accepted to appear in Proceedings of 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks - ACM Workshop on Wireless Security and Machine Learning (WiseML 2022)

Online Stream Sampling for Low-Memory On-Device Edge Training for WiFi Sensing

Steven M. Hernandez
hernandezsm@vcu.edu
Department of Computer Science
Virginia Commonwealth University
Richmond, Virginia, USA

Eyuphan Bulut
ebulut@vcu.edu
Department of Computer Science
Virginia Commonwealth University
Richmond, Virginia, USA

ABSTRACT

Deploying machine learning models on-board edge devices allows for low latency model inference and data privacy by keeping sensor data local to the computation rather than at a central server. However, typical TinyML systems train a single global model which is duplicated across all edge devices. This leads to a model that is generalized to the training data, but not specialized to the unique physical environment where the device is deployed. In this work, we evaluate how we can train machine learning models on-board low-memory edge devices with streams of incoming data. When using these low-memory devices, storage space is at a minimum and as such, representative data samples from the data stream must be captured to ensure that the models can improve even with a limited set of available training samples. We propose the Variable Low/High Loss sampling method for selecting representative data samples from a data stream and demonstrate that our methods are able to increase the accuracy of the machine learning model compared to state-of-the-art methods. We demonstrate the applicability of our proposed method for WiFi sensing based human activity detection, where WiFi signals are used to predict human activities in a given environment without requiring sensors on their bodies.

CCS CONCEPTS

• **Computing methodologies → Online learning settings**; • **Security and privacy → Privacy protections**.

KEYWORDS

On-device machine learning, importance sampling, WiFi sensing, edge learning, TinyML

ACM Reference Format:
Steven M. Hernandez and Eyuphan Bulut. 2022. Online Stream Sampling for Low-Memory On-Device Edge Training for WiFi Sensing. In *Proceedings of the 2022 ACM Workshop on Wireless Security and Machine Learning (WiseML '22), May 19, 2022, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3522783.3529521

1 INTRODUCTION

TinyML [4] has appeared as a field combining machine learning with embedded systems research where models are designed to run on resource constrained devices at the edge rather than at central servers. Performing model inference at the edge allows for lower latency prediction making, removes the reliance on an internet connection, improves user privacy by keeping private data on-location and finally enables new applications for smart homes, smart workplaces, improved at-home healthcare, and more. While TinyML is gaining further attention for allowing model inference at the edge, the next natural extension is to allow model training to be performed at the edge as well [12]. However, because of the constrained nature of the devices at the edge, training models on-device and at the edge will provide additional challenges. Most notably, edge devices have (i) *small storage* for retaining training samples, (ii) *low memory* which means that machine learning models must be shallow rather than deep, and (iii) *slow computation speed* which means that each epoch of model training takes longer to perform which thus limits the amount of training that can be performed at these edge devices.

In this work, we focus on one of these steps towards making on-device model training a reality. Specifically, we consider the case where the edge device used for on-device training has only a small amount of local storage for storing training samples. Additionally, we assume that a stream of annotated training samples are made available to the device which far exceed the available storage. Due to the low storage available, the device must be selective in storing a sub-sample of the stream for on-device training. Allowing for on-device model training ensures that deployed models can adapt to changes in sensor readings due to changes in the environment. This is particularly important for WiFi sensing [9] where received WiFi signals are directly affected by the physical environment due to multipath signal propagation.

In this work we make the following contributions:

(1) We introduce a novel method for sampling streams of sensor data on-device which we call *Variable Low/High Loss (VLHL)* sampling.
(2) We evaluate our proposed method on a simulated data stream for the novel task of WiFi-sensing human activity detection (HAD) using low-cost edge devices.
(3) We demonstrate that VLHL achieves higher accuracy than baseline methods as well as state-of-the-art methods: Most Recent Lowest Loss (MRLL) and Most Recent Highest Loss (MRHL).

The rest of the paper is organized as follows. In Section 2 we discuss background information on WiFi sensing as well as techniques used for machine learning at the edge. After this, we discuss our streaming sample selection method as well as review model training on-board edge devices in Section 3. We evaluate the results of our proposed algorithm compared to other state-of-the-art sampling methods in Section 4. Finally, we make our concluding remarks in Section 5.

