On-Device Deep Learning for IoT-based Wireless Sensing Applications

Manoj Lenka and Ayon Chakraborty
SENSE LAB, Department of Computer Science and Engineering
IIT Madras

***Abstract*—Recent innovations in Wi-Fi sensing capitalizes on a host of powerful deep neural network architectures that make inferences based on minute spatio-temporal dynamics in the wireless channel. Many of such inference techniques being resource intensive, conventional wisdom recommends offloading them to the network edge for further processing. In this paper, we argue that edge based sensing is often not a viable option for many applications (cost, bandwidth, latency etc). Rather, we explore the paradigm of on-device Wi-Fi sensing where inference is carried out locally on resource constrained IoT platforms. We present extensive benchmark results characterizing the resource consumption (memory, energy) and the performance (accuracy, inference rate) of some typical sensing tasks. We propose `Wisdom`, a framework that, depending on capabilities of the hardware platform and application's requirements, can compress the inference model. Such context aware compression aims to improve the overall utility of the system - maximal inference performance at minimal resource costs. We demonstrate that models obtained using the `Wisdom` framework achieve higher utility compared to baseline models in more than 85% of cases.**

I. INTRODUCTION

Wi-Fi sensing has gained significant traction from the research community due to its ability to leverage existing wireless infrastructure as a sensing modality. The recent `IEEE 802.11bf` [1] amendment outlines sensing specific procedures and protocols in a WLAN setting, advocating for large scale adoption, standardization and interoperability among Wi-Fi devices doubling as 'wireless sensors'. This opens up new opportunities for IoT platforms to perform large scale wireless sensing, specifically leveraging Wi-Fi networks. Recent literature in this area have majorly focused on designing sophisticated inference models (e.g., utilizing deep neural networks) [2] to make Wi-Fi sensing robust and accurate. While such efforts have lead to several pioneering contributions in the Wi-Fi sensing landscape, a prominent research gap exists in realizing the systemic bottlenecks involved in translating such solutions to an IoT based ecosystem. In this paper, we highlight the key challenges associated with Wi-Fi sensing on resource constrained IoT devices and perform extensive benchmark experiments to understand the various system bottlenecks.

In a nutshell, Wi-Fi sensing leverages the multipath characteristics of the underlying wireless channel as a sensing metric. The Channel State Information (CSI) estimated on a Wi-Fi receiver captures such multipath effects that can be utilized to *learn* specific dynamics within a physical environment. In this paper, we consider Wi-Fi enabled IoT devices which have access to their respective CSI estimates, that can be further used to train sensing models and/or perform inference tasks. Such tasks are often claimed to be resource intensive and the conventional folk wisdom conveniently recommends offloading them off the device - for instance, to the network edge.

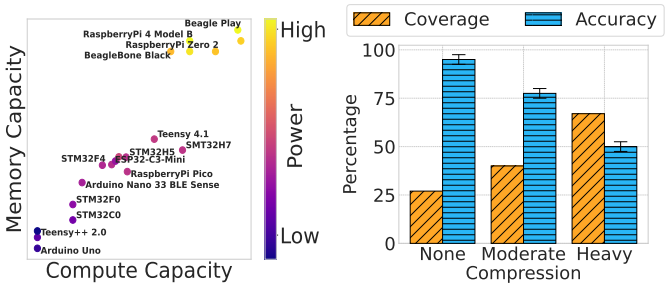


Fig. 1: Only 25% of the test devices are able to host the original model. Although model compression improves the deployablity/coverage, the model's accuracy takes a drastic hit, from 95% in the original model to ≈50% in the highly compressed version. Figure on the left shows increase in energy consumption with increase in resources

However, if wireless data communication and sensing need to co-exist, improving sensing at the cost of communication is clearly not a proposition that scales well. For instance, the wireless sensory data footprint takes a toll on the network performance, degrading QoS/QoE for regular data traffic. Similarly, deploying models for on-device inference tasks needs tailor made solutions that do not scale well, often affecting applications running locally on the same device.

In this paper, we extensively benchmark neural network models commonly used in wireless sensing applications and demonstrate how specific parameterization or compression of such models improves overall system performance. In particular, we consider architectures based on Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Fully Connected Network (FCN) and explore various compression strategies including *quantization*, *pruning*, *clustering* or their specific combinations. We highlight how specific strategies for compression impact various key performance metrics including inference accuracy, inferencing rate, energy consumption per inference and memory usage. We propose `Wisdom`[1], a framework that can cater to specific constraints related to a particular deployment and fine tune sensing models accordingly. `Wisdom` internally implements a *decision tree* based structure that recommends best effort compression strategies to meet such constraints. Our framework recommended models outperform vanilla compression strategies like weight quantization (often a de facto choice) in 85–95% of the cases.

[1]All data, traces, scripts related to the `Wisdom` framework are open sourced at https://cse.iitm.ac.in/~sense/wisdom/.

We make the following key contributions:

- We provide extensive benchmark results to demonstrate that no single sensing model exists that can suit heterogeneous IoT platforms with diverse performance requirements.
- We propose an automated framework, `Wisdom`, that can optimize a sensing model while satisfying a set of user provided constraints related to expected performance measures or hardware capabilities.

