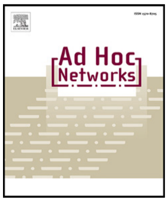
Ad Hoc Networks 178 (2025) 103915

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc



# WISDOM: A framework for scaling on-device Wi-Fi sensing solutions☆

Manoj Kumar Lenka, Ayon Chakraborty*

*Sensing and Networks Systems Engineering (SeNSE) Lab, Department of CSE, IIT Madras, Chennai 600036, India*

ARTICLE INFO

*Keywords:*
Wireless sensing
Wi-Fi
IoT
On-device inferencing
Model compression
TinyML
Utility maximization

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 (due to 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 that are just quantized for 83% of the cases. While for non-compressed models it has higher utility 99% of the time.

## 1. Introduction

Wi-Fi sensing has gained significant traction from the research community due to its versatility and ability to leverage existing wireless infrastructure itself as a sensing modality. In fact, the recent `IEEE 802.11bf` [1–3] amendment (2024) 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 inferencing models (e.g., utilizing deep neural networks) [4–7] to make Wi-Fi sensing robust and accurate, across a variety of application scenarios ranging from human sensing, activity recognition to healthcare monitoring. While such efforts have lead to several pioneering contributions disrupting 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 [8].

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 and provides a reasonable signature to *learn* specific dynamics of a physical environment (more on this later). In this paper, we consider a generic Wi-Fi network setting, where a number of IoT devices act as receivers and have access to their individual CSI estimates. Such CSI data is typically used to pre-train sensing models relevant to specific applications of interest [1,4] as well for inference tasks. These tasks are often resource intensive and the conventional folk wisdom conveniently recommends offloading them off the device — for instance, to the network edge [8–10]. A detailed background on wireless sensing is presented in Section 2.

### 1.1. Deploying wireless sensing models: Local vs. edge-based inference

As wireless sensing applications evolve to tackle increasingly complex tasks – such as human activity recognition, gesture detection, or environmental monitoring – they often rely on computationally intensive inference models, including transformer architectures and attention-based networks. These models offer higher accuracy and richer semantic interpretation but come with substantial memory and processing requirements that exceed the capabilities of typical IoT

☆ This document is an extension of the preliminary results published in: Manoj Lenka and Ayon Chakraborty, *"On-Device Deep Learning for IoT-based Wireless Sensing Applications"*, WiSense Workshop, IEEE PerCom 2024, DOI: https://doi.org/10.1109/PerComWorkshops59983.2024.10502448.

* Corresponding author.
*E-mail address:* ayon@cse.iitm.ac.in (A. Chakraborty).

https://doi.org/10.1016/j.adhoc.2025.103915
Received 31 January 2025; Received in revised form 28 April 2025; Accepted 14 May 2025
Available online 2 June 2025

devices. Consequently, local deployment of such models becomes infeasible, shifting the computational burden to more capable edge or cloud platforms. In the following sections, we examine both deployment alternatives and argue that while edge-based inference has its own merits, local on-device processing often emerges as a practical and context-aware solution for a wide range of wireless sensing applications.

■ **Edge-Based Inference.** Inference on edge-based platforms offers significant advantages in many contexts. It enables the use of more sophisticated models than those feasible on-device, while avoiding the high latencies and privacy risks associated with cloud offloading. Edge platforms also support rapid model updates and coordination across multiple devices, which are valuable for large-scale deployments. In well-connected environments with sufficient bandwidth and power resources-such as smart buildings or factory floors — edge inference can strike an effective balance between computational capability and responsiveness. Although suitable for certain contexts, we argue that edge-based inferencing is often not a viable option for many real-world wireless sensing applications, particularly those requiring low-latency, reliable decision-making in dynamic or resource-constrained environments. We highlight two representative scenarios where such limitations become especially evident.

First, consider an unmanned aerial vehicle (UAV) tasked with sensing-driven exploration, such as structural inspection or search-and-rescue operations [11]. These missions often occur in remote or infrastructure-poor environments—deep indoors, hilly terrains, or disaster-struck zones, where network connectivity is intermittent or unavailable. In such cases, continuous streaming of CSI or other sensory data to a remote edge server is infeasible. Even when a connection is available, the latency introduced by round-trip transmission to the edge can undermine real-time responsiveness, especially for control-loop tasks like obstacle avoidance or path replanning. Relying on edge inference in such scenarios jeopardizes the mission success and compromises autonomy. As a second example, in intelligent transportation systems, edge-based inferencing may not meet the latency demands of safety-critical applications. For example, drowsiness detection or real-time distraction monitoring in driver-assist systems must trigger alerts or control overrides within milliseconds to prevent accidents [12]. Routing such inference through the edge introduces unpredictable delays, and a missed inference window could lead to catastrophic outcomes. Here, even a marginal increase in accuracy afforded by larger models at the edge cannot justify the risk introduced by delayed decision-making. Overall, the following concerns challenge the practicality of edge-based sensing:

- **Network usage.** Streaming sensory data off-device consumes significant bandwidth and stresses shared wireless networks. In our testbed, with just five IoT nodes streaming CSI data at approximately 120 samples/s per node, we observe a 2× increase in average ping latency and a 1.5–2× reduction in average throughput across the network (see Fig. 1(c)). This impact becomes more pronounced with dense deployments or shared network infrastructure, where Quality of Experience (QoE) of network applications are impacted.
- **Inference latency.** The round-trip time incurred by transmitting data to the edge and awaiting inference results adds non-trivial delay, which is detrimental for time-sensitive applications such as emergency braking, fall detection, or intrusion response. These delays may range from 50–150 ms under ideal conditions, which can be unacceptable depending on the sensing task.
- **Operational expenditures (OpEx).** Continuous use of commercial edge computing services incurs both energy and monetary costs. Surveying five globally prominent edge platforms, we find that running continuous inference workloads — such as human activity recognition from CSI streams — can cost upwards of $50–$60 per month per node, assuming minimal pricing tiers. These recurring costs scale poorly for large sensor deployments and diminish the cost–benefit appeal of edge solutions.

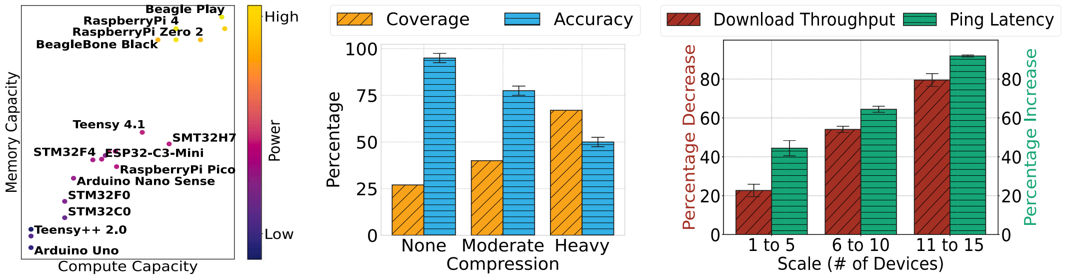
Finally, other factors including privacy implications of the sensory data or availability of the last-mile link itself, particularly in pervasive environments are critical in deciding off-device deployments.