2 BACKGROUND

2.1 WiFi Sensing

WiFi sensing [9] uses the radio-frequency (RF) signals found propagating throughout our homes and offices to detect and sense physical properties of the environment. When RF signals are transmitted from a transmitter (TX) to a receiver (RX), they propagate over multiple unique paths (signal multipath) which may propagate directly to the RX or might reflect off surfaces in the environment such as people, furniture and walls. Tracking the changes in the signal multipath over time can allow us to identify physical changes in the environment such as large furniture being moved, and can even be used for human activity detection. WiFi RF signals offer many benefits for sensing in indoor environments because RF signals: (i) can propagate through walls which allows for one sensor to be used over multiple rooms, (ii) are transmitted omnidirectionally rather than in a single direction as would be found in a camera based system, and (iii) are present on even the smallest smart devices within our homes which means that we can gain additional sensing coverage by leveraging these RF signals passively. Recent work in [6] shows that WiFi sensing can be performed on the low-cost ESP32 WiFi-enabled microcontroller (ESP32-MCU)[1]. We leverage this ESP32-MCU in this work for collecting channel state information (CSI) to achieve WiFi sensing in many physical locations throughout our smart home environment.

Channel state information is captured in communication systems such as 802.11 which use orthogonal frequency-division multiplexing (OFDM), to allow for data to be encoded in multiple subcarrier frequency allowing for higher symbol throughput as well as resilience to signal fading and shadowing caused by multipath interference in the channel. CSI is modeled with

$$\hat{x}^{(i)} = H^{(i)} x^{(i)} + \eta^{(i)} \tag{1}$$

where i is the subcarrier index, x is the transmitted signal, $\hat{x}$ is the received signal, η is a noise vector, H is a complex vector containing the channel state information denoting the transformation change required from the input x to the output $\hat{x}$. The complex CSI vector contains 64 subcarriers where 52 are data-subcarriers while 12 are null-subcarriers where the CSI value for each subcarrier is defined as a complex number with a real component ($H_r^{(i)}$) and an imaginary component ($H_{im}^{(i)}$). We can transform this raw CSI into amplitude:

$$A^{(i)} = \sqrt{\left(H_{im}^{(i)}\right)^2 + \left(H_r^{(i)}\right)^2}, \tag{2}$$

and phase:

$$\phi^{(i)} = atan2\left(H_{im}^{(i)}, H_r^{(i)}\right). \tag{3}$$

[1]https://stevenmhernandez.github.io/ESP32-CSI-Tool/

In this work, we solely look at the amplitude and do not take into account the phase.

2.2 Machine Learning at the Edge (TinyML)

Recent work in the field of TinyML has pushed towards a number of improvements for combining machine learning with embedded edge systems. Research into TinyML can be split into three categories: deep learning algorithm design, hardware design and applications of TinyML [13]. However, TinyML focuses on performing model inference using models that were pretrained at some more powerful system before being embedded into embedded MCUs.

While model inference is the primary concern of TinyML, there are a few works that do consider methods for training models on low resource embedded MCUs. For example, both TinyOL [13] and TinyFedTL [10] take the approach that a pretrained TinyML model can be personalized on-device at the edge by training a single output layer for the given machine learning model. Specifically, all layers before the final layer are stored on-board and run through inference like a normal TinyML model. The output of this TinyML model is then input into a separate single training layer which can be trained much quicker and with a simpler training algorithm than full backpropagation. TinyTL [3] takes a different approach where each layer is still trained on-board but only a subset of model parameters are trained while the others remain frozen. Specifically, the model weights are frozen and only the biases are trained on-device. This allows training to occur on all of the multiple layers throughout the machine learning model while still being able to performed in a timely manner.

2.3 Applications

Thus far, both WiFi sensing and TinyML have been applied in a number of settings including health tracking [5, 19], smart vehicles and traffic monitoring [2, 15], UAVs [11, 14], agriculture [1, 8] and more. However, until now, it is rare that WiFi sensing is combined together with TinyML or with on-device edge machine learning in the research literature.