II. Wi-Fi Sensing Primer and Related Works

A. Wi-Fi Sensing and the CSI metric.

Wi-Fi sensing leverages from the phenomenon of multipath reflections within the wireless channel. When a modulated RF signal is transmitted, it not only reaches the receiver device along a direct path, but also gets reflected and scattered around by reflectors (obstacles) present in the environment before finally reaching the receiver at delayed intervals. Such delays in the time domain introduce distortions in the corresponding frequency response. Wi-Fi receivers estimate the Channel State Information (CSI) that captures such frequency response at the granularity of individual OFDM subcarriers present within its modulation bandwidth (e.g., 20 MHz or 40 MHz etc.). The CSI is estimated from the preamble symbols each time the a new data packet arrives at the receiver. Note that, such information is environmentally superimposed on the signal itself and is not affected by the actual data bits being communicated.

CSI Toolkits. A host of hardware-software solutions exist that make CSI available from specific Wi-Fi chipsets. Some of the foremost solutions include the Linux 802.11n CSI toolkit [3] (on Intel-5300 chipset), toolkits for the Qualcomm Atheros chipsets [4] followed by software defined radio based solutions like OpenWiFi [5] or Picoscenes [6]. Note that majority of the research on Wi-Fi sensing in the past decade was based on one of such tools that mandated heavy compute machinery to capture or process such data. Recently, the community have explored portable options like the Nexmon toolkit [7] for Broadcom chipsets (Raspberry Pi, Google Nexus smartphones) or the ESP-CSI toolkit [8] for the Esp32 based microcontrollers. Such recent developments have made it possible to experiment with CSI data available from low cost, microcontroller based systems resembling IoT devices.

Sensing with CSI Spectrograms. CSI captures the instantaneous state of the channel (coherence time), however, often the phenomena we want to sense span a duration of time that is several orders of magnitude more than the coherence time. Naturally, instead of analyzing such CSI vectors individually, a common practice is to look at an aggregation of such vectors obtained within a specific time window. Such time ordered and aggregated vectors represent a CSI spectrogram (see fig 4).

B. Compressing Neural Network Models

Following are the techniques we utilize for compressing the neural network models used for sensing.

Pruning [9]. Pruning is used to reduce the size and computational complexity of a neural network by identifying and removing irrelevant or low-magnitude weights, thus inducing sparsity in the model.

Weight Clustering [10]. Weight clustering involves grouping similar weights together into clusters and then representing all the weights in a cluster with a single centroid value. This reduces the number of unique weight values in a layer to a maximum of C, where C is the number of clusters. The centroids are estimated using clustering methods, such as K-means clustering.

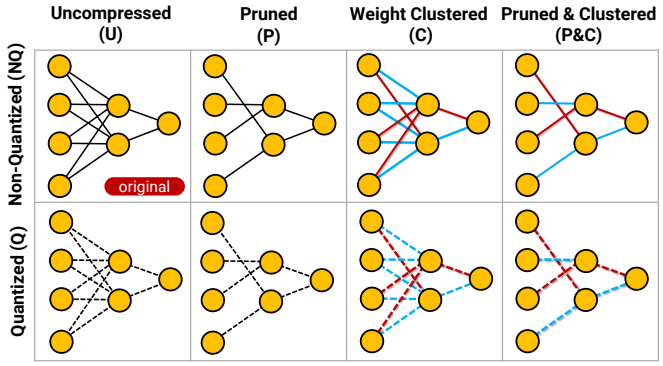


Fig. 2: Schematic of various model compression techniques. For *weight clustering*, only two clusters (colored red and blue) are used. The solid lines (top row) denote non-quantized weights while the dashed lines (bottom row) denote quantized weights.

Weight Quantization [11]. Quantization reduces the precision of the weights, for instance from single precision (4 byte `float`) to a single byte. This is achieved by creating a mapping between the real valued weights (r) and the quantized weight values (q). The discretization is done at a desired scale (S) with an origin or zero-point at Z. The mapping can be expressed as $r = S(q - Z)$. Note that quantization can occur (a) *post training*, i.e., the model is trained using floating point weights and the trained weights are then quantized, or, (b) the training can itself be *quantization aware*, i.e., quantized weight values are introduced during the forward propagation in neural network.

C. Existing Research Gaps in Related Literature

Existing literature on Wi-Fi sensing increasingly tends to explore deep learning models with the primary goal of improving inference accuracy. However, majority of such works do not highlight the feasibility or challenges related to deployment of such models on IoT class hardware. Second, often the CSI data considered in such works are obtained from well provisioned systems- for instance the Linux 802.11n CSI toolkit [3] or SDR based solutions like Picoscenes can generate upto 1000 CSI samples/sec, an order of magnitude more than what IoT class devices can barely support. Although such high data rates facilitate noise filtering or estimating Doppler shifts accurately etc., eventually leading to accurate modes, however, it is equally important to cope with *low fidelity* CSI data being processed on barely provisioned devices. Some initial works on model compression are reported in the work by Hernandez [12] [8], but it focuses on only model quantization aspects.

We resort to some recent works and platforms that enable performing neural network compute on embedded devices, particularly techniques for model compression [9]–[11]. Frameworks like Google's TensorFlow (TF) [13] along with tools like TFLITE or TFLITE-MICRO [14] provides hooks to make models optimized and lightweight for various target microcontroller architectures. Motivated by the existing research gap and equipped with the recent advances in embedded ML/DL frameworks, we move forward to investigate on-device inferencing for Wi-Fi sensing tasks.

III. WIRELESS SENSING TESTBED

To perform benchmark experiments we create a measurement setup to estimate various parameters related to the inference tasks, along with generating our dataset and models.

A. Measurement Setup

As shown in fig. 1, we perform preliminary experiments on a host of twenty test devices. While single board computers like Raspberry Pi or Beagle Boards prove to be over provisioned and have higher energy consumption, 8-bit microcontrollers (e.g., ATMEGA328P used in some Arduino devices) have insufficient resources to demonstrate any interesting cost-performance trade-off. For the various micro-benchmarks, we choose ESP32-C3-MINI that features a 160 MHz single core RISC-V processor with a 400 KB main memory (RAM) and a 4 MB on-chip flash memory. This device allows us a sweet spot to experiment with various cost-performance configurations and make observations representative of IoT class devices. Most importantly, ESP-32 has an integrated Wi-Fi chip that exports CSI making it convenient to build a complete on-device sensing application.

Energy Measurement. We advocate reporting energy consumption per inference. This helps us make a fair comparison across various models and compression techniques. We use a Nordic Semiconductor Power Profiler Kit II (PPK2) for such measurements. The PPK2 supplies a constant voltage of 5V to the device, and measures the current drawn. For relative comparison across inference models M_A and M_B, we take the ratio, $R_{AB} = \frac{i_A^2 T_A}{i_B^2 T_B}$, where i_A, i_B are the current draws and T_A, T_B are the inference times respectively.

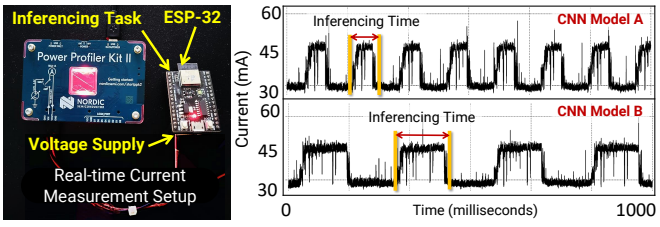


Fig. 3: The PPK2-based measurement setup shows a couple of sample current draw profiles for two different inference models. Note the difference in inference times.

Memory Usage. Memory availability directly modulates the *size* of the inference model that can be hosted. The device flash is a non-volatile memory that holds models and application programs. The RAM holds the runtime parameters such as inputs, outputs and the values of the intermediate layers.

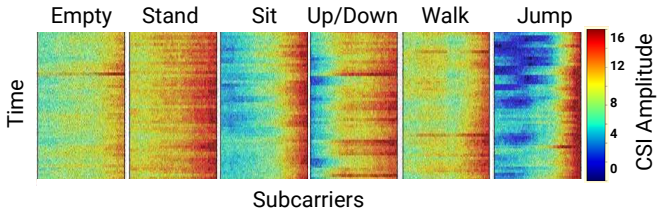


Fig. 4: Sample CSI spectrograms for the six activity classes in our dataset. Each spectrogram is a 100×48 dimensional real valued matrix

On-Device Deployment. We create various optimized versions of a given inference model by applying the respective compression techniques or their combination as discussed earlier. In order to deploy such models on a microcontroller based device (ESP-32), we use the Tensorflow [13] Model Optimization Toolkit and TFLITE-MIRO [14] that provides necessary hooks to achieve the same.

B. Target Application and Models

Dataset. We create an extensive dataset to test the performance of our models. The target wireless sensing application we choose is *Human Activity Recognition* (HAR). We consider six activity classes – two static: *stand* and *sit*, three dynamic: *sit up/down*, *jump*, *walk*, and a class indicating human absence. To make the dataset robust, we collect data at four different locations – two indoors and two outdoors – using five different human volunteers. For each location, every volunteer performs five different activities each for 30 seconds while the CSI was simultaneously recorded, roughly at 90–100 samples/sec. The experiments were repeated ten times, at different times of the day to reduce any bias. Overall, we record a rich dataset of ≈300K CSI samples spanning all the six classes, more or less uniformly. Each CSI sample contains 52 complex IQ components along the 52 OFDM subcarriers at 20 MHZ bandwidth. Out of 52, we only consider 48 data subcarriers leaving out the four pilot subcarriers. We use 100 CSI samples per spectrogram i.e., each spectrogram has a dimension of 100×48 amplitude values.

Neural Network Architectures. Existing neural network models for HAR (using Wi-Fi sensing) majorly comprise of CNN, RNN or FCN based architectures. For each architecture, we create nine models with increasing number of parameters, viz., 250, $1.5K$, $3K$, $6K$, $12K$, $24K$, $50K$, $90K$ and $180K$. Each model takes the 100×48 dimensional CSI spectrogram as input and predicts one of the six HAR classes. For the CNN, we use a RESNET-like structure where the number of residual blocks are increased to capture higher number of parameters. For FCN, we simply increase the number of layers in the network, as well as the number of neurons in each layer. We use a specific variant of RNN called Long Short Term Memory (LSTM), that helps in preserving long distance relationship within the samples. To increase the number of parameters we increase size of the weight matrix for the different gates in an LSTM cell, later we also stack multiple LSTM cells on top of each other. To avoid over-fitting, we use standard techniques like batch normalization and drop-out at each layer.

C. Relative Performance Benchmarks

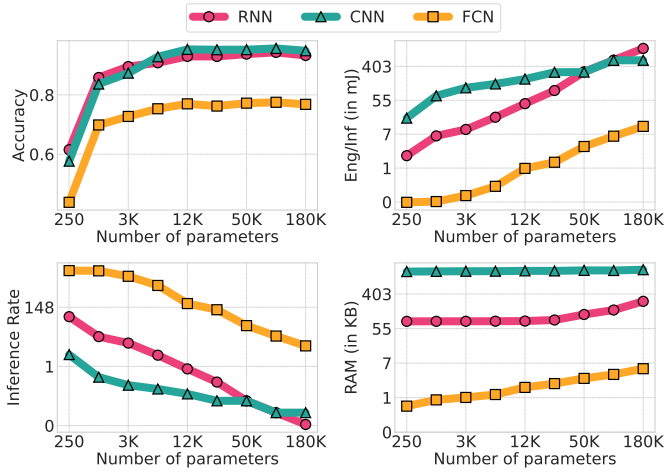


Fig. 5: Comparison of accuracy, energy per inference, inference rate, and runtime memory reserved for the different architectures as we increase the number of parameters. The models presented here are uncompressed.

Before we delve deeper into model compression in the next section, we showcase some macro performance results and discuss their implications. Fig. 5 demonstrates a comparison among the RNN, CNN and FCN derived models for all their parameterized versions. We focus on four key performance indicators – the accuracy obtained by the model, energy consumed per inference, inference rate and the amount of runtime memory (RAM) required. For CNN as well as RNN, their accuracy saturates to about 95%, beyond $12K$ parameters. FCN can only yield 80% accuracy even with judicious over-parameterization. However, CNNs are at least two orders of magnitude slower than their FCN counterparts. Being on the slower side, the energy consumed per inference is also relatively high for CNNs and RNNs. Another interesting observation is how CNN's runtime memory consumption is always much higher compared to the RNN and FCN counterparts. Fig. 5 implicitly indicates a room for trade-off with prioritizing one performance metric over the other. For instance, if accuracy is of the highest priority, CNNs are the way to go, while if it is higher inference rates or low energy consumption FCNs can be a good choice.

IV. Compression Benchmarks

In the previous section, we discuss general performance trends and the way it is affected by a model's architecture and parameter count. In the following, we re-examine these trends in the premise of model compression. In particular, we analyze, (i) sensing performance related measures, viz., accuracy and inference rate, (ii) cost measures, viz., energy and runtime/flash memory consumption.

Sensing Performance. In addition to *inference accuracy*, we also consider the *inference rate* as a performance metric that is crucial for real-time sensing systems.

Accuracy. Fig. 6 (left column) shows the impact of pruning, clustering and both combined on the classification accuracy when compared to an uncompressed version of the same

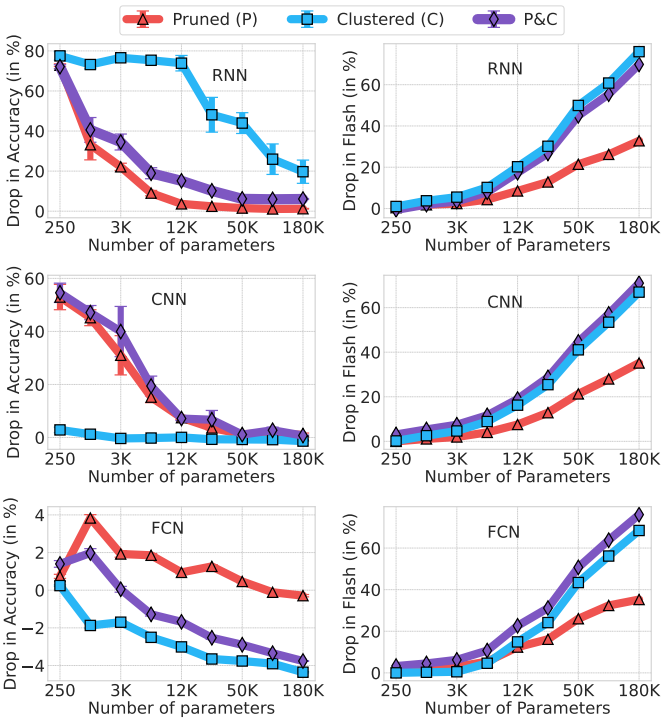


Fig. 6: Percentage decrease in accuracy and flash memory consumption due to *Clustering* (C), *Pruning* (P) and both (P&C) when compared to a uncompressed model. The left column shows decrease in accuracy, while the right one shows decrease in flash consumption.

model. Even after compression, CNNs continue to provide higher accuracy. Though uncompressed RNNs provided a reasonable middle ground between CNNs and FCNs, its accuracy is quite sensitive to model compression (see fig. 6 – top-left). While quantizing, for smaller models, the drop is not much as the absolute accuracy itself is poor. For moderate-sized models it impacts accuracy, before the models become over-parametrized and robust against quantization.

Inference Rate. Pruning and clustering have minimal effects on the execution time of the inference task, primarily because of the floating point operations. However, quantization bumps up the inference rate to as high as 30×, see fig. 7 (left). Quantization optimizes the model for integer arithmetic, hence offers a substantial benefit.

Cost Measures. We benchmark the system resource consumption and observe how it is impacted by various compression strategies.

Energy Consumption. We observe that energy consumption is heavily affected by the arithmetic type - integer versus floating point operations. Hence, in this case also (like inference rate), quantization provides the expected benefits. Fig. 7 (right) shows how the energy consumed per inference improves with quantization, particularly for models with high parameter count.

Flash Memory Consumption. As the parameter count increases, the accuracy figures remain robust in the face of compression. This is accompanied with significant savings in the flash memory consumption, so much so that the flash consumption figures become equivalent to models with lower parameter count (with lower accuracy) – refer fig. 6 (right

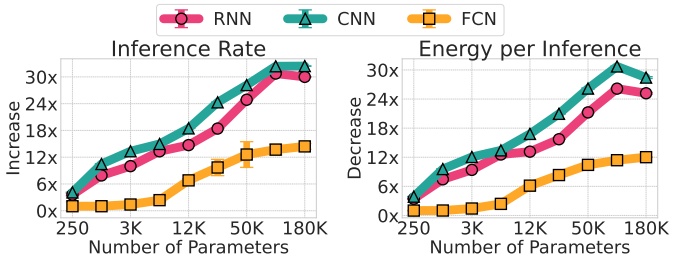


Fig. 7: Percentage decrease in inference rate and energy per inference of quantized models compared to their non-quantized counterparts for different model architectures

column). A rule of the thumb will be to choose a compressed model with a higher parameter count than an uncompressed model with a lower parameter count. Quantizing models that are already weight-clustered provides minimal improvements in saving flash. This is because clustering, in effect, has an approach that is similar to quantization in representing the weights, although the weights themselves are still `floats`.

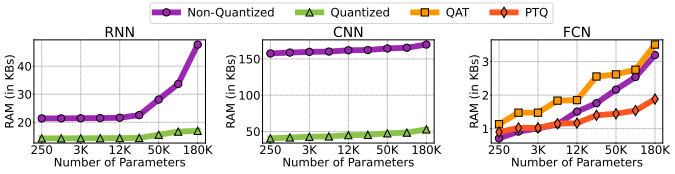


Fig. 8: Comparison of runtime memory usage for different architectures when quantized or otherwise. For CNN and RNN, QAT (quantization aware training) and PTQ (post training quantization) reserve the same amount of RAM, while for FCN they are different.

Runtime memory / RAM Usage. We show the impact on runtime memory consumption in fig. 8. For a CNN, although the RAM consumption does not change appreciably with the size of the model, i.e., parameter count, quantization can improve this by upto 72%. For, RNNs, quantized models do free up the RAM, but note that it affects accuracy non-trivially. Overall RAM usage for FCNs are much lower compared to CNNs/RNNs.

V. Wisdom Framework

The results above demonstrate a complicated interplay among various system parameters and its impact on the overall performance. This necessitates a unified framework that draws intelligent conclusions on such performance data and recommends a suitable model optimization strategy. We define `Wisdom`, a framework that can automate the process of picking the best compression strategy based on specific contexts, for instance, accuracy requirements, hardware constraints, latency constraints, energy goals etc. In short, `Wisdom` provides a tuning knob that allows users to specify relative priorities among such factors – e.g., lower energy usage over higher accuracy, or lower memory usage over higher accuracy and so on.

A. The Utility Metric

A system with more resources (higher cost) naturally offers a better performance and vice versa. The challenge is to find suitable a middle ground where neither the performance is critically affected due to lower costs, nor does the cost-performance trade-off result in diminishing returns. To capture this trade-off between cost and performance, we define a utility function $\mathcal{U}$ (eqn. 1) that we intend to maximize.

$$\mathcal{U}(\boldsymbol{i}) = \mathcal{P}(\boldsymbol{i}) - \mathcal{C}(\boldsymbol{i}) \tag{1}$$

In the above equation, we use $\boldsymbol{i}$ to encode details of the underlying model architecture (t) along with its parameter count (n) and compression technique (o) being used. The various choices for t, n and o, that we use in this work, are provided in eqn. 2.

$$\begin{aligned} \boldsymbol{i} \in \mathbf{I} = \{[t,n,o] | t \in \mathbf{T}, n \in \mathbf{N}, o \in \mathbf{O}\} \;\; where \\ \mathbf{T} = \{FCN,\, CNN,\, LSTM\} \\ \mathbf{N} = \{250,\, 1.5K,\, 3K,\, 6K,\, \ldots,\, 180K\} \\ \mathbf{O} = \{none,\, prune,\, cluster,\, qat,\, \ldots,\, pcptq\} \end{aligned} \tag{2}$$

$\mathcal{P}(\boldsymbol{i})$ denotes performance and $\mathcal{C}(\boldsymbol{i})$ denotes the cost of running an inference model with configuration $\boldsymbol{i}$. We define $\mathcal{P}$ (eqn. 3) as a weighted sum of accuracy ($\mathcal{A}$) and inference throughput ($\mathcal{I}$), where the respective weights w_{acc} and w_{inf} can be tuned by the user adhering to application requirements.

$$\begin{aligned} \mathcal{P}(\boldsymbol{i}) = w_{acc}\mathcal{A}(\boldsymbol{i}) + w_{inf}\mathcal{I}(\boldsymbol{i}) \\ where \qquad \mathcal{A} \geq A_{min} \qquad \mathcal{I} \geq I_{min} \end{aligned} \tag{3}$$

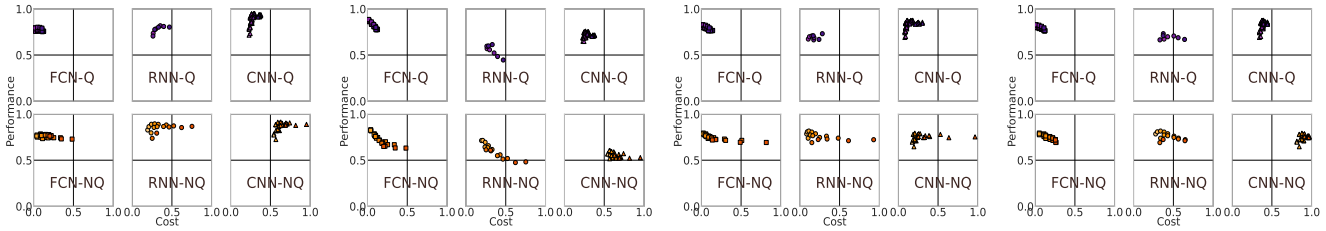
Similarly, we define the cost $\mathcal{C}(\boldsymbol{i})$ (eqn. 4) as the weighted sum of the energy per inference ($\mathcal{E}$), runtime memory requirements ($\mathcal{R}$) and flash memory ($\mathcal{F}$) consumed by the model configuration $\boldsymbol{i}$. The respective weights are denoted by w_{eng}, w_{ram} and w_{flh} respectively. Such weights directly determine the priority a user assigns to various system resources.

$$\begin{aligned} \mathcal{C}(\boldsymbol{i}) = w_{eng}\mathcal{E}(\boldsymbol{i}) + w_{ram}\mathcal{R}(\boldsymbol{i}) + w_{flh}\mathcal{F}(\boldsymbol{i}) \\ where \qquad \mathcal{E} \leq E_{max} \qquad \mathcal{R} \leq R_{max} \qquad \mathcal{F} \leq F_{max} \end{aligned} \tag{4}$$

Henceforth, we use a weight vector $\boldsymbol{w}$ to denote all the different weights $[w_{acc}, w_{inf}, w_{eng}, w_{ram}, w_{flh}]$. Also all the metrics are subject to constrains as defined in eqns. 3 and 4. For performance metrics $\mathcal{A}$ and $\mathcal{I}$, we define A_{min} and I_{min} as respective lower bounds. For cost metrics $\mathcal{E}, \mathcal{R}$ and $\mathcal{F}$, we define E_{max}, R_{max}, and F_{max} as upper bounds. $\boldsymbol{c}$ denotes the vector $[A_{min}, I_{min}, E_{max}, R_{max}, F_{max}]$ for all the constraints. Note that the metrics $\mathcal{A}, \mathcal{I}, \mathcal{E}, \mathcal{F}$ and $\mathcal{R}$ are normalized in the range [0,1] using min-max feature scaling. Similarly, the weight vector $\boldsymbol{w} \in [0,1]^5$. We also ensure that $w_{acc}+w_{inf} = 1$ for performance $\mathcal{P}$, and $w_{ram} + w_{flh} + w_{eng} = 1$ for cost $\mathcal{C}$. This leads both $\mathcal{P}$ and $\mathcal{C}$ to be in the range [0,1] and the utility metric $\mathcal{U}$ in the range [-1, 1].

B. Representative Scenarios

We present a few representative scenarios to demonstrate the impact of weight vector ($\boldsymbol{w}$) on the $\mathcal{C}$ and $\mathcal{P}$ metrics and how it modulates the choice of the model configuration.



(a) Scenario-1 **(b)** Scenario-2 **(c)** Scenario-3 **(d)** Scenario-4

Fig. 9: The cost-performance trade-off as exhibited by various model architectures in the four representative scenarios. For each scenario, a specific combination of weights are assigned to the vector, $[w_{acc}, w_{inf}, w_{eng}, w_{ram}, w_{flh}]$.

- Scenario-1 (**S1**): In this scenario, we assign a higher weight to accuracy ($w_{acc} = 0.9$) compared to the inference rate ($w_{inf} = 0.1$). The weights related to the cost metrics are assigned uniformly (w_{eng}, w_{ram} and w_{flh} are all assigned 0.33). From fig. 9a we observe that CNNs and RNNs (both quantized and non-quantized) exhibit better performance – higher accuracy with lower inference rate. However, FCNs are significantly cost efficient compared to CNNs/RNNs with a minor compromise in performance. CNNs/RNNs are typically energy hungry and memory intensive compared to FCNs.
- Scenario-2 (**S2**): In contrast to **S1**, in this case, we assign equal weights to accuracy and inference rate ($w_{acc} = w_{inf} = 0.5$). Weights related to the cost metrics remain the same, as in **S1**. FCNs perform significantly better providing reasonable accuracy (≈70-80%) and higher inference rate (by upto two orders of magnitude) (see fig. 9b).
- Scenario-3 (**S3**): In this case, we prioritize accuracy moderately higher than inference rate ($w_{acc} = 0.7$ and $w_{inf} = 0.3$). Regarding the cost metrics, primary importance is given to lower the flash memory consumption ($w_{flh} = 0.8$). Energy and runtime memory related metrics are assigned equal weights of 0.1. Although CNNs/RNNs are energy intensive and showcase a higher runtime memory footprint compared to FCNs, it has negligible effect on the cost due to the minimal weights assigned to such factors. It is important to note that the flash memory footprint for all the three architectures are roughly similar (depends on the number of parameters). We do not observe a prominent cost-performance trade-off in such scenarios.
- Scenario-4 (**S4**): In contrast to **S3**, we assign higher weights to runtime memory consumption and energy per inference ($w_{eng}, w_{ram} = 0.45$ and $w_{flh} = 0.1$). The performance related metrics are the same as in **S3**. Fig. 9d shows a clear trade-off in terms of model selection - the FCN exhibits the maximal performance at minimal cost.

The constrains used for all the above scenarios are as follows: $A_{min} = 0.70$, $I_{min} = 0.03$ Hz, $R_{max} = 200$ KB, $F_{max} = 2048$ KB and $E_{max} = 50$ mA (please note that these absolute values are normalized when actually applied for filtering). All the points in fig. 9 are of valid models that satisfy these constraints.

C. Model Selection using Decision Trees

The goal of `Wisdom` framework is to choose a model $\boldsymbol{i}$ that maximizes the utility metric $\mathcal{U}$, given weight vector $\boldsymbol{w}$ and a constraint vector $\boldsymbol{c}$. Overall we can summarize our objective as finding the optimal model $\boldsymbol{i}^*$ as defined below:

$$\begin{gathered} \boldsymbol{i}^* = \underset{i}{\operatorname{argmax}}\, \mathcal{U}(\texttt{Wisdom}(\boldsymbol{w}, \boldsymbol{c})) \\ where\ \texttt{Wisdom} : \{\boldsymbol{w}, \boldsymbol{c}\} \rightarrow \mathbf{I} \end{gathered} \tag{5}$$

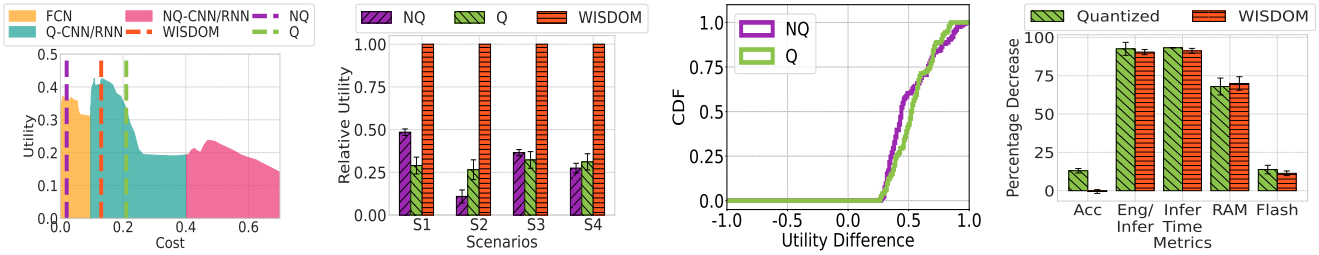
`Wisdom` uses a *decision tree* that takes as input $\boldsymbol{w}, \boldsymbol{c}$ and outputs $\boldsymbol{i}$. As $dim(\boldsymbol{i}) = 3$ denoting the model architecture t, number of parameters n and compression technique o as defined in eqn. 2, it therefore performs three classification tasks simultaneously. We decide to use a single decision tree with three outputs instead of three independent decision trees, because the outputs t, n and o are correlated to each other, e.g., certain compression techniques work better with certain architectures (refer sec. IV).

In order to train the decision tree we create a dataset with $\approx 27K$ different $\boldsymbol{w}$ and $\boldsymbol{c}$ configurations. For each $\boldsymbol{w}$, we calculate $\mathcal{U}$ for all the models we have trained (i.e., 3 architectures, 9 different parameter counts, and 12 compression techniques yielding a total of 324 models). We choose the one ($\boldsymbol{i}^*$) that has maximum utility while satisfying the constraints $\boldsymbol{c}$. $\boldsymbol{i}^*$ serves as our ground truth for a given weight vector $\boldsymbol{w}$. We handcraft 126 unique test cases covering different scenarios like the ones described in sec. V-B, and ensure that these are not part of training set. The trained decision tree has an accuracy of 97.61%, where accuracy denotes the fraction of cases the optimal model $\boldsymbol{i}^*$ is chosen.

D. Comparison with baseline models

We demonstrate the effectiveness of `Wisdom` in choosing the optimal model $\boldsymbol{i}^*$ by comparing utilities of the chosen model over a host of baselines.

Baseline Models. The baseline models span three architecture types (FCN, CNN and RNN). For each type, we consider three different parameter counts (≈1500, ≈6K and ≈24K) regulating the overall size of the model – small, moderate and large. This yields nine uncompressed model configurations (NQ). We also create a quantized counterpart of these nine models referred as Q. The baseline models are indicative of a naive (non-informed) choice.



(a) Utility ($\mathcal{U}$) vs Cost ($\mathcal{C}$) **(b)** Relative utility **(c)** CDF for difference in relative utilities **(d)** Comparing $\mathcal{A}, \mathcal{E}, 1/\mathcal{I}, \mathcal{R}$ and $\mathcal{F}$

Fig. 10: Fig. 10a shows the diminishing return of utility with increase in cost for a particular configuration of $\boldsymbol{w}$. Fig. 10b compares the utility of all NQ baseline model and all Q baseline model, and the model recommended by `Wisdom`. The utility is relative to the model with highest utility for that scenario i.e., $\boldsymbol{i}^*$. Here **S1, S2, S3** and **S4** denote different scenarios with different weights $\boldsymbol{w}$, as defined in sec. V-B. Fig. 10c shows CDF of the utility difference between `Wisdom` recommended model and utility values of all the Q and NQ baseline models for all test cases. Fig. 10d shows the percentage decrease in various cost/performance metrics of models chosen by `Wisdom` and compare it with Q baseline models. The percentage decrease is w.r.t NQ models.

As shown in fig. 10a, for scenario **S1**, increasing the cost increases the utility till a certain point after which it starts decreasing (diminishing returns). $\boldsymbol{i}^*$ is the model whose $\mathcal{C}$ corresponds to the peak of the plot i.e., maximum utility. The three dashed vertical lines correspond to the *best* quantized (Q), non-quantized (NQ) and `Wisdom` recommended models. Observe that, compared to the `Wisdom` recommended model, the best non-quantizd model (FCN) has lower cost as well as lower accuracy. Similarly, the best quantized model (CNN/RNN) has greater cost but lower accuracy.

Fig. 10b presents the (average) relative utility of the quantized, non-quantized and `Wisdom` recommended models. The relative utility is measured as $\frac{\mathcal{U}(\boldsymbol{i})}{\mathcal{U}(\boldsymbol{i}^*)}$. First, the `Wisdom` recommended models achieve a relative utility close to one. Second, in terms of relative utility, such models outperform the Q and NQ counterparts by 0.5 or more. In fig. 10c, we plot the empirical CDF for the difference in relative utility values between the `Wisdom` recommended models and the Q/NQ models respectively. For both cases, the median stands at ≈ 0.5 while the 75^{th} percentile is at ≈ 0.7. Overall, the `Wisdom` recommended model achieves a higher utility over the Q and NQ baseline models 85% and 99% of the test cases respectively. Further, we observe that models recommended by `Wisdom` on average consumes similar amount of resources compared to Q models, but have negligible decrease in accuracy. However, if the models are simply quantized, we observe a $\approx 15\%$ drop in accuracy on an average (see fig. 10d).

VI. Conclusions

Wireless sensing has gained significant traction, particularly with its proliferation among resource-constrained IoT devices. Traditionally, such wireless sensing applications assume an edge oriented architecture where the sensory information (CSI) is communicated to the edge for inference tasks. Such architecture has some apparent drawbacks related to sharing of the network bandwidth and edge processing costs. In this paper, we explore on-device wireless sensing that advocates hosting the inference models on the IoT device itself. We showcase the related challenges and demonstrate that a *one-size-fits-all* type of model is infeasible given the heterogeneity of IoT-class devices and diverse user requirements/priorities. We propose `Wisdom`, a framework that, depending on capabilities of the hardware platform (memory availability, energy constraints) and application's requirements (accuracy, inference rate), can compress the inference model. Such context aware compression aims to improve the overall utility of the system - maximal inference performance at minimal resource costs. We demonstrate that models obtained using the `Wisdom` framework achieve higher utility compared to baseline models in more than 85% of cases.

References

[1] E. Cianca *et al.*, "Radios as sensors," *IEEE IoT Journal*, 2016.
[2] Y. Ma *et al.*, "Wifi sensing with channel state information: A survey," *ACM Comp. Sur.*, 2019.
[3] D. Halperin *et al.*, "Tool release: Gathering 802.11 n traces with channel state information," *SIGCOMM Comput. Commun. Rev.*, 2011.
[4] Y. Xie *et al.*, "Precise power delay profiling with commodity wifi," in *ACM MobiCom*, 2015.
[5] X. Jiao *et al.*, "openwifi: a free and open-source ieee802. 11 sdr implementation on soc," in *IEEE VTC*, 2020.
[6] Z. Jiang *et al.*, "Eliminating the barriers: Demystifying wi-fi baseband design and introducing the picoscenes wi-fi sensing platform," *IEEE IoT Journal*, 2022.
[7] F. Gringoli *et al.*, "Free your csi: A channel state information extraction platform for modern wi-fi chipsets," in *WiNTECH*, 2019.
[8] Hernandez *et al.*, "Lightweight and standalone IoT based WiFi sensing for active repositioning and mobility," in *IEEE WoWMoM*, 2020.
[9] M. Zhu *et al.*, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv*, 2017.
[10] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv*, 2015.
[11] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE CVPR*, 2018.
[12] S. M. Hernandez *et al.*, "Wifi sensing on the edge: Signal processing techniques and challenges for real-world systems," *IEEE Communications Surveys & Tutorials*, 2022.
[13] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.
[14] R. David *et al.*, "Tensorflow lite micro: Embedded machine learning on tinyml systems," *arXiv*, 2020.