bench-marking parameters across different design choices. Finally, we open-source our new data, instructions, codes, baseline model weights, and compressed model to enable future research [1].

II. Background and Related Work

Video rPPG. Human facial veins' blood volume varies synchronously with the hearts' systolic and diastolic cycles, and these variations are embedded in the light reflected from the exposed skin. Proximal video sensors (RGB [4], Near-infrared (NIR) [7], IR [8]) with sufficient sensitivity and resolution can pick up these variations embedded in the video on various channels of the video from the exposed facial skin's light reflection. HR can be obtained by counting the peaks per minute from the obtained rPPG signals.

rPPG. Filtering-based methods like filtering the green channel [4], chrominance signal analysis [9], plane orthogonal to the skin tone [10], matrix decomposition [11] have been applied to estimate PPG from skin videos. Signal Source Separation based methods such as Independent Component Analysis [12], source separation [13] have shown their potential in estimating rPPG. Recent deep learning (DL) methods have found their application in the rPPG system. *VitaMon* [14], [15] have proposed convolutional neural network (CNN) and fully connected layer-based architecture for rPPG estimation.

Model Compression. Compression techniques reduce computations of over-parameterized DNN models by learning a fast and compact model that approximates the function learned by the baseline DNN model [16]. The existing techniques can be broadly categorized into post-training compression (pruning, quantization) and developing a new model. Pruning-based methods identify and remove the insignificant nodes/weights of models [17]. The quantization methods reduce the floating-point precision (FP) for each node/weight [18], eventually reducing the inference complexity and computations. The quantization also helps the model to comply with different hardware platforms [19], [20]. Besides post-training optimization, [21] develop a compact model from a large trained model. Further, as shown in [22], these methods can also be carefully combined to achieve even greater compression gain.

rPPG on limited resource devices. The limited resource devices rPPG system is an emerging field. Authors of [23] integrated NIR LED sensors with field-programmable gate array (FPGA) to detect motion and HR in real-time. [24] assessed the HR estimation performance of the Facereader^{TM} by Noldus. However, these existing systems lack an end-to-end setup having a camera, edge processing, and output display under the same roof. To our knowledge best, *RhythmEdge* is the first attempt towards developing and benchmarking a complete real-time rPPG system on edge devices.

III. Model Development

We develop the CNN-based baseline rPPG-extraction model and a compressed edge-device compatible model [figure 3] using a computationally powerful server.

1) Baseline Model: We build the data-driven baseline DNN model by collecting rPPG datasets, recreating a capable architecture, training, and validating the network. Here, we chose the open-source *CamSense* DL-based approaches [15] to develop a baseline model as it offers PPG reconstruction, interpretability, and adaptability for new environments.

Network architecture: The architecture consists of four CNN, two inception layers, two Global average pooling (GAP), and two fully-connected (FC) feed-forward (FF) layers with l2 regularization. Each CNN layer couples with Batch Normalization, Rectified Linear Unit (ReLU) activation function, and max-pooling layers. The final FCFF layer performs a regression task. The output layer is the head network for the multi-task learning (MTL) network. The model with a single head consists of about 1M parameters with an input size of $100 \times 100 \times 40$. The network optimizes the regression loss of the weighted sum of the root mean squared error and sign agreement loss between target and predicted PPG.

Data Preprocessing: The input video preprocessing prepares the high-resolution video at 30 fps for the inference model. The network operates on facial video instances of normalized consecutive 40 green channel frames $(1.33$ seconds) of 100×100 spatial resolution. During training, the network weights are updated to minimize the loss objective between the network estimated and the normalized target PPG. Further, to support robust learning, the network receives augmented (random face crop, brightness variations) training instances.

Transfer learning: We have utilized a transfer learning (TL) to develop a baseline model with minimal labeled examples

[1] https://github.com/mxahan/rPPG_edge_implementation

Table I: Size and accuracy comparison for baseline model, pruned model, pruned-qunatized models for different sparsity with different pruning priors.

Sparcity %	Model size Prune (KB)		Model size Tflite (KB)		Test Error		Model size (KB)	
	FC	Full	FC	Full	FC	Full	FC layer	Full
10	600	550	540	500	0.9	1.6	200	180
40	650	600	600	560	0.7	1.4	250	200
70	750	680	690	620	0.5	1	270	250
90	1080	920	1050	870	0.4	0.7	350	300

for the potential application areas. We initialized the recreated model with MTL pretrained weights and fine-tune the model for different camera sensors and variabilities (backgrounds, light variations) in the target environments.

2) *Model Compression:* The baseline model is computationally heavy and cannot be deployed directly to most limited-resource devices. We propose domain-specific compression (pruning and quantization) methods for the rPPG application without dropping performance to scale down the developed high-performing rPPG model for fast and resource-efficient PPG inference.

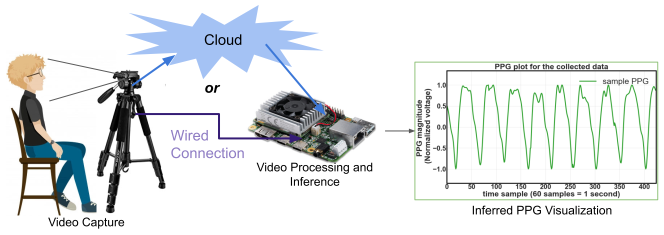
We have explored the weight pruning method because of its architectural stability and prevalence [25]. We prune the baseline model $f(x; W)$ with parameter W, and produces a new sparse model $f(x; M \odot W')$. Here, W' and W are different, $M \in \{0,1\}^{|W'|}$ is an explicit binary mask that reduces some weights to 0, $\odot$ represents element-wise matrix product operation. As dropping weights reduces accuracy, we try to recover performance by fine-tuning and pruning in an iterative process. We control the pruning parameters to find the balance between compression and accuracy. Based on our empirical observation in the rPPG setting, we propose to prune the fully connected layers for the best performance while achieving sufficient compression.

3) *Quantization:* We apply floating-point precision (FP) 16 and 32-bit quantization that improves latency and reduces memory, model size, and computation overhead by compromising the number of computational bits for each weight. In our case, an increased quantization level resulted in only a negligible drop in estimation accuracy. Finally, we experiment with quantization and pruning combinations to generate four different models to compare and benchmark the rPPG performance: Quantized FP32 (Q32), Quantized FP16 (Q16), Pruned Quantized FP32 (PQ32), and Pruned Quantized FP16 (PQ16).

4) *Executable Model:* We prepare the lightweight and embedded device compatible TensorFlow Lite model (tflite) using the TensorFlow Lite converter from the quantized-compressed model and equip the devices with the executable model via the GitHub platform. We have used TensorFlow to develop our models, post-training optimization, and TfLite conversion relying upon the servers' resources.

IV. System Architecture

Our rPPG system, *RhythmEdge* (prototype in figure 4), consists of *sensing*, *preprocessing*, and *PPG inference* phases [Figure 1]. Firstly, the edge devices receive the facial videos at 30 fps for extracting PPG. *RhythmEdge* facilitates operation in



Cloud
or
Wired Connection
Video Processing and Inference
Video Capture
PPG plot for the collected data
sample PPG
PPG magnitude (Normalized voltage)
time sample (60 samples = 1 second)
Inferred PPG Visualization

Figure 2: Illustration of *RhythmEdge* in real-time and offline.

both offline and real-time rPPG inference. The edge computing devices *RhythmEdge* perform video processing (center crop, reshape, and frame crop) on the received videos. Finally, the device executes the rPPG model to infer PPG from the processed frames. *RhythmEdge* also stores and sends the approximated PPG to authorized users.

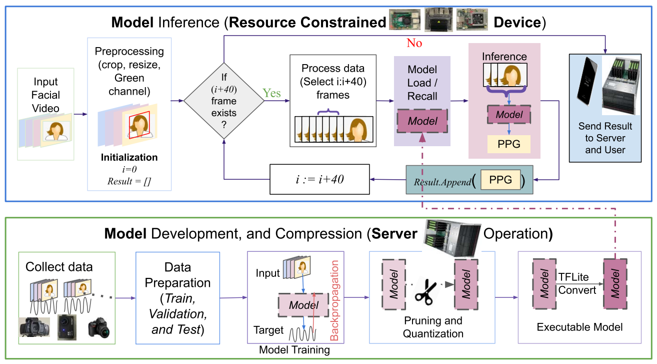


Figure 3: Model development and compression (bottom) and rPPG inference inside the limited resource devices (top).

1) *Real-time video input:* We integrate the USB-C compatible camera with the computing devices' embedded USB-A port to capture facial videos in real-time. The *ffmpeg* command enables the devices to receive real-time high-resolution video streams from the connected sensors. We provide bash scripting commands to control and schedule the video recording. However, the edge devices may automatically reduce input video fps and resolution due to the specific compatibility requirement. To meet these technical issues and retain the video with consistent quality, we propose to use the *MJPEG* compression scheme and overcome the compatibility concerns.

2) *Edge Device Operations:* **Sensing Input.** The limited resource devices receive videos either in real-time or offline mode. In real-time, the devices use integrated cameras to capture real-time videos by its automated bash commands. Alternatively, for offline inference, these devices receive human facial videos (captured externally by DSLR, IR, and NIR cameras) via memory chips or cloud platforms (google drive or GitHub) using their I/O communication ports.

Preprocessing and PPG approximation. The computing devices process the received video and apply the model for PPG approximation. Here, we tackle the technical challenges of incorporating video processing python libraries in a single pipeline under the memory, time, and power constraints. The installed OpenCV processes each 1.33 second of video to

$100 \times 100 \times 40$ shape of one input instance by taking only the consecutive green channel frames. In practice, we have considered 22S of video together consisting of $22 \times 30 = 660$ frames, approximately equivalent to the batch of 16 input instances. Finally, the devices use the compressed tflite model and tensorflow runtime interpreter to extract PPG from processed video.

Quick analysis. The computing devices analyze the estimated PPG signal (HR approximation, Visualization, server data recording) and perform Fast Fourier Transform to report the HR from estimated PPG instantaneously. The devices transfer the inferred PPG to the sever using onboard Wi-Fi to compute additional features, such as the peak to peak distance, output quality, and PPG irregularity. The server stores the users' longitudinal PPG information with the date and time and informs the user via email communication. The pseudocode of the entire process is described in algorithm 1.

Algorithm 1: Raw video inference in devices.

```
Data: Stored Video
Result: Corresponding rPPG
initialization;
Result: []
$tfliteNet$ : Compressed rPPG network;
$VIndex$ : 0; %%Video Frame Pointer
while $[VIndex + 40]^{th}$ video frame exists do
    $VFrames \leftarrow$ Data$[VIndex : VIndex + 40]$;
    $DReady \leftarrow preprocess(VFrames)$;
    $TemprPPG \leftarrow tfliteNet(DReady)$;
    Result$.append(TemprPPG)$;
    $VIndex \leftarrow VIndex + 40$;
Send Result to server and user
```

V. System Implementation

In this section, we describe the hardware-software integration and implementation of RhythmEdge. We also articulate the three public datasets and our collected dataset containing both indoor and outdoor environments.

1) Hardware: **Limited Resource Computing Devices.** We have experimented *RhythmEdge* with three available heterogeneous edge devices (based on incorporated architecture, memory, and sensor integrating capabilities) to check our system's adaptability across multiple hardware architectures and resource configurations. i) **Coral Dev Board (GCDB).** The Dev Board is a single-board computer that executes fast ML inference. It scales to production using the onboard Coral System-on-Module combined with PCB hardware covering an edge TPU coprocessor capable of performing 4 trillion operations per second (TOPS), using 0.5 watts for each TOPS. It supports wireless connectivity, a Mendel OS, and is also compatible with Linux. ii) **Jetson Nano (JN).** NVIDIA's JN developer kit can run multiple neural network models for parallel inference by supporting a high-performance DL inference runtime for neural networks called TensorRT. Besides, by integrating the JetPack SDK kit, JN can utilize the latest Linux driver packages with Linux OS and CUDA-X accelerated libraries and APIs for DL. iii) **Raspberry Pi (RPi).** The RPi covers a series of small single-board computers featuring a Broadcom system on Chip with an integrated ARM-compatible CPU supporting the Raspbian OS. RPi-4 Model B, 2019, is equipped with a 1.5 GHz 64-bit quad-core ARMv8 CPU processor, 4GB RAM, onboard 802.11ac Wi-Fi, USB 2.0, 3.0, an HDMI port, and Bluetooth.

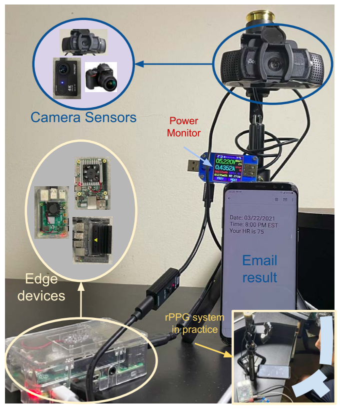


Figure 4: *RhythmEdge* system complete setup. The demonstration is framed in the inset (Bottom-right).

Video Sensors. We have experimented with three heterogeneous camera sensors: DSLR (offline with the highest quality), Action Camera (AC), and Web-Camera (WC) to show the versatility of *RhythmEdge* . i) **Offline input.** Our DSLR Nikon-D3500 camera with the AF-P DX NIKKOR 18-55mm f/3.5-5.6G VR lens supports 30 fps videos. The device uses a memory chip or cloud to communicate with edge devices in offline mode. ii) **Real-time input.** We demonstrate two different cameras for real-time video capture: Logitech C920s PRO HD WC and Akaso-Ek7000 AC. AC has professional 4K 25Fps and 2.7K 30Fps wide-angle video sensor, built-in Wi-Fi, and HDMI. Meanwhile, the WC camera is a USB class-supported device capable of capturing 1080p flat-angle video at 30 fps with both YUV and MJPEG format.

2) Dataset: To demonstrate the wide applicability, along with our collected dataset we additionally studied three public rPPG datasets [table II] to showcase the scalability and generalization of the *RhythmEdge* components.

MPSC-rPPG dataset [15] We utilize the public MPSC-rPPG dataset for developing the *RhythmEdge* baseline model. The dataset contains facial videos using DSLR (H.264 video codec with $10,000$ bit rate) cameras and Empatica E4 record simultaneous PPG ground truth. The dataset contains videos under different artificial lights. The dataset contains volunteers with different skin colors, gender, and facial characteristics.

Following the MPSC-rPPG dataset protocol, we have collected datasets for new environments using AC (H.264 High $4:2:2$ profile with 2000 bit rate, wide-angle) and WC (flat angle). We have considered multiple realistic environments like natural lights, angular exposure, and sensor variabilities.

MERL [26]. MERL-Rice NIR Pulse (MR-NIRP) dataset consists of 8 subjects, varying facial expressions with near-infrared (NIR), RGB raw, and RGB demosaiced modalities at 30 fps. The data provide simultaneous finger PPG at $60Hz$. **UBFC-rPPG [27].** The UBFC-rPPG dataset contains about 8

Table II: Experiment Dataset Overview

Dataset (Devices)	Subs	Videos	Time (min/vid)	Variabilities
MERL (NIR, RGB, Demosaiced)	8	24	3.5	Beard, Glass, skin color, gender
MPSC-rPPG (DSLR)	8	14	7	Lighting, Beard, skin
UBFC-rPPG (RGB)	8	42 (8)	-	Lighting, Beard
AC	2	4	8	Lighting, Beard, angle
WC	2	4	8	Lighting, angle

simple and 42 realistic RGB videos with varying fps from 28 to 30. For each subject, simultaneous PPG varies from 30 to 62.5 Hz, records are available as the target.

3) Compressed model development: For our implementation, we recreate the baseline network and utilize the shared weights of open-source *CamSense* approaches [15] on Keras functional API environment. The Keras API permits pruning compression (the original model occurred on the TensorFlow subclass environment, which does not allow pruning yet). We experimented with both complete model pruning (both CNN and FC layers) and only FC layers pruning to compress the baseline model. We prune the model layer uniformly to match the predefined sparsity in both experiments. We iterate over the sparsity range from 10 to 90 percent and keep the sparsity constant for each setup. Finally, we fine-tune the pruned network for 30 epochs and closely observe their training and validation loss characteristics to ensure quality.

VI. Evaluations

We benchmark the performance by evaluating models' statistical performances, accuracy, latency, memory usage, and power consumption during *RhythmEdge* execution.

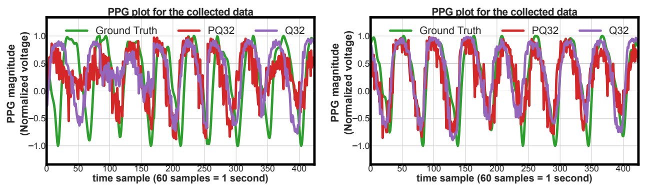


Figure 5: rPPG Plot of two test cases: Ground truth (Empatica E4), PQ32, and Q32. The outputs contain random noise of high frequencies, however, it retains the PPG information.

1) Statistical Behavior of Estimated PPG signal: We provide the statistical analysis of closeness between the models' estimated signal and the target PPG signal to demonstrate the models' robust performance. We compute the probability distribution of the normalized cross-correlation values between different video segments' ground truth and models' predicted PPG over the entire dataset. The distribution bias towards high probability values shows the higher statistical similarities between estimated and ground-truth signals. Our evaluation result shows the consistent performance of the models, although the pruned model performance dropped a bit [figure 6].

2) Accuracy: We evaluate the models' accuracy on three bases: peak detection, false-peak, and RMSE. The server compares ground-truth PPG with the edge devices inferred PPG (for the Q32, PQ32, Q16, and PQ16 models) and baseline

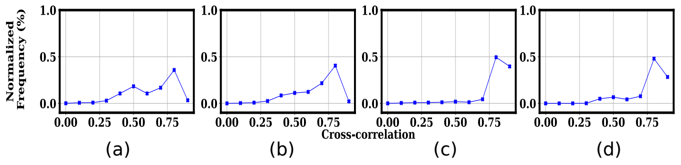


Figure 6: Pruned (a) test, (b) train, Baseline (c) train, (d) test.

model prediction on the same scale and reports the average values of these metrics [figure 5]. The detection accuracy depends on the video streaming camera sensors and the inferring models and is independent of the computing devices. Accordingly, we discuss our accuracy results on the camera sensor and compressed model. Firstly, out of all three camera sensors, the DSLR performed the best due to the DSLR's fine-grained signal capturing ability as detailed in the earlier section V-2. However, unlike AC and WC, DSLR is not directly compatible with computing devices. Further, in between AC and WC, the WC performs better than AC sensors. It can be explained by the properties of the AC and WC sensors. Unlike the WC flat-angle, the wide-angle of the AC camera captures more noise and reduces the PPG SNR in the recorded video. Moreover, WCs provide stable performance compared to AC.

We find the compressed models performing similarly to the baseline model [table III]. We explain it by the nature of the considered data. The higher bits of the input and model weights contain insignificant information and trimming their precision until FP16 points almost have no consequences. These findings are consistent across multiple test trials, and we report the mean performance of different rPPG models. We depict a sample performance of the Q32 and PQ32 model for real-time test data using GCDB and WC in figure 5. Table III shows their comparative results for the three datasets. Finally, we report the model's average performance and put forth the comparative analysis between the models in the table IV.

3) Execution Time: The execution time of the system measures its' response lag and compatibility in real-time operation by reporting the time to process the input data. The rPPG system execution time depends on the specific components and the executing computing devices. We measure the execution time of the five components in the rPPG system, (1) OpenCV video processing, (2) video preprocessing, (3) Data loading, (4) Model load, and (5) Inference time on three experimented computing platforms. We measure the metrics of the [28] to benchmark the system components' run-time by finding the first run time (warm-up time after reboot) and the consecutive execution time (regular time) for each section. Moreover, we report the data and model load time. The loading, warming, and regular run-times of the components measure the system's response. We leveraged the python 'time' library to estimate the run-time by measuring the difference between the start and end time for the different operation blocks.

We observe that the OpenCV processing time dominates the among the components of execution time across all the computing devices as OpenCV requires operating on HD videos with high fps. Moreover, JN outperformed the other two computing devices, GCDB and RPi, in almost every

Table III: Accuracy comparison with baseline for both compressed models using TP, FP and FN for three datasets.

Subject		Personalized model			Q32			MSE	PQ32			MSE
		TP /Actual Peaks	FP	FN	TP /Actual Peaks	FP	FN	Q32 and Origin	TP /Actual Peaks	FP	FN	PQ32 and Origin
MPSC-rPPG RGB	S1	12/12	1	0	12/12	1	0	0.1	12/12	1	0	0.15
	S2	8/12	4	4	8/12	4	4	0.1	8/12	4	4	0.16
	S3	10/11	2	1	10/11	2	1	0.05	10/11	2	1	0.1
	S4	10/12	2	4	10/12	2	4	0.1	10/12	2	4	0.2
	S5	11/12	1	2	11/12	1	2	0.1	11/12	1	2	0.2
MERL RGB	Sub 1	8/9	1	2	8/9	1	2	0.1	8/9	1	2	0.15
	Sub 2	7/10	3	3	7/10	3	3	0.08	7/10	3	3	0.1
	Sub 3	11/11	0	0	11/11	0	0	0.05	11/11	0	0	0.1
	Sub 4	0/11	11	11	0/11	11	11	0.1	0/11	11	11	0.15
MERL NIR	Sub 1	8/9	1	2	8/9	1	2	0.1	8/9	1	2	0.15
	Sub 2	7/10	3	3	7/10	3	3	0.08	7/10	3	3	0.1
	Sub 3	11/11	0	0	11/11	0	0	0.05	11/11	0	0	0.1
	Sub 4	0/11	11	11	0/11	11	11	0.1	0/11	11	11	0.15
UBFC rPPG RGB	Sub 1	8/15	6	7	8/15	6	7	0.1	8/15	6	7	0.15
	Sub 2	11/11	0	0	11/11	0	0	0.1	11/11	0	0	0.12
	Sub 3	11/11	0	0	11/11	0	0	0.05	11/11	0	0	0.1
	Sub 4	10/12	2	2	10/12	2	2	0.05	10/12	2	2	0.1

Table IV: Comparison of average $\%$ accuracy for four compressed models with the baseline model for all datasets and video sensors. Their performance shows that the compressed models matches the baseline performance for peak detection.

Model	MERL (RGB)			MERL (NIR)			UBFC-rPPG			MPSC-rPPG (DSLR)			AC			WC		
	TP R	FP/ Actual peaks	FN R	TP R	FP/ Actual peaks	FN R	TP R	FP/ Actual peaks	FN R	TP R	FP/ Actual peaks	FN R	TP R	FP/ Actual peaks	FN R	TP R	FP/ Actual peaks	FN R
Base	91	11	9	88	10	12	83	16	17	87	12	13	73	26	27	82	19	18
Q32	91	11	9	88	10	12	83	16	17	87	12	13	73	26	27	82	19	18
PQ32	91	11	9	88	10	12	83	16	17	87	12	13	73	26	27	82	19	18
Q16	91	11	9	88	11	12	83	16	17	87	12	13	73	26	27	82	19	18
PQ16	90	10	10	88	11	12	83	16	17	84	12	16	71	27	29	82	19	18

execution time. Between GCDB and RPi, to our surprise, GCDB outperformed RPi in most of the system components, although RPi is a more resource-heavy computing device. This result can be attributed to the fact that despite having narrower specifications than RPi, the GCDB may be using a lighter version of OpenCV and has better compatibility with Python and TensorFlowLite. However, JN and RPi performs best in data and model loading. The comparative illustration of our findings is depicted in figure 7 and table V. These run-time values enable us to design scheduling of rPPG components for different devices while avoiding system clogging.

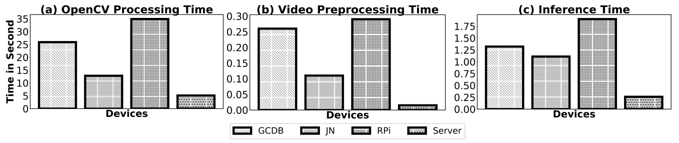


Figure 7: Run-time: (a) OpenCV Processing (b) Video Preprocessing, and (c) Inference on server and computing devices.

4) Memory: Investigating the memory footprint of deep models in the limited resource devices enables us to apprehend the feasibility of the *RhythmEdge* . The feasibility is satisfied if the maximum memory of the system components during execution never exceeds the device memory capacity. The memory handling issue depends on the device hardware architecture and background software. To measure run-time memory usage, we monitor the device idle and run-time memory using the *top* command for GCDB and RPi, and *jtop* for JN during component execution and report the maximum and average value during five successive iterations by calculating the difference between idle and execution memory. To benchmark execution memory, we observe the occupied memory for three major tasks: (1) OpenCV video processing, (2) Data preparation, and (3) inferences [20]. We empirically observe that the model execution memory depends on the compression methods. The PQ16 (most compressed) requires the least memory. As compression reduces, the inference memory increases. Secondly, OpenCV operation occupies maximum run-time, whereas the PPG inference utilizes the most memory. It is due to the serialized frame-by-frame computation by OpenCV. Meanwhile, the inference operation considers all the inputs and models simultaneously.

Finally, we compare the memory performance for the *RhythmEdge* components in three devices. The JN requires much higher memory since there is no dedicated memory. The RPi and GCDB (slightly worse than RPi) are memory-efficient among the three devices as they contain dedicated on-chip memory. However, since the GCDB, RPi stores the network on chip, it offer some slack in memory requirement by minimizing the burden of repetitive model loading. Furthermore, we sort the average disk storing memory for the uncompressed (3.5 MB), Q32 (2.5 MB), PQ32 (2.4 MB), Q16 (1.4 MB), and PQ16 (1.3 MB) models. We report our overall results in table VI and visualize them in figure 9.

5) Power: We investigate the power profile *RhythmEdge* components as a part of our feasibility study and report the maximum power and average power in all three devices. To measure these values, we place a wattmeter with the power source to observe the change in power while executing different operations and report the average difference between

Table V: Run-time of *RhythmEdge* for the most compressed PQ16 model. Q32, Q16, and PQ32 showed similar behavior.

		OpenCV Process Time (Second) (Vid siz 22s)		Preprocessing Time (Second) (22 S of video)		Data Load Time (Second) (Optional) (22 S of video)		Tflite Model Load Time (Second)		Inference Time (Second) (22 S of video)	
		warm Up	Regular	warm Up	Regular	warm Up	Regular	warm Up	Regular	warm Up	Regular
PQ 16	Server	5.12	5.12	0.02	0.015	0.005	0.005	0.01	0.0002	0.26	0.22
	GCDB	25.9	25.9	0.27	0.26	0.043	0.039	0.029	0.003	1.34	1.33
	JN	**12.8**	**12.8**	**0.12**	**0.11**	**0.023**	**0.023**	0.07	0.003	**1.01**	**0.99**
	RPi	34.9	34.9	0.29	0.29	0.19	0.18	**0.001**	**0.0005**	1.43	1.4

Table VI: Memory usage of the computing devices. Results is generated for 22S of video and same TfLite models. The top and below values represents maximum and average memory respectively.

	OpenCV Video 22S	Q32 Model (Batch size=16)		PQ32 Model (Batch size=16)		Q16 Model (Batch size=16)		PQ16 Model (Batch size=16)	
		Process	Inference	Process	Inference	Process	Inference	Process	Inference
Coral Dev	107 MB	**111** MB	284 MB 246 MB	**111** MB	278 MB 232 MB	106 MB	284 MB 236 MB	106 MB	268 MB 220 MB
Jetson Nano	112 MB	202 MB	291 MB 257 MB	202 MB	280 MB 250 MB	197 MB	290 MB 252 MB	197 MB	284 MB 245 MB
Rasberry Pi	**104** MB	160 MB	**210** MB **179** MB	120 MB	**205** MB **168** MB	**98** MB	**205** MB **173** MB	**98** MB	**192** MB **139** MB

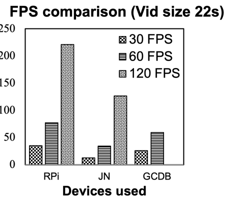


Figure 8: Trend of OpenCV processing time with varying video fps.

the idle and execution power over multiple times. Among the rPPG components, video input requires more power since, during video recording, the devices energize both the camera sensor and the computing device simultaneously. We observe similar power profiling independent of the models, devices, and rPPG system components for other operations. We also noticed that the idle power of the limited resource devices shows an inversely proportional relationship with their specifications. We report power consumption profile for all the devices in table VII and visualize *RhythmEdge* them in figure 9.

Table VII: Power consumption (in Watts) by devices during different component execution for the computation of 22S video (Batch size 16). The top and below values represents maximum and average power.

Device Power	Idle Power	Video input	Ope-nCV	Inference			
				Q32	PQ32	Q16	PQ16
GCDB	3.5	7.9 6.9	5.8	**6** **5.4**	**5.9** **5.3**	**5.9** **5.4**	**5.9** **5.3**
JN	**1.6**	7.89 7.03	**5.07**	6.22 5.9	6.12 5.8	6.3 5.9	6.07 5.7
RPi	2.6	**7.2** **6.7**	6.6	6.23 5.8	6.28 5.6	6.44 6.03	6.38 5.93

Our empirical result demonstrates the feasibility and compatibility of the *RhythmEdge* on the limited resource devices.

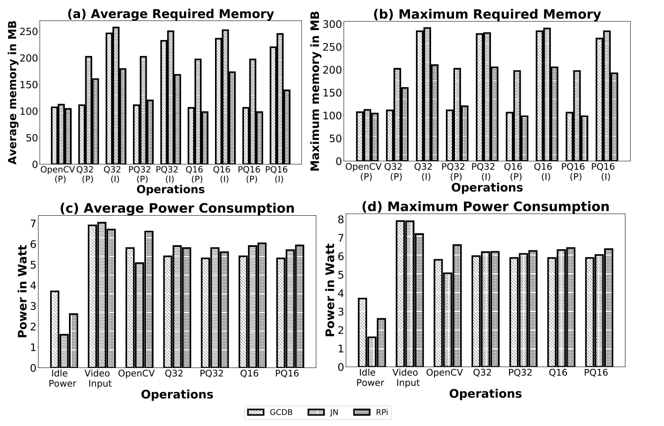


Figure 9: Comparison of (a) average required memory, (b) Maximum required memory, (c) Average Power consumption, and (d) Maximum Power consumption during running OpenCV Processing (P), Data Processing (P), and different models' inference (I) for different models.

VII. Discussion

Model Compression Ablation Study. The CNN layer pruning significantly deteriorated rPPG model performance. However, as shown in Table I, an equivalent pruning of the FC layer parameters does not result in significant performance degradation. We explain this behavior by the nature of tasks the layers are performing. The earlier convolution layers performs the general tasks of finding face and extracting inherent PPG [29]. The FC layers reconstruct target PPG from CNN features. Although pruning FC layers increase the reconstruction error, the model retain the underlying PPG shape [figure 5] and act as a regularization.

System Robustness. We have studied the system robustness by varying the angle, distance between subject and camera, lighting condition (natural and artificial), subject skin, gender, and relative motion. i) *RhythmEdge* can reliably extract PPG from the angular face (one side of the face) and occluded faces from three feet distances. It also performs robustly for subjects' gender and skin tone variations. ii) The system performance degrades with increasing subject-sensor distance as the weak diffusion signal gets significantly weaker due to the inverse squared law. Further, the maximum workable subject-sensor distance depends on the camera sensor properties. For example, the DSLR (specified lens) capture video PPG from about seven feet and the viable distance may increase with more powerful lens. WC captured video significantly degrades after four feet in terms of PPG extraction. The AC has the least range, and the model fails to extract PPG beyond three feet. iii) The lighting condition plays a vital role in encoding PPG as it dictates the diffusion reflection scattered from the skin. Although the systems' performance degrades with decreasing light intensity, *RhythmEdge* can estimate PPG consistently under dim light for videos within three feet distance. We further tested the system under natural light and successfully estimated PPG (under shade). However, direct exposure to sunlight degrades the performance. iv) The relative motion between the subject and the sensors degrades the system's performance and usability.

Experimenting with varying camera parameters. To demonstrate our system's robustness under varying sensor types, we test the system with different video settings by varying fps and resolution. We experimented with combinations of three different fps: 30, 60, and 120 and two resolutions: 720p and 1080p. Since with varying inputs only the OpenCV processing varies, we report the OpenCV processing time in this experiment. The trend of OpenCV processing time with fps variation is illustrated in figure 8. As expected, the OpenCV processing time tends to ascend with fps increment. Among edges, The JN is the fastest and the GCDB fails to perform the processing of 120 fps video due to its limited memory and computation power.

VIII. Conclusion

In this research, we have successfully developed a real-time rPPG system, *RhythmEdge*, on three limited resource platforms by addressing relevant challenges. Firstly, we compress the baseline rPPG model while preserving the accuracy. Further, *RhythmEdge* supports two different video sensors to infer rPPG from real-time videos. Finally, we have benchmarked *RhythmEdge* by reporting the computation time, memory usage, and power consumption of the devices during the execution of rPPG components. We have validated the feasibility of real-time cardiovascular activities monitoring system inside multiple edge devices by evaluating under multiple realistic conditions and using three public datasets.

Acknowledgements

This research is supported by NSF CAREER grant 1750936, U.S. Army grant W911NF2120076.

References

[1] D. Zhang and et al., "Association between resting heart rate and coronary artery disease, stroke, sudden death and noncardiovascular diseases: a meta-analysis," *Cmaj*, vol. 188, no. 15, pp. E384–E392, 2016.

[2] D. Castaneda, A. Esparza, M. Ghamari, C. Soltanpur, and H. Nazeran, "A review on wearable photoplethysmography sensors and their potential future applications in health care," *International journal of biosensors & bioelectronics*, vol. 4, no. 4, p. 195, 2018.

[3] Z. Zhang, "Photoplethysmography-based heart rate monitoring in physical activities via joint sparse spectrum reconstruction," *IEEE transactions on biomedical engineering*, vol. 62, no. 8, pp. 1902–1910, 2015.

[4] W. Verkruysse, L. O. Svaasand, and J. S. Nelson, "Remote plethysmographic imaging using ambient light.," *Optics express*, vol. 16, no. 26, pp. 21434–21445, 2008.

[5] D. J. McDuff, E. B. Blackford, and J. R. Estepp, "The impact of video compression on remote cardiac pulse measurement using imaging photoplethysmography," in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, IEEE, 2017.

[6] R.-T. Wu, A. Singla, M. R. Jahanshahi, E. Bertino, B. J. Ko, and D. Verma, "Pruning deep convolutional neural networks for efficient edge computing in condition assessment of infrastructures," *Computer-Aided Civil and Infrastructure Engineering*, vol. 34, no. 9, 2019.

[7] E. Magdalena Nowara and A. et al., "Sparseppg: Towards driver monitoring using camera-based vital signs estimation in near-infrared," in *CVPR*, pp. 1272–1281, 2018.

[8] W. Chen and D. McDuff, "Deepphys: Video-based physiological measurement using convolutional attention networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 349–365, 2018.

[9] G. De Haan and V. Jeanne, "Robust pulse rate from chrominance-based rppg," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 10, pp. 2878–2886, 2013.

[10] W. Wang et al., "Algorithmic principles of remote ppg," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 7, pp. 1479–1491, 2016.

[11] B.-F. Wu and et al., "Motion resistant image-photoplethysmography based on spectral peak tracking algorithm," *IEEE Access*, vol. 6, pp. 21621–21634, 2018.

[12] D. McDuff, S. Gontarek, and R. W. Picard, "Remote detection of photoplethysmographic systolic and diastolic peaks using a digital camera," *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 12, pp. 2948–2954, 2014.

[13] R. Macwan, Y. Benezeth, and A. Mansouri, "Heart rate estimation using remote photoplethysmography with multi-objective optimization," *Biomedical Signal Processing and Control*, vol. 49, pp. 24–33, 2019.

[14] S. Huynh, R. K. Balan, J. Ko, and Y. Lee, "Vitamon: measuring heart rate variability using smartphone front camera," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 2019.

[15] Z. Hasan, S. R. Ramamurthy, and N. Roy, "Camsense: A camera-based contact-less heart activity monitoring," *Smart Health*, vol. 23, p. 100240, 2022.

[16] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, p. 535–541, Association for Computing Machinery, 2006.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, pp. 1135–1143, 2015.

[18] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[19] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[20] N. D. Lane et. al., "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the 2015 international workshop on internet of things towards applications*, 2015.

[21] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[22] Y. Gong et. al., "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[23] Y.-C. Lin, N.-K. Chou, G.-Y. Lin, M.-H. Li, and Y.-H. Lin, "A real-time contactless pulse rate and motion status monitoring system based on complexion tracking," *sensors*, vol. 17, no. 7, p. 1490, 2017.

[24] S. Benedetto, C. Caldato, D. C. Greenwood, N. Bartoli, V. Pensabene, and P. Actis, "Remote heart rate monitoring-assessment of the facereader rppg by noldus," *PloS one*, vol. 14, no. 11, p. e0225592, 2019.

[25] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[26] E. M. Nowara and et al., "Sparseppg: towards driver monitoring using camera-based vital signs estimation in near-infrared," in *2018 IEEE/CVF CVPRW*, pp. 1353–135309, IEEE, 2018.

[27] S. Bobbia, R. Macwan, Y. Benezeth, A. Mansouri, and J. Dubois, "Unsupervised skin tissue segmentation for remote photoplethysmography," *Pattern Recognition Letters*, vol. 124, pp. 82–90, 2019.

[28] M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource characterisation of personal-scale sensing models on edge accelerators," in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pp. 49–55, 2019.

[29] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *arXiv:1411.1792*, 2014.