3 PROPOSED METHOD

3.1 Streaming Sample Selection

Assume that we are given a streaming dataset ($\mathcal{X}$) and labels (y) where $\mathcal{X}[t]$ is a single CSI sample and $y[t]$ is the corresponding label at time $t \in \mathcal{T} = \{1, 2, \ldots, t, \ldots, T-1, T\}$ where 1 is the earliest possible time instance and T is some unreachable final time instance. Our goal is to capture a buffer of CSI-samples ($\tilde{\mathcal{X}} \subseteq \mathcal{X}$), along with a buffer of labels for each CSI sample ($\tilde{y} \subseteq y$), such that $|\tilde{\mathcal{X}}| = |\tilde{y}| < T$. To simplify our notation in the following sections, we assume that we have a buffer $\mathcal{B} \subseteq \mathcal{T}$ such that $|\mathcal{B}| = |\tilde{\mathcal{X}}| = |\tilde{y}|$ which is used to store the time-index of the samples captured in $\tilde{\mathcal{X}}$ and $\tilde{y}$ such that $\tilde{\mathcal{X}} = \{\mathcal{X}[\mathcal{B}[1]], \ldots, \mathcal{X}[\mathcal{B}[|\mathcal{B}|]]\}$ and $\tilde{y} = \{y[\mathcal{B}[1]], \ldots, y[\mathcal{B}[|\mathcal{B}|]]\}$. In this notation[2], any operations performed on $\mathcal{B}$ directly translate to similar updates to both $\tilde{\mathcal{X}}$ and $\tilde{y}$. At each time instance t, we must decide:

[2]We note that while this simplifies our notation, real-world implementations would not actually retain $\mathcal{B}$. Instead, all operations would automatically apply to both $\tilde{\mathcal{X}}$ and $\tilde{y}$ directly.

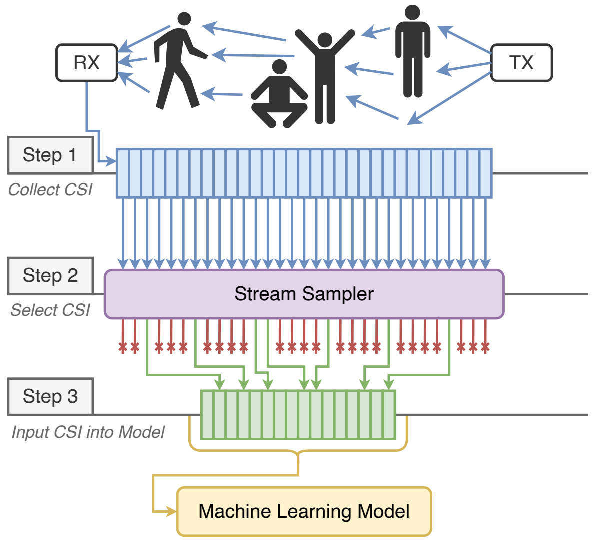

Figure 1: Illustration of streaming sample selection. In step 1, CSI multipath signal data is captured from the transmitter (TX) and the receiver (RX) for human activity detection. In step 2, the stream sampler then selects which samples are high quality. Step 3 places these quality samples into a buffer ($\mathcal{B}$) which is then used for training the machine learning model.

(1) *Should we store this sample in the buffer or ignore it?*
(2) *If we store it, which sample in $\mathcal{B}$ should be replaced?*

This process is illustrated in Fig. 1 where at the top, human participants perform some physical activities while WiFi signals are sent from the transmitter (TX) to the receiver (RX). The CSI samples are captured by the RX (step 1) at each time instance and then passed to a stream sampler (step 2). The stream sampler automatically determines whether the CSI sample should be retained or ignored. If the sample is retained, then the stream sampler places the sample within a buffer ($\mathcal{B}$) (step 3). Finally, the buffer can be used to train the machine learning model.

The following sections describe a number of possible stream samplers which can be used to make these two decisions in real-time.

Most Recent Lowest Loss (MRLL) [16]: When training the machine learning models, the loss ($\mathcal{L}$) captures the overall average error of the model given the available dataset. We use the binary cross-entropy loss function:

$$\mathcal{L}(y,\hat{y}) = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) + (1-y_{ij})\log(1-\hat{y}_{ij}), \quad (4)$$

where N is the number of samples in our dataset, C is the number of classes that our model is able to predict, y is the set of true-classes (target) for our model, y_i is the i-th element in y and $y_{ij} \in \{0,1\}$ is the value of the j-th class after being one-hot encoded. We can then say that $\hat{y}_{ij} \in [0,1]$ is the probability prediction from our model that the i-th sample is of class j.

In the most-recent lowest-loss (MRLL) method, we rank each individual incoming sample based on the per-sample loss which means that we can simplify Equation (4) down to a per-sample loss:

$$\mathcal{L}[t] = -\sum_{j=1}^{C} y_j\log(\hat{y}_j) + (1-y_j)\log(1-\hat{y}_j). \quad (5)$$

Assuming that $\mathcal{B}$ is sorted such that

$$\mathcal{L}[\mathcal{B}[i]] < \mathcal{L}[\mathcal{B}[i+1]],\ \forall i \in \{1,\ldots,|\mathcal{B}|-1\}, \quad (6)$$

then for each time-instance t, MRLL will ignore the sample if $\mathcal{L}[t] > \mathcal{L}[\mathcal{B}[|\mathcal{B}|]]$. Otherwise, if $\mathcal{L}[t] < \mathcal{L}[\mathcal{B}[|\mathcal{B}|]]$, then $\mathcal{B}[|\mathcal{B}|]$ is replaced with t and $\mathcal{B}$ is resorted. Due to $\mathcal{B}$ being presorted at the beginning of this step, this sorting operation has $O(|\mathcal{B}|)$ time-complexity.

The idea to select samples for model training with the lowest loss appears in [16] where the understanding is that samples with low-loss are less likely to be outliers which will only cause degradation in model accuracy.

Most Recent Highest Loss (MRHL) [18]: The most recent highest loss (MRHL) takes the approach where if $\mathcal{B}$ is sorted as shown in Equation (6), then if $\mathcal{L}[t] < \mathcal{L}[\mathcal{B}[1]]$, the sample is ignored. Otherwise, when $\mathcal{L}[t] > \mathcal{L}[\mathcal{B}[1]]$, then $\mathcal{B}[1]$ is replaced by t and is sorted. As with MRLL, the time complexity for this sorting operation is $O(|\mathcal{B}|)$. The intuition behind this method is that, if a sample has a high loss value, then the model is unable to successfully recognize the sample and thus, the model is able to gain novel knowledge from training on the sample.

Proposed Approach: Variable Low/High Loss (VLHL): We might notice that MRLL and MRHL have conflicting stories on why a highest-loss approach or a lowest-loss approach is the best option. We propose that a mixture of high-loss and low-loss samples should be retained for training the machine learning. Just as the previous methods suggest, retaining high-loss samples allows the model to learn new knowledge from samples it is unable to recognize while retaining low-loss samples ensure that the model does not lose the ability to understand samples that it was previously good at recognizing.

In our proposed approach, the Variable Low/High Loss (VLHL) sampler uses two separate buffers: $\mathcal{B}_{high-loss}$ and $\mathcal{B}_{low-loss}$ such that $|\mathcal{B}_{high-loss}| + |\mathcal{B}_{high-loss}| = |\mathcal{B}|$ and $|\mathcal{B}_{high-loss}| = \mathcal{B} * \mathcal{R}_{high}$ and $|\mathcal{B}_{low-loss}| = \mathcal{B} * (1 - \mathcal{R}_{high})$ where $\mathcal{R}_{high} \in (0,1)$ is the percentage of samples that should be allocated to $\mathcal{B}_{high-loss}$ and $(1 - \mathcal{R}_{high}) \in (0,1)$ is the percentage of samples that should be allocated to $\mathcal{B}_{low-loss}$.

Baseline: Random Sampler (Random) [17]: In addition to the MRLL and MRHL methods, we compare our proposed VLHL method to three additional baseline approaches. The first approach is to use a random sampler, where the sample at each time instance t is selected or ignored randomly and when selected, randomly replaces one of the previous elements in $\mathcal{B}$. If the method is unable to perform better than the random sampler, than the method is not improving the accuracy of the model, but is instead sabotaging the quality of the model.

Baseline: Rolling Window Method (Rolling): The rolling window method takes a naïve approach to selecting samples by selecting the most recent window $w = |\mathcal{B}|$ of samples at time t such that

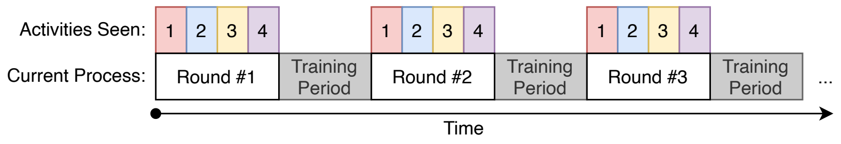

Figure 2: Simulation setting time-table. At each round, 4 distinct actions are seen by the sampler. At the end of each round, a training period of 10 epochs is given to train the local model based on the samples retained in $\mathcal{B}$ for the given round.

$\mathcal{B} = \{t - w + 1, \ldots, t - 1, t\}$. With this method, the assumption is that we can learn more information from the most recent samples rather than retaining old samples which may no longer be relevant to the machine learning model.

Baseline: Unconstrained Expanding Method (Expanding): The third baseline metric explored assumes that $|\mathcal{B}|$ expands to hold all samples such that at any given time t, $\mathcal{B} = \{1, 2, \ldots, t - 1, t\}$. In this approach, the device is assumed to be able to store an unlimited number of samples which is unrealistic for edge devices. Even so, this baseline metric allows us to understand what the accuracy of the model would be when all data samples are available and none of the important samples are missed.

3.2 Local Learning

After processing the sample stream using the described sampler methods, we have $\mathcal{B}$, $\tilde{X}$, and $\tilde{y}$. We use $\tilde{X}$ and $\tilde{y}$ to train the local-model on-device by minimizing the loss function described in Equation (4) as so:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{L}\left(\mathcal{F}_\theta\left(\tilde{X}_i\right), \tilde{y}_i\right), \tag{7}$$

where θ are the model weight parameters and $\mathcal{F}_\theta(\tilde{X}_i)$ is the predicted model output given input $\tilde{X}_i$ and model-parameters θ. The trained model can be used locally for on-device model inference, but can also be used to share knowledge to other devices in the network such as through federated learning.

4 EVALUATION

4.1 Dataset

In this work, we perform human activity detection experiments in 5 locations within a home environment using the dataset collected in [7]. For each location, we perform 4 HAD actions (sitting, sitting to standing, standing, standing to sitting) in a round-robin fashion 50 distinct times. The first set of 25 are used for training our model while the final 25 are used for evaluation.

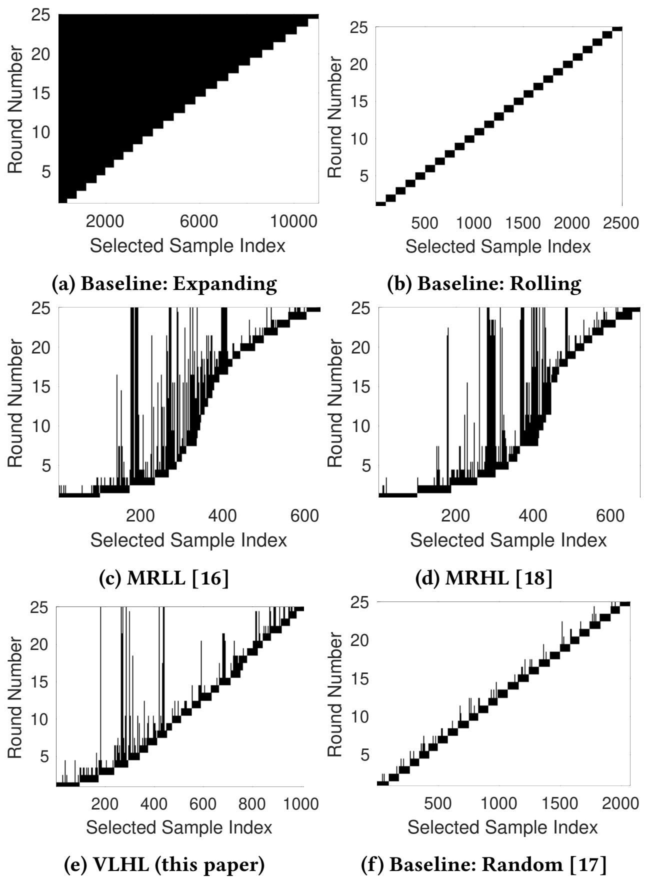

Figure 3: Samples selected for different methods. Black indicates that the sample was selected for the given round. Samples which are never selected by a given sampler are not given a sample index number.

4.2 Simulation Setting

To simulate a streaming design using the described dataset, we assume that one set of all 4 actions is made available each round and that each round appears at distinct times with an arbitrary amount of time between rounds. Training is performed over 25 rounds where the sampler is able to filter the dataset in real-time, but cannot perform any model training while receiving new samples during a round. Instead, model training occurs at the end of each physical round before the next round is performed. After each round, the model trains on the selected samples for 10 epochs. The diagram in Fig. 2 shows an illustrated example where during each round, actions 1, 2, 3 and 4 are seen by the sampler, then the round is followed by a training period where no new samples are collected from the environment. The goal is that during the data-collection round, the CSI-samples are naturally annotated within the environment during this period. We will note that naturally annotating the data stream in real-time is an important future work for allowing on-device training at the edge for all applications, but is out of scope for the work discussed here.

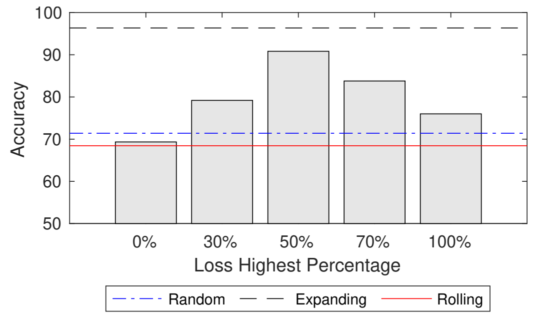

Figure 4: Accuracy per sampling method.

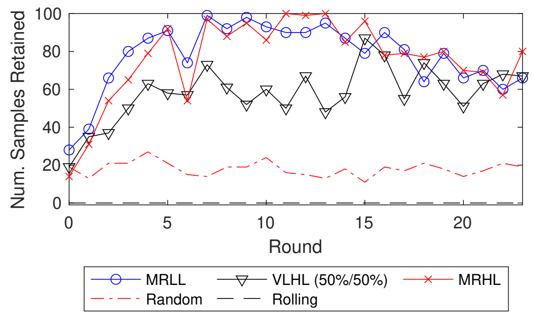

Figure 5: Number of samples retained per round.

4.3 Result of Sampling

For the six samplers evaluated in this work, Fig. 3 shows which samples are selected over the 25 rounds of training. We can see in Fig. 3a that the number of samples selected by the Expanding sampler increases over each round resulting in a total of 11, 035 samples selected by round 25. All other samplers retain $|\mathcal{B}| = 100$ samples per round. In Fig. 3b, the Rolling Window sampler does not retain any samples captured from previous rounds which means that the models are unable to train on any potentially important samples over more than a single round of training resulting in exactly 2, 500 unique samples being selected. The remaining four samplers; MRLL (Fig 3c), MRHL (Fig 3d), VLHL (Fig 3e), Random sampler (Fig 3f), each capture a varying number of unique samples. For example, the Random sampler captures 2, 068 unique samples which is almost equal to what the rolling window sampler captures but is much higher than the sample counts captured by MRLL (636 unique samples), MRHL (675 unique samples) and the VLHL (1, 008 unique samples) methods. From this view, we can see that both MRLL and MRHL capture the fewest number of unique samples which may suggest that the sampler gets stuck with the same samples over time, thus reducing the diversity of the samples available for training. The Random sampler, Rolling window sampler and VLHL samplers each form a straight line in Fig. 3 while MRLL and MRHL create an "S"-shaped curve. The more straight the line, the more often the sampler is selecting new samples while the more curved the line, the more often the sampler is opting to retain old samples rather than capturing new samples.

Now that we understand the differences in how the samplers behave, we can evaluate the accuracy of the models locally for each

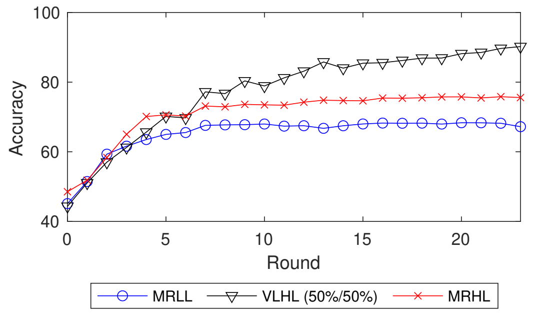

Figure 6: Accuracy over subsequent rounds.

location. Fig. 4 shows the accuracy using the different samplers where the bar graph show the accuracy with different values set for parameter $\mathcal{R}_{high}$ for the VLHL method from 0% (equivalent to MRLL) to 100% (equivalent to MRHL). Additionally, the line plots show the samplers that are not affected by $\mathcal{R}_{high}$. Out of these three line graphs, we can see that Rolling achieves the lowest accuracy of just 68.4% accuracy followed closely by the Random sampler with an accuracy of just 71.4%. We can see that the MRLL ($\mathcal{R}_{high} = 0\%$) only achieves 69.3% while MRHL ($\mathcal{R}_{high} = 100\%$) achieves 76.0% accuracy which is greater than random and rolling, but not by much. The Expanding sampler is able to store all samples at the end of each round and thus does not lose any potentially important samples due to sampling. As such, Expanding is the best-case scenario and can achieve an accuracy of 96.4%. Using VLHL with $\mathcal{R}_{high} = 50\%$, the accuracy achieved is 90.8% which is not as high as Expanding but does greatly improve the accuracy compared to each other limited-buffer sampler. This shows that balancing low-loss samples and high-loss samples does allow for an improved accuracy compared to just using low-loss or high-loss samples exclusively.

If we consider the number of retained samples per method in Fig. 5, we can see that MRLL and MRHL both retain far higher number of retained samples compared to VLHL. This directly translates into a lower accuracy because the model is not trained on a diverse set of samples over each round, but instead mostly the same samples are retained over subsequent rounds. In fact, we can see in Fig. 6 that MRLL and MRHL reach a plateau after approximately 5 rounds of sampling while VLHL continues to increase accuracy up until the final round. This corresponds directly with the fact that MRLL and MRHL are stuck with many of the same samples round after round as was illustrated in both Fig. 3 and Fig. 5.

At the end of each round, we train the model locally on the data in $\mathcal{B}$ for 10 epochs before moving on to the next round. A low value for epochs ensures that the model trains quickly and without consuming too much energy. In Fig. 7, we can see that at 10 epochs, the model trained with VLHL achieves 92.4% accuracy while 50 epochs achieves only a slightly larger accuracy of 93.5% while also increases the time to train and thus energy consumption. In this example, the highest accuracy (96.4%) is achieved when the number of epochs per round is set to 35. Furthermore, we can see that VLHL is able to achieve accuracy higher than both MRLL and MRHL except when the number of epochs per round is very low (i.e., 1 or 5) where VLHL is unable to surpass 70.0% accuracy. This shows that while we can achieve high accuracy by training for

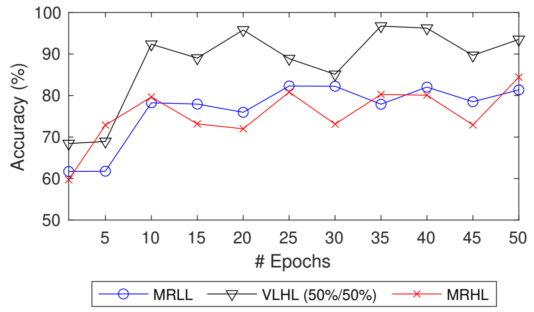

Figure 7: Effect of number of epochs per round on accuracy.

many epochs after each round, we can still achieve sufficient model accuracy with as low as 10 epochs of training per round.

While we consider very small buffer sizes in this work, (i.e., $|\mathcal{B}| = 100$), we find in Fig. 8 that VLHL still achieves greater accuracy compared to MRLL and MRHL up to $|\mathcal{B}| = 1,000$. As we would expect, as $|\mathcal{B}|$ increases, so does the accuracy, however we can see that while VLHL achieves 93.8% accuracy when $|\mathcal{B}| = 100$, the accuracy only increases up to 97.5% when $|\mathcal{B}| = 500$ while MRLL and MRHL achieve considerably lower accuracy of 64.0% and 76.3%, respectively when $|\mathcal{B}| = 100$ which can only increase to 94.7% (when $|\mathcal{B}| = 600$) and 95.4% (when $|\mathcal{B}| = 900$), respectively. This shows that VLHL still outperforms both MRLL and MRHL as the buffer size increases which ensures that our algorithm can be applied to edge devices with varying memory constraints.

5 CONCLUSION

In this work, we evaluated the use of online streaming sampling for training machine learning models on-board edge devices. We propose the use of Variable Low/High Loss (VLHL) which selects a balanced set of high loss and low loss samples to capture novel samples for training (i.e., high loss samples) while also retaining samples which are well understood and representative of the dataset (i.e., low loss samples) for the current environment. Overall, we find that VLHL achieves greater accuracy than baseline sampling algorithms as well as two state-of-the-art sampling algorithms: Most Recent Highest Loss (MRHL) and Most Recent Lowest Loss (MRLL). We find that while MRLL and MRHL get stuck with samples over time, VLHL seeks novelty from newer samples. By identifying this on-device sampling algorithm, we can capture useful training data for on-device model training while also upholding memory constraints that prevent all data from being buffered on low resourced and pervasive edge devices.

ACKNOWLEDGMENTS

This work is supported in part by National Science Foundation (NSF) Graduate Research Fellowship Program (GRFP) under Grant No. 1744624, and Commonwealth Cyber Initiative (CCI). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

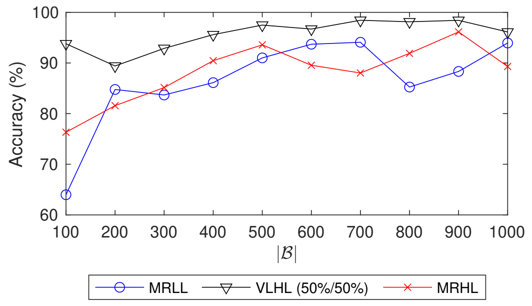

Figure 8: Effect of different internal buffer sizes ($|\mathcal{B}|$) on model accuracy.

REFERENCES

[1] Maria Francesca Alati, Giancarlo Fortino, Juan Morales, Jose M Cecilia, and Pietro Manzoni. 2022. Time series analysis for temperature forecasting using TinyML. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 691–694.
[2] Yunhao Bai and Xiaorui Wang. 2020. CARIN: Wireless CSI-based Driver Activity Recognition under the Interference of Passengers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (2020), 1–28.
[3] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning. *arXiv preprint arXiv:2007.11622* (2020).
[4] Lachit Dutta and Swapna Bharali. 2021. TinyML Meets IoT: A Comprehensive Survey. *Internet of Things* 16 (2021), 100461.
[5] Lingchao Guo, Zhaoming Lu, Shuang Zhou, Xiangming Wen, and Zhihong He. 2021. Emergency Semantic Feature Vector Extraction From WiFi Signals for In-Home Monitoring of Elderly. *IEEE Journal of Selected Topics in Signal Processing* 15, 6 (2021), 1423–1438.
[6] Steven M. Hernandez and Eyuphan Bulut. 2020. Lightweight and standalone IoT based WiFi sensing for active repositioning and mobility. In *2020 IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 277–286.
[7] Steven M. Hernandez and Eyuphan Bulut. 2021. WiFederated: Scalable WiFi Sensing using Edge Based Federated Learning. *Internet of Things Journal* (2021).
[8] Steven M Hernandez, Deniz Erdag, and Eyuphan Bulut. 2021. Towards Dense and Scalable Soil Sensing Through Low-Cost WiFi Sensing Networks. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, 549–556.
[9] Hongbo Jiang, Chao Cai, Xiaoqiang Ma, Yang Yang, and Jiangchuan Liu. 2018. Smart Home Based on WiFi Sensing: A Survey. *IEEE Access* 6 (2018), 13317–13325.
[10] Kavya Kopparapu and Eric Lin. 2021. TinyFedTL: Federated Transfer Learning on Tiny Devices. *arXiv preprint arXiv:2110.01107* (2021).
[11] Wamiq Raza, Anas Osman, Francesco Ferrini, and Francesco De Natale. 2021. Energy-Efficient Inference on the Edge Exploiting TinyML Capabilities for UAVs. *Drones* 5, 4 (2021), 127.
[12] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.
[13] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE.
[14] Yu Rong, Andrew Herschfelt, Jacob Holtom, and Daniel W. Bliss. 2021. Cardiac and Respiratory Sensing from a Hovering UAV Radar Platform. In *IEEE Statistical Signal Processing Workshop (SSP)*. IEEE.
[15] A. Navaas Roshan, B. Gokulapriyan, C. Siddarth, and Priyanka Kokil. 2021. Adaptive Traffic Control With TinyML. In *Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE.
[16] Vatsal Shah, Xiaoxia Wu, and Sujay Sanghavi. 2020. Choosing the sample with lowest loss makes sgd robust. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2120–2130.
[17] Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *ACM Trans. Math. Software* 11, 1 (1985), 37–57.
[18] Xiao Zeng, Ming Yan, and Mi Zhang. 2021. Mercury: Efficient On-Device Distributed DNN Training via Stochastic Importance Sampling. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. ACM.
[19] Taiyu Zhu, Lei Kuang, John Daniels, Pau Herrero, Kezhi Li, and Pantelis Georgiou. 2022. IoMT-Enabled Real-time Blood Glucose Prediction with Deep Learning and Edge Computing. *Internet of Things Journal* (2022).