■ **Local or On-device Inference.** Fig. 1(a) presents a diverse set of twenty commercially available, microcontroller-based (MCU) platforms that are representative of the compute and memory capabilities of commercial-off-the-shelf (COTS) IoT devices deployed in real-world settings. While edge or cloud-based inference offers scalability and model flexibility, it comes with network, latency, and operational cost trade-offs, as discussed earlier. In contrast, on-device inference enables immediate decision-making, preserves privacy, and eliminates network dependency. However, this approach is far from trivial due to inherent hardware constraints.

Recent research in Wi-Fi-based wireless sensing has proposed a variety of deep neural network architectures for tasks such as activity recognition, gesture classification, and occupancy estimation [13–15]. These models are typically evaluated under assumptions of abundant compute and memory resources. However, when attempting to deploy them directly on resource-constrained MCUs, a clear gap emerges. There is no *one-size-fits-all* solution for on-device inference in such settings. To illustrate this, we trained a state-of-the-art convolutional neural network (CNN) model [16] on CSI data for human activity classification and observed robust performance in simulation (accuracy ≥95%). Yet, deployment trials on our device set revealed that 75% (15 out of 20) of these platforms failed to even host the model due to insufficient memory — both in terms of runtime RAM and available flash for storing weights. This is unsurprising given that many of these devices are powered by 8-bit or 16-bit MCUs operating at clock frequencies in the tens of MHz range. Such platforms are fundamentally limited in their ability to support high-throughput inference or store models with millions of parameters, especially when represented in double-precision floating-point format. These limitations are exacerbated when trying to achieve real-time or near-real-time inference.

*Is vanilla model compression sufficient?* To mitigate these constraints, model compression techniques — such as quantization, pruning, and weight clustering — have been proposed in literature [17,18]. We experimented with forced compression of our baseline model to enable deployment on constrained platforms. As shown in Fig. 1(b), moderately compressed models could be deployed on approximately 50%–75% of the devices. However, this came at a significant cost to classification accuracy, often degrading performance by 10–20 percentage points. Beyond model size and inference latency, energy consumption remains a critical factor. IoT devices are typically battery-powered, and even a few additional millijoules per inference can impact device lifetime in always-on sensing scenarios. Further, the achievable inference rate – i.e., how frequently the device can process incoming CSI streams without stalling or missing data – is tightly coupled with both the model complexity and the microcontroller's processing budget [17,19,20].

Summarizing the above arguments, 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. Similarly, deploying models for on-device inferencing tasks needs tailor made solutions that do not scale well, often affecting applications running locally. We present extensive benchmark results to showcase the performance of three popular neural network architectures (commonly used in wireless sensing) and how specific parameterization or compression of the models impact the overall system performance. Namely, we look at 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 inferencing accuracy, inferencing rate, energy consumption per inference and memory



**(a)** Energy consumed by devices **(b)** Compression Vs Accuracy **(c)** Decrease in QoS

**Fig. 1.** Fig. 1(a) shows how increase in resources i.e., compute and memory leads to increase in energy consumption. Fig. 1(b) shows a specific sensing model tested on a collection of 20 test devices (shown in Fig. 1(a)). Only 25% of the test devices are able to host the original model. Although model compression improves the deployability or coverage, the model's accuracy takes a drastic hit, from 95% in the original model to ≈50% in the highly compressed version. Fig. 1(c) shows the decrease in download throughput and increase in ping latency as we increase the number of devices that are sending CSI data to access point. The percentage decrease/increase is w.r.t to the scenario when no devices are sending CSI data.

usage. We propose `WISDOM`, a framework that can cater to specific requirements/constraints of a deployment and finetune 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. We make the following key contributions:

- **System Deployment Perspective.** We present one of the first comprehensive studies that approach Wi-Fi sensing from a system deployment perspective. Rather than optimizing solely for classification accuracy using large-scale, over-parameterized models, we focus on deployability — identifying models that can operate on resource-constrained hardware while still achieving reasonably high accuracy (e.g., ≥95%) for real-world tasks such as activity recognition and occupancy estimation.
- **Dataset Contribution and Reproducibility.** We conduct extensive benchmarking across 20 commercially available IoT platforms, encompassing a range of microcontroller architectures and memory footprints. We observe that most wireless sensing models proposed in recent literature are infeasible to be deployed without modification, due to memory or execution-time constraints. Even after applying compression techniques, we observe a trade-off in model accuracy, highlighting the *lack of a one-size-fits-all solution* for heterogeneous deployment context. All our datasets, traces, scripts and models are open-sourced at https://sense.cse.iitm.ac.in/wisdom/ [21].
- **Optimal Inference Model Selection.** We design and implement an automated model optimization framework, `WISDOM`, that assists practitioners in selecting and customizing inference models based on user-defined constraints such as maximum model size, inference latency, or minimum accuracy targets. Given a set of deployment requirements, `WISDOM` returns a compressed and quantized model tailored to the target platform, achieving up to 20%–25% reduction in model size and 70%–80% reduction in inference latency, with bounded degradation in performance (accuracy degradation is ≤5%).

## 2. Wi-Fi sensing primer and related works

In the following we provide some preliminary background on the wireless sensory data, i.e., the above mentioned CSI metric. Second, we introduce state-of-the-art compression techniques that we use to optimize our inferencing models used for sensing. Third, we showcase the state-of-the-art literature on Wi-Fi sensing and demonstrate the research gaps that motivate our current direction of work.

### 2.1. 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 or shortest path, but also gets reflected and scattered around by reflectors present in the environment before finally reaching the receiver at delayed intervals. Such delays in the time domain typically introduce distortions that can be perceived in the corresponding frequency response. To improve communication efficiency, these distortions need to be estimated and corrected - a process known as channel equalization which is fundamental to a wireless receiver. For instance, Wi-Fi receivers estimate the Channel State Information (CSI) that represents the signal attenuation at the granularity of individual OFDM subcarrier frequencies present within its modulation bandwidth (e.g., 52 or 108 subcarriers for 20 MHz or 40 MHz bandwidths). While CSI helps a wireless device to adapt and modulate its communication parameters based on dynamic channel conditions, it also provides signatures capturing characteristics of the ambient environment or the dynamics of reflectors in its vicinity. The CSI is estimated from the preamble symbols each time the receiver receives a new data packet. Note that, such information is *environmentally superimposed* on the signal itself and is not affected by the actual data bits being communicated.

*CSI Toolkits.* Today, a host of hardware–software solutions make CSI available from Wi-Fi chipsets. One of the foremost solutions was based on the Intel-5300 chipset and the Linux 802.11n CSI toolkit [22]. This was followed by toolkits for the Qualcomm Atheros chipsets [23]. Solutions based on software defined radio continued to be developed that led to extensive frameworks like OpenWiFi [24] or Picoscenes [25]. 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 [26] that can extract CSI from some Broadcom chipsets (Raspberry Pi, Google Nexus smartphones). The Esp-8266 Wi-Fi chipset from Espressif Systems provide direct register access to read CSI through its platform native APIs. 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 (e.g., 1 s). Such time ordered and aggregated vectors represent a CSI spectrogram (see Fig. 4).

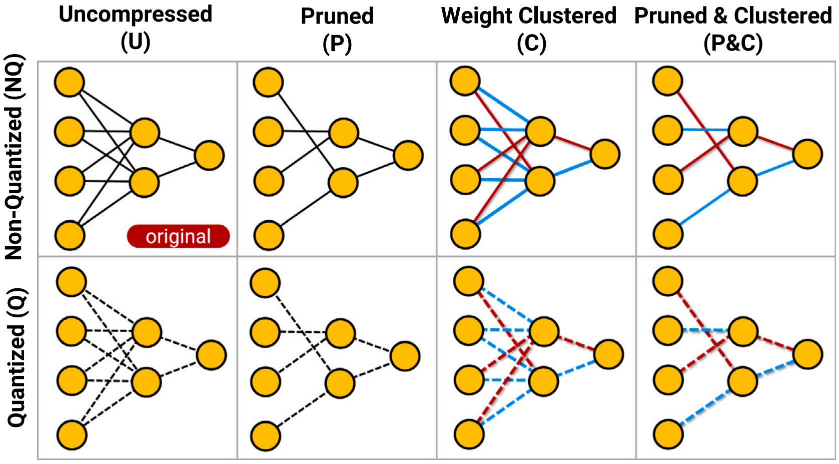


**Fig. 2.** Schematic of various model compression techniques. For *weight clustering*, only two clusters (colored red and blue) are used. The solid lines (top row) denote non-quantized weights while the dashed lines (bottom row) denote quantized weights.

### 2.2. IoT-izing models - compression of neural networks for edge or local inference

CSI spectrograms are typically used to train neural network models for various inferencing tasks. A key observation from the above line of works is that a majority of them have a single dimensional focus on improving the (classification) accuracy that result in heavy, over-parameterized models. In the following, we mention the state-of-the-art strategies [17,19,20] for compressing such models (see Fig. 2).

*Pruning.* [27] Pruning is a technique 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 [18].* 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 can be learned through various methods, such as K-means clustering. Note that both Pruning and Weight Clustering can be simultaneously applied to a network architecture.

*Weight Quantization [28].* 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 henceforth quantized, or, (b) the training can itself be *quantization aware*, i.e., quantized weights are introduced during forward propagation and is used to calculate the loss. Although for back propagation, floating point weights are used as usual. This keeps the function (neural network) continuous and allows us to calculate gradients for updating the weights. Model compression leads to several key advantages, including reducing the memory footprint of the model, speeding up inference time, and potentially improving the model's generalization ability. However, it is essential to strike a balance between choosing the right hyperparameters (e.g., pruning sparsity or number of clusters or the level of quantization etc.) and maintaining model performance.

### 2.3. Existing research gaps in related literature

As mentioned earlier, a host of existing literature on Wi-Fi sensing exclusively focus on building better deep learning models for the inferencing tasks. Some of the recent examples include El [7], DeepMV [13], DeepSeg [29], BiLSTM [14,30], DeepSense [31], Wisdar [32] for HAR; Widar 3.0 [5], WiHF [33], WiGRUNT [15], WiGR [34] for gesture recognition, and WiCount [6] for people counting. However, the above works prioritizes solely on the model's classification accuracy. Second, such works present results primarily from trace-based analysis and not much insights regarding the deployment of the proposed solutions in the upcoming IoT landscape are highlighted. Third, many such works use the CSI data of *highest fidelity*. For instance, SDR based toolkits (e.g., PicoScenes) or wireless NIC driver patches for Intel-5300 can provide CSI data even at 1000 samples/s, one order of magnitude more than what IoT class devices (e.g., ≤100) can barely support. Such high data rates are favorable for implementing efficient noise filter algorithms. Also, estimating *Doppler shifts* on CSI data becomes relatively straightforward in such cases. Our goal is primarily to cope with *low fidelity* CSI data being processed on barely provisioned devices.

*Systems Considerations.* With the growing interest in IoT based wireless sensing, some recent works [35,36] have indeed highlighted the necessity of on-device sensing both from a computation as well from a security/privacy viewpoint. Some initial works on model compression are reported in the work by Hernandez [35,37] focuses on only model quantization aspects. EfficientFi [36] demonstrates an actual system deployment for Wi-Fi sensing but restricted to the edge/cloud sensing paradigms.

*Embedded Machine Learning.* We resort to some recent works and platforms that enable performing neural network compute on embedded devices. A lot of efforts went in creating frameworks (e.g., TinyML) that implement ML stacks for resource constrained devices [18,27,28,38–40], particularly introducing techniques for model compression. Frameworks like Google's TensorFlow (TF) [41] along with tools like TFLite or TFLite-Micro [42] provides hooks to make models optimized and lightweight for various target microcontroller architectures [19,20]. 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.

*Data Compression.* Recent works have recognized the complexity associated with processing wireless sensing data and have proposed methods to compress such data — such as CSI spectrograms — prior to offloading it to Edge computing services [9,10,43]. While these techniques aim to reduce network transmission loads, many introduce their own significant computational overheads, making real-time application challenging on resource-constrained IoT devices. For instance, approaches based on AutoEncoder-driven compression [9,36] require non-trivial compute and memory resources, which are often unavailable on low-power embedded platforms. Simpler strategies such as spectrogram quantization are more feasible, however, deep neural compression, despite its effectiveness in minimizing data size, remains largely impractical

**Table 1**
Comparison of existing Wi-Fi sensing works highlighting system-level considerations.

| Work | Model complexity | Compression | Power analysis | Deployment | Scalability |
|---|---|---|---|---|---|
| | (High/Low) | (Data/Model) | (Yes/No) | (Yes/No) | (Yes/No) |
| El [7] | High | None | No | No | No |
| Deepsense [31] | High | None | No | No | No |
| Wisdar [32] | High | None | No | No | No |
| Widar 3.0 [5] | High | None | No | No | No |
| WiHF [33] | High | None | No | No | No |
| WiGRUNT [15] | High | None | No | No | No |
| WiGR [34] | High | None | No | No | No |
| WiCount [6] | Medium | None | No | No | No |
| CFNet [10] | Medium | Data (CSI) | No | No | No |
| RScNet [9] | Medium | Data (CSI) | No | No | No |
| CSI compression [43] | High | Data (CSI) | Yes | Yes | No |
| EfficientFi [36] | Medium | Data (CSI) | No | Yes | Yes |
| Edge sensing [35] | Medium | Model | No | Yes | Yes |
| **WiSDOM (Ours)** | **Low**[a] | **Data + Model** | **Yes** | **Yes** | **Yes** |

[a] WISDOM chooses the model with the least complexity while preserving sensing accuracy and respecting available hardware resources.

at the IoT scale (see, Section 4.1). In fact, the computational cost of applying deep compression on-device can, in some cases, approach that of running the full inference model locally—undermining the very purpose of offloading to the edge.

Overall, the existing body of work on Wi-Fi sensing predominantly focuses on maximizing inference accuracy through complex neural network models, often evaluated using ideal, high-fidelity CSI data collected with specialized hardware. Such studies, however, offer limited insights into the practical feasibility of deploying these solutions on constrained IoT devices in realistic scenarios. Additionally, prior work on embedded machine learning has demonstrated the potential for model compression and optimization, yet a systematic evaluation and deployment-focused analysis specifically targeting Wi-Fi sensing tasks remains largely unexplored. Table 1 summarizes a range of recent representative works. The top (blue-shaded) block lists efforts that primarily focus on improving sensing accuracy, while offering little to no consideration of practical *deployment challenges* such as device-level resource constraints, *power consumption* for battery-operated nodes, or overall *system scalability*. In contrast, the lower (green-shaded) block highlights more recent studies that begin to acknowledge systems-level bottlenecks in real-time wireless sensing deployments. [35] does not look into power consumption and uses a fixed sensing model. Our primary contribution goes beyond recognizing resource limitations: *We actively optimize the sensing models themselves based on the hardware and energy constraints of the target IoT platforms*.

## 3. Wireless sensing testbed

To perform system benchmark experiments we create an extensive measurement setup that enables us to estimate various system parameters related to the inferencing tasks.

### 3.1. Measurement setup

*Device Choice.* A few key factors modulate our choice for the test device. As shown in Fig. 1, we performed preliminary experiments on a host of twenty test devices. While single board computers like Raspberry Pi or Beagle Boards (various models) prove to be overprovisioned in some cases, 8-bit microcontrollers (e.g., ATMEGA328P used in some Arduino devices) have insufficient resources to demonstrate any interesting cost-performance tradeoff. 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 onchip flash memory. This device allows us a sweetspot 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.* Instead of reporting raw power consumption during inferencing, we advocate reporting the total 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) [44] for such measurements. The PPK2 supplies a constant voltage of 5 V to the device, and measures the current drawn in milliamperes scale with a nanoampere precision. For relative comparison across inferencing 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 inferencing times for the respective models. We use energy consumed per inference as one of our benchmark metrics.

*Memory Usage.* Memory availability is restrictive in low end microcontroller devices that directly modulates the *size* of the inferencing model that can be possibly hosted. The device flash (typically, few MBs) is a non-volatile memory that holds such models as well as application programs. The RAM (typically, few hundred KBs) holds the runtime parameters such as inputs, outputs and the values of the intermediate layers. ESP-32 used for benchmarking has 4 MB worth of flash and ≈250 KB worth of RAM remaining for running the application.

*On-Device Deployment.* We attempt to create various optimized versions of a given inferencing 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 [41] Model Optimization Toolkit, TFLITE-MIRO [42] that provides necessary hooks to achieve the same. Additionally, TFLITE-MIRO helps to preserve sparsity of the model as well as weight clusters when performing quantization aware training. It also preserves sparsity when performing weight clustering.

### 3.2. Target application and models

*Dataset.* We build an extensive dataset to test the performance of our models. The target wireless sensing application we choose is *Human Activity Recognition* (HAR), where the literature have effectively demonstrated its feasibility with a high degree of accuracy. The number of activities considered in HAR varies across works – roughly 4–6. For our dataset, we consider six activity classes – two static: standing and sitting, three dynamic: sitting up/down, jumping, walking, 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 (sixth indicates human absence) for a period of 30 s while the CSI was simultaneously recorded, roughly at 90–100 samples/s. The experiments were repeated ten times, at different times of the day to reduce any bias whatsoever. 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

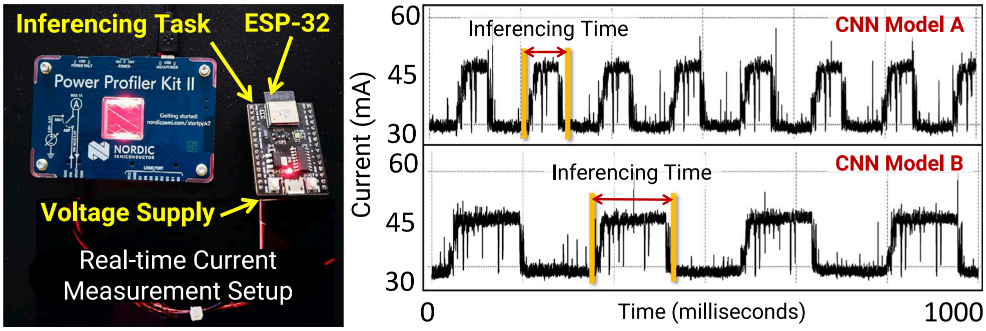


**Fig. 3.** The Ppk2-based measurement setup shows a couple of sample current draw profiles for two different inference models. Note the difference in inference times.

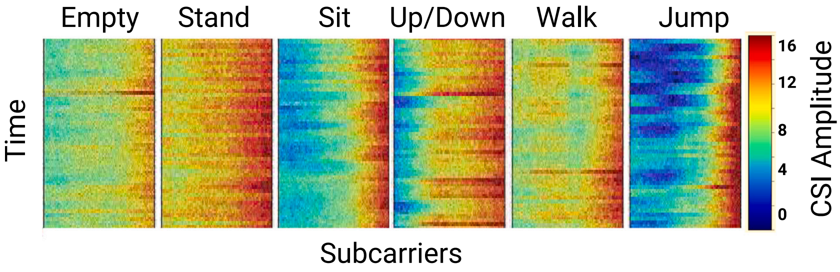


**Fig. 4.** Sample CSI spectrograms for the six activity classes in our dataset. Each spectrogram is a 100 × 48 dimensional real valued matrix.

IQ components (*inphase*-byte and *quadrature*-byte) along the 52 OFDM subcarriers at 20 MHz bandwidth. Out of 52, we only consider 48 data subcarriers leaving out the four pilot subcarriers. Our CSI spectrograms consist of 100 CSI samples, i.e., each spectrogram has a dimension of 100 × 48 amplitude values (norm of the complex IQ component).

*Neural Network Architectures.* Existing literature on Wi-Fi sensing based HAR majorly looks at variations of three different architectures - Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Fully Convolutional Networks (FCN). For each such architecture, we create nine models with increasing number of parameters starting from 250 for the simplest base model. The other eight versions have 1500, $3K$, $6K$, $12K$, $24K$, $50K$, $90K$ and $180K$ parameters. 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 [16] 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) [45], 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, which further increases the number of parameters. To avoid over-fitting, we use standard techniques like batch normalization [46] and drop-out [47] at each layer.

### 3.3. Relative performance benchmarks

Before we delve deeper into model compression and how performance is impacted by such strategies in the next section, here we showcase some macro performance results and discuss their implications. Fig. 5 demonstrates a comparison among the RNN, CNN and the FCN derived models for all their parameterized versions. We focus on four key performance indicators — the accuracy obtained by the model, energy consumed to do a single inference computation, inferencing rate and the amount of runtime memory (RAM) consumed when the model actively runs the inferencing tasks. 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. For instance, in Fig. 5 *(lower left)* for the model with $12K$ parameters, FCN has an inferencing rate of ≈150, i.e., about 7 ms per prediction while a CNN takes close to a second. 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 (reserved RAM) is always much higher compared to the RNN and FCN counterparts. Note that this is different from the model's actual size which is generally stored in the system's flash memory. 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.

## 4. Compression benchmarks

In Section 3.3, we present how various system performance metrics are impacted by the architecture chosen for the neural network model as well as its parameter count. It is evident that although over-parameterized models may boost accuracy, however, considering the cost to run them (energy, memory usage etc.) may lead to diminishing returns. In this section, we explore possible ways to compress our inference models and draw insights about the performance benefits (or losses) such compression entails.

### 4.1. Data compression or model compression?

Before attempting to move forward with model compression techniques, we discuss a popular alternative, i.e., compressing the sensory data itself. Note that the entries in our CSI spectrogram are real valued (32-bit `floats`). It is intuitive to think that compressing these entries itself may lead to lightweight models and also be less taxing on the CPU or memory consumption. We argue that this is always not the case.

*Data Quantization:* This method is quite trivial that truncates the given entries to a desired bit resolution, for instance to a single byte. Although this improves the data intake rate (up to 4×) or the runtime memory consumption (2–4×), it affects the classification accuracy non-trivially.

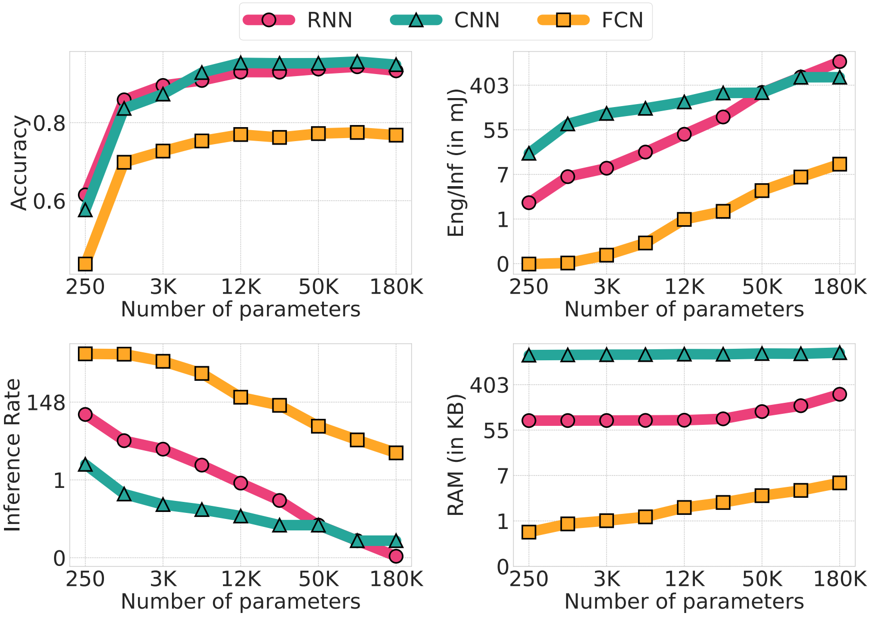


**Fig. 5.** Compares the change in accuracy, energy per inference, inference rate, and RAM reserved for the different architectures as we increase the number of parameters. The models are non-compressed.

Across various models, our consistent observation is that a brute-force bit truncation strategy is detrimental to the model's performance. For a byte-sized truncation the average accuracy across the models dropped by 15%–20%.

*Low Rank Approximation of CSI Spectrograms:* Instead of naively getting rid of the least significant bits as in the previous case, a more effective way is to quantize while preserving the general structure of the data. This is done using the *Singular Value Decomposition* (Svd) algorithm [48] that gives us a tuning knob to compress data according to a desired rank. Svd generates a list of singular values that encode the *full-rank* representation of our 100 × 48 dimensional spectrogram image [43]. Truncating the top-$K$ singular values generates a rank-$K$ approximation of the CSI spectrogram image, thereby intelligently compressing it. Although we found the inference models to work well even for $K$ as low as 8–10, we found the accuracy being sensitive to the value of $K$. This is due to the changing nature of the spectrogram structure. However, running SvD has a few limitations. First, we require a set (e.g., tens or hundreds) of CSI spectrograms to find its 'eigen structure'. This process is extremely memory intensive. Second, the computational time complexity of Svd is in the order of $\mathcal{O}(min(mn^2, m^2n))$, where $m$ is the cardinality of the set and $n$ is the dimension of the CSI spectrogram (i.e., $n = 100 \times 48$). Clearly, running Svd at frequent intervals is prohibitive and defeats the purpose of saving systems resources.

It is apparent that we vouch for model compression compared to data compression for IoT-based wireless sensing tasks. In the remaining part of this section, we primarily concentrate on the choice of the model architecture and relevant compression strategies that can make such models lightweight while retaining sensing accuracy.

### 4.2. Effects of model compression

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 reexamine 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, i.e., metrics related to resource consumption, viz., energy, runtime and flash memory consumption. We perform extensive benchmark experiments with the results presented in Figs. 6, 7, 8, 9.

For our experiments we tried different cluster sizes and sparsity levels (percentage of weights that are set to 0). Increase in cluster size or decrease in sparsity level improves accuracy of the models, but at the cost of higher resource consumption (memory and energy). Therefore, for all the benchmarks discussed below we chose a cluster size and sparsity level that provides the best cost-performance trade off, and avoids diminishing returns. We chose a cluster size of 8 and a sparsity level of 50%.

**Sensing Performance.** Typically, the general notion is to characterize such performance with accuracy, however, we feel that analyzing the inference rate (or the prediction latency) is crucial for a real-time sensing system.

*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 model. Even after compression, CNNs continue to provide higher accuracy and FCNs continue to provide quicker inference times or lower resource usage (albeit at the cost of accuracy). Another interesting observation is on RNNs. Though uncompressed RNNs provided a reasonable middle ground between CNNs and FCNs, its accuracy is quite sensitive to model compression, in particular pruning and clustering (see Fig. 6 *(top-left)*). On quantizing the models Fig. 7 *(left column)* shows the impact on accuracy. Initially, 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-parameterized 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. 8 *(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.* Energy consumption is heavily affected by the arithmetic type — integer vs. floating point operations. Hence, in this case also (like inference rate), quantization provides the expected benefits. Fig. 8 *(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 by a significant savings in the flash memory consumption, so much

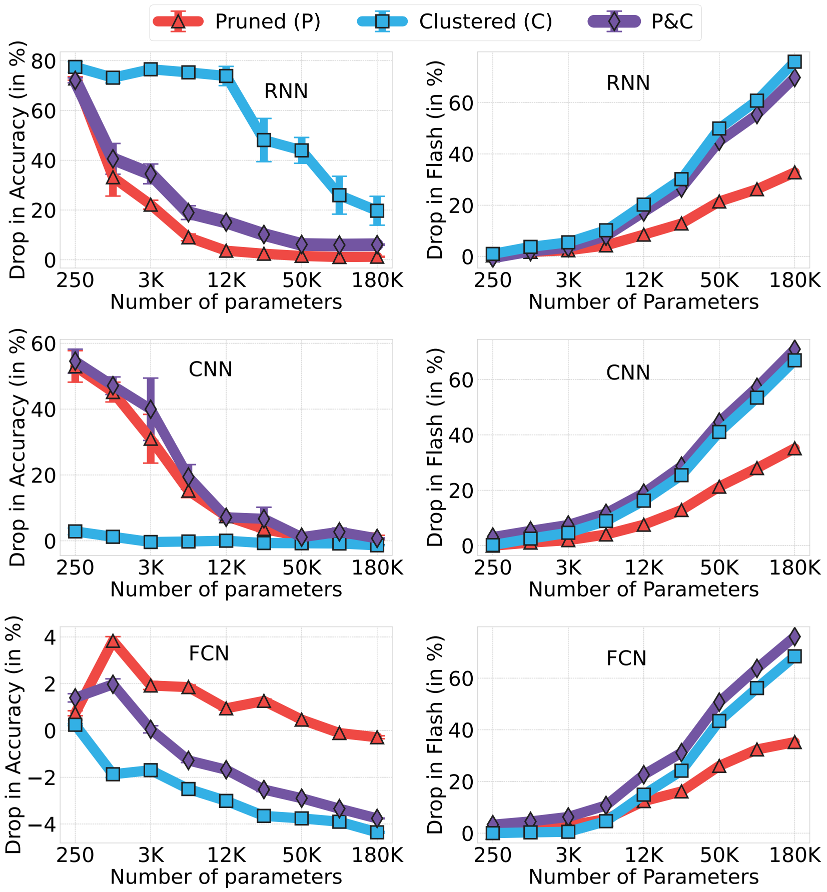


**Fig. 6.** Percentage decrease in accuracy and flash consumption due to *Clustering* (C), *Pruning* (P) and both (P&C) when compared to a uncompressed model. The left column shows decrease in accuracy, while the right one shows decrease in flash consumption.

so that the flash consumption figures become equivalent to models with lower parameter count (with lower accuracy). Hence, a best practice will be to choose a compressed model with a higher parameter count than an uncompressed model with a lower parameter count. See Figs. 6 and 7 *(right columns)*. Note, that quantizing models that are already weight-clustered provides minimal improvements in saving flash (see Fig. 7 *(right)*). This is because clustering, in effect, has an approach that is similar to quantization in representing the weights. For instance, using $K$-means to cluster weights will result in $K$ clusters, where $K$ is much lesser than the total number of weights (although the weights themselves are still `floats`).

*RAM Usage.* We show the impact on runtime memory consumption in Fig. 9. 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 up to 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. Further, post-training-quantization (PTQ) is slightly better that quantization-aware-training (QAT). This is not the case for CNNs/RNNs.

### 4.3. *Insights from empirical results*

In the following, we make some general observations from the above illustrated benchmark results.

- Although FCNs are generally less accurate when compared to CNNs/RNNs (≈10%–15% in our studies), however the trade-off is justified when factors like inference rate, runtime memory usage or energy footprint per inference is considered. For such factors, there is at least an order of magnitude improvement that is observed.
- It is recommended to use a compressed version of a larger model (higher number of parameters) than an uncompressed version of a smaller model itself. Although both have similar energy and memory footprint, the compressed model generally shows a higher accuracy (≈20%–40%) in our studies.
- RNNs are worst affected by compression. For instance, we observe an accuracy loss of up to 30% even for a large RNN model when compared with only 5% for larger CNNs/FCNs.
- Many low-end processors typical of IoT class devices are restricted to integer operations. In such scenarios, model quantization significantly affects critical performance metrics like runtime memory, inference rate and energy footprints. Energy per inference decreases while inference rate increases by 15–30×, wheres runtime memory consumption decreases by up to 70%.
- Clustering (and P&C) provides the most significant decrease in flash consumption (≈80% for large models). Though quantization on top of clustering does not further decrease flash consumption by much. Drop in flash consumption <5% for CNNs/RNNs and <20% for FCNs when a clustered model is further quantized. For other cases the drop in flash consumption after quantization goes up to 70%.

## 5. `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

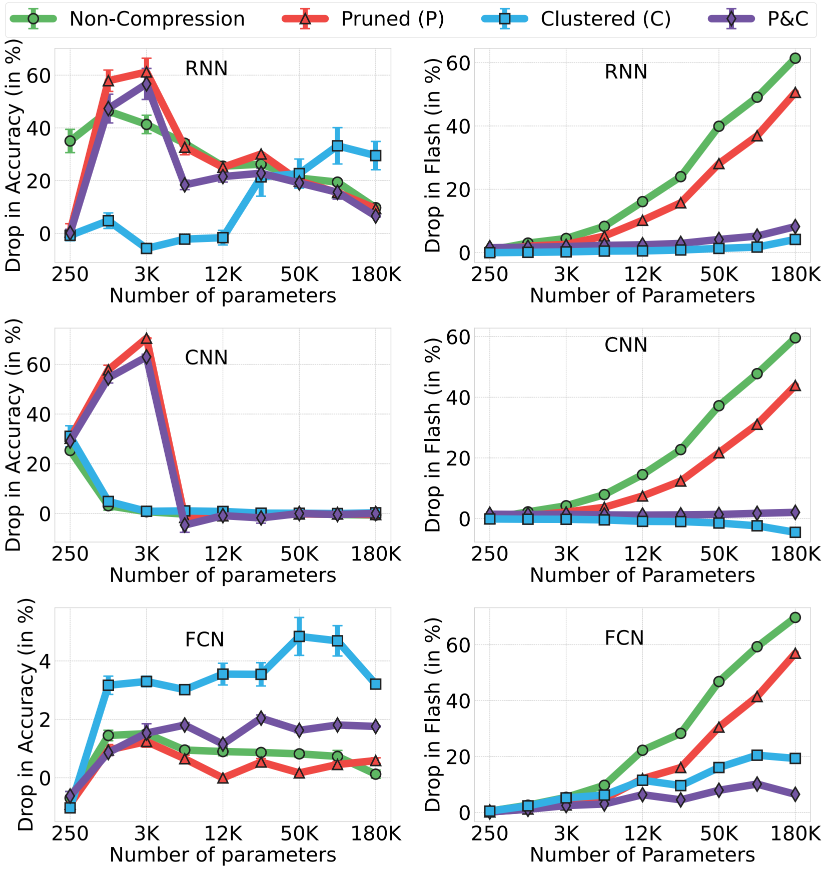


**Fig. 7.** Percentage decrease due to quantization when compared to their non-quantized counterparts. The left columns shows decrease in accuracy, while the right one shows decrease in flash consumption.

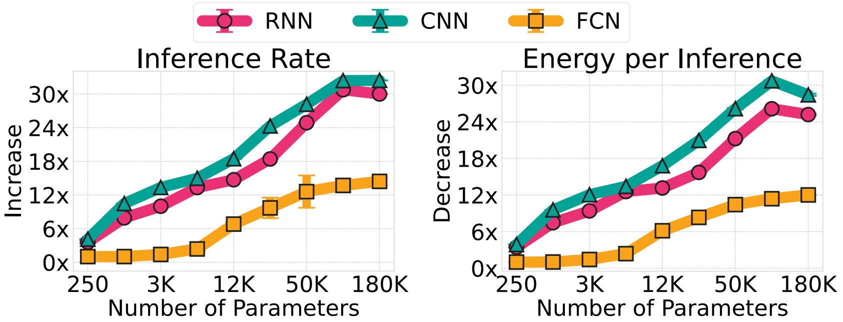


**Fig. 8.** Percentage decrease in inference throughput and energy per inference of quantized models compared to their non-quantized counterparts for different model architectures.

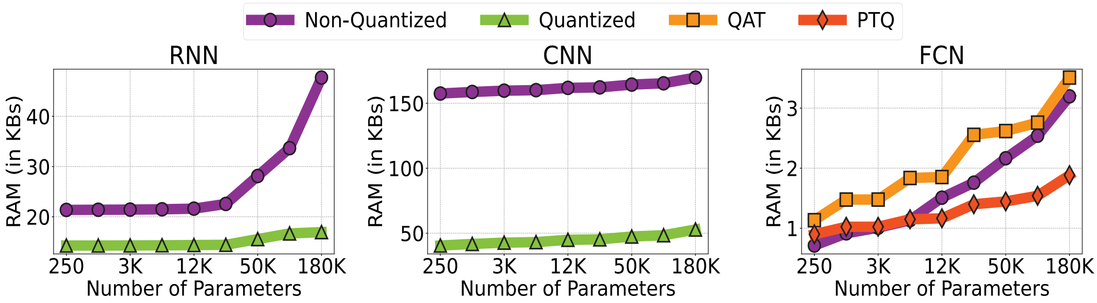


**Fig. 9.** Comparing RAM reserved for different architectures when quantized or otherwise. For CNN and RNN, quantization-aware-training (QAT) and post-training-quantization (PTQ) reserve the same amount of RAM, while for FCN they are different.

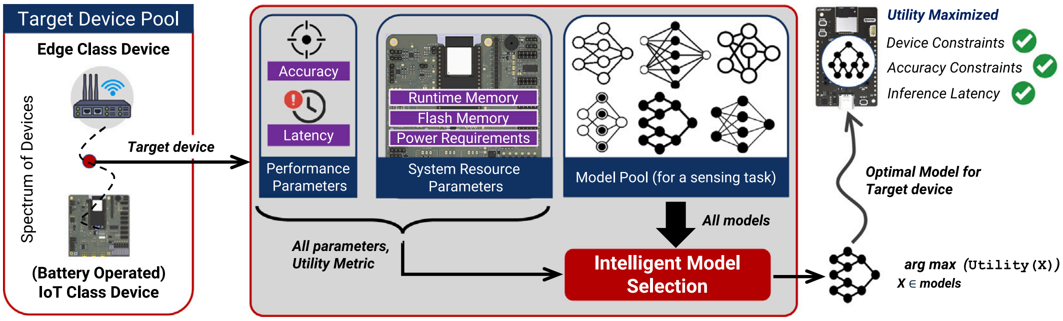


**Fig. 10.** Schematic diagram of the WISDOM framework, implemented as a complete end-to-end system in our testbed. Target devices span from highly resource-constrained IoT platforms to moderately powered edge-class nodes. Operating within a serverless ecosystem, each compute node requires an optimized model to function efficiently.

contexts, for instance, accuracy requirements, hardware constraints, latency constraints, energy goals etc. Fig. 10 presents a schematic overview of the WISDOM framework. In brief, once the target device is selected, essential system parameters are evaluated, including available runtime memory (RAM), flash storage capacity for the model, CPU clock speed, and power consumption characteristics (e.g., see Fig. 3). Additionally, the user specifies two key performance constraints: the desired model accuracy and the acceptable inference latency.

Given that complex models often achieve higher accuracy at the cost of increased latency, energy consumption, and memory footprint, we define a utility metric (Section 5.1) that balances performance goals against system constraints. To optimize this trade-off, WISDOM employs a decision-tree based search (Section 5.3) to select the best-suited model architecture that maximizes the utility. Importantly, this model selection occurs offline during the deployment phase. By enabling users to specify system priorities and by automatically adapting model choices to device capabilities, WISDOM facilitates scalable wireless sensing across a heterogeneous landscape of IoT and edge platforms.

### 5.1. The utility metric

A system with more resources (higher cost) naturally offers a better performance, similarly system with less resources (low cost) perform worse. The challenge is to find suitable a middle ground where neither the performance is critically affected due to lower costs, nor does increasing the cost result in diminishing returns. To capture this trade-off between cost and performance, we define a utility function $\mathcal{U}$ (Eq. (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 Eq. (2).

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

$\mathcal{P}(\boldsymbol{i})$ denotes performance and $\mathcal{C}(\boldsymbol{i})$ denotes the cost of running an inference model with configuration $\boldsymbol{i}$. We define $\mathcal{P}$ (Eq. (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 $\mathcal{C}(\boldsymbol{i})$ (Eq. (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.

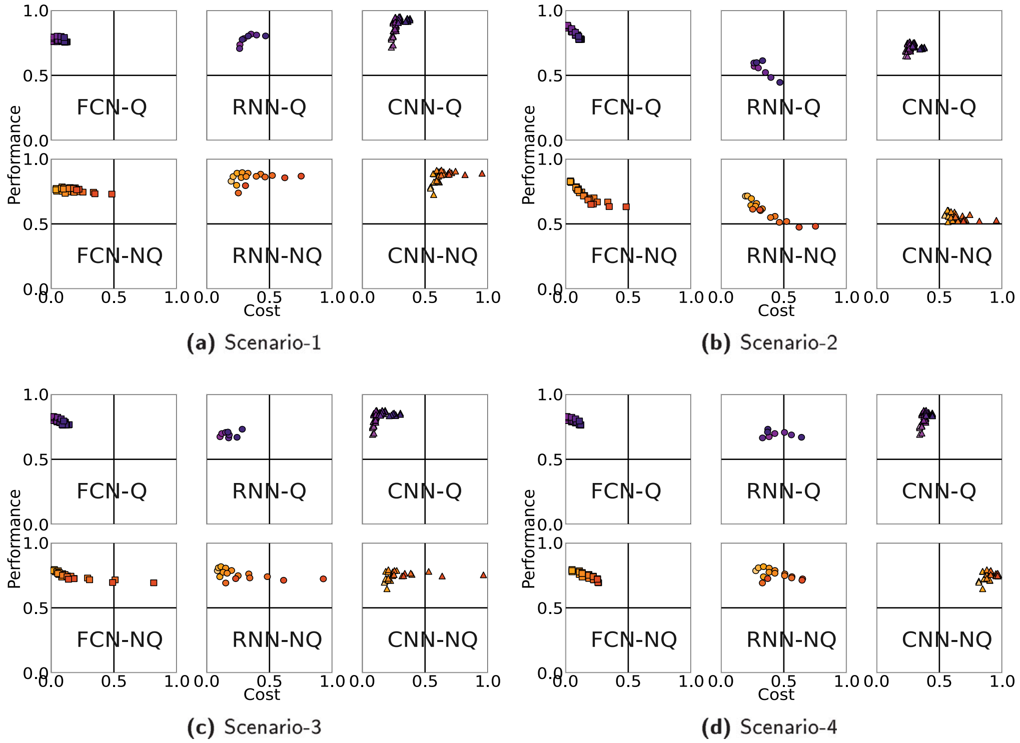
$$\begin{aligned} \mathcal{C}(\boldsymbol{i}) = w_{eng}\mathcal{E}(\boldsymbol{i}) + w_{ram}\mathcal{R}(\boldsymbol{i}) + w_{flh}\mathcal{F}(\boldsymbol{i}) \\ where \qquad \mathcal{E} \leq E_{max} \qquad \mathcal{R} \leq R_{max} \qquad \mathcal{F} \leq F_{max} \end{aligned} \tag{4}$$

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 Eqs. (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].

### 5.2. 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.

- 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. 11(a) 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 up to two orders of magnitude) (see Fig. 11(b)).
- 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.



**(a)** Scenario-1

**(b)** Scenario-2

**(c)** Scenario-3

**(d)** Scenario-4

**Fig. 11.** 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-4 (**S4**): In contrast to **S3**, we assign higher weights to runtime memory consumption and energy per inference ($weng$, $w_{ram} = 0.45$ and $w_{flh} = 0.1$). The performance related metrics are the same as in **S3**. Fig. 11(d) 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. 11 are of valid models that satisfy these constraints.

### 5.3. 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:

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

WISDOM uses a *decision tree* that takes as input $\boldsymbol{w}, \boldsymbol{c}$ and outputs $\boldsymbol{i}$. As the length of $\boldsymbol{i}$ is 3, denoting the model architecture $t$, number of parameters $n$ and compression technique $o$ as defined in Eq. (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 Section 4).

In order to train the decision tree we create a dataset with ≈27$K$ 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 Section 5.2, 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.

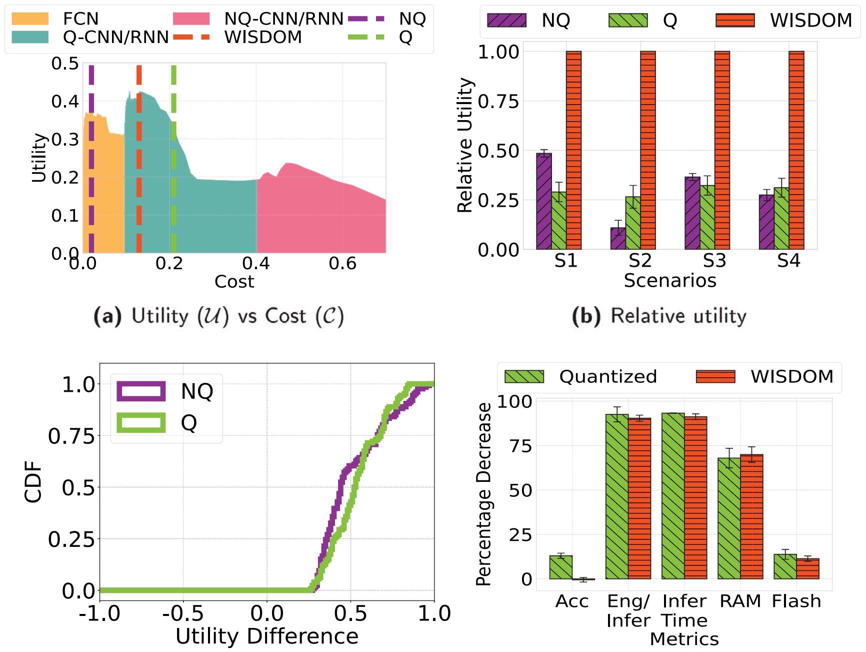
### 5.4. Comparison with baseline models

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, ≈6 K and ≈24 K) 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.

As shown in Fig. 12(a), 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 $C$ 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-quantized model (FCN) has lower cost as well as lower accuracy. Similarly, the best quantized model (CNN/RNN) has greater cost but lower utility.

Fig. 12(b) presents the (average) relative utility of the quantized, non-quantized and WISDOM recommended models. The relative utility is measured as $\frac{\mathcal{U}(i)}{\mathcal{U}(i^*)}$. 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. 12(c), we plot the empirical CDF for the difference in relative utility values between the WISDOM recommended models and the



**(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. 12.** Fig. 12(a) shows the diminishing return of utility with increase in cost for a particular configuration of $\boldsymbol{w}$. Fig. 12(b) 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 Section 5.2. Fig. 12(c) 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. 12(d) 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.

Q/NQ models respectively. For both cases, the median stands at ≈0.5 while the 75th percentile is at ≈0.7. Overall, the WISDOM recommended model achieves a higher utility over the Q and NQ baseline models for 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 ≈15% drop in accuracy on an average (see Fig. 12(d)).

## 6. Discussion and insights

In this section we discuss some key takeaway points based on the performance of WISDOM. First, majority of the models suggested by WISDOM are large (more number of parameters) with some form of compression rather than smaller (less number of parameters) models. This is in agreement with our benchmark results where larger models that are compressed have better accuracy than smaller models, while having a similar resource footprint. Second, almost all models suggested by WISDOM perform quantization. This follows from our benchmark results where quantization leads to a significant decrease in runtime memory and energy consumption, while increasing inference rate. Third, RNNs are rarely suggested by WISDOM, and also when RNNs are chosen, they are not compressed. We observe that RNNs are very susceptible to compression that causes a significant drop in accuracy. Fourth, CNNs are often suggested by WISDOM when $w_{acc}$ is high (more priority is given to model accuracy), while in other cases FCNs are suggested. This follows from our benchmark results, where we observe that CNNs have higher accuracy than FCNs, though it also consumes more resources. Finally, since clustering and pruning combined with clustering (P&C) show a significant decrease in flash consumption (Fig. 6 – right) compared to only pruning. Clustering (or P&C) is often suggested by WISDOM along with quantization, especially if $w_{flh}$ is high (more priority to lower flash consumption). Quantization on top of clustering does not significantly decrease flash consumption, but it provides huge savings to runtime memory and energy consumption, it also improves inference rate. Therefore, clustering and quantization were often used together in models suggested by WISDOM.

The above discussion reflects the similarities between the models suggested by WISDOM and our key observations from the benchmarking experiments. This further solidifies the effectiveness of our framework.

In this work we primarily focus on on-device computation in contrast to edge based computation for CSI based sensing applications. There may be a middle ground where we split the deep learning model [49], and run it on different operators in the computing continuum i.e., on-device, access point, gateways, micro-datacenters, cloud servers, etc. Depending on the user requirements (accuracy, latency, etc.) and budget (Opex, energy, etc.) how much should the learning model be split, and on which operators it should run on changes. This creates a very general and interesting optimization problem. We believe a lot more benchmarking is required in order to understand such systems. This would enable us to make intelligent deployment decisions, which would increase the adaptation of Wi-Fi sensing in the real world.

## 7. Conclusion

In this work, we address these challenges through the design of WISDOM, a framework that systematically recommends an optimized inference model — defined by its architecture, parameter count, and compression techniques — based on specific deployment requirements and device constraints. To this end, we make the following key contributions. First, we present one of the first comprehensive studies that approach Wi-Fi sensing from a *system deployment perspective*. Rather than pursuing higher classification accuracy alone, we prioritize model deployability on constrained hardware, demonstrating that real-world performance requires a balance between accuracy, resource footprint, and inference rate. Second, we conduct extensive benchmarking across 20 commercially available IoT platforms, revealing that most state-of-the-art sensing models are infeasible to deploy without significant modification. To support reproducibility and further research, we open-source our datasets, benchmarking scripts, and models at https://sense.cse.iitm.ac.in/wisdom/ [21]. Third, we develop an automated model optimization framework, WISDOM, that selects compressed and quantized models tailored to user-defined constraints, achieving up to 20%–25% reduction in model size and 70%–80% reduction in inference latency, while maintaining bounded accuracy degradation (≤5%).

We thoroughly benchmark candidate models by measuring their inference accuracy, inference rate, energy per inference, RAM, and flash memory consumption across our testbed. Experimental evaluation shows that WISDOM outperforms naive baselines: around 83% of the models recommended by WISDOM achieve higher utility compared to the best quantized models, and in 99% of cases outperform the best non-quantized models. These results demonstrate that simplistic strategies like pure quantization or uncompressed small models are insufficient; effective model selection and compression must be jointly optimized, guided by application-specific constraints and system goals. We believe WISDOM represents an important step toward sustainable and scalable wireless sensing deployments for the emerging IoT landscape.

## CRediT authorship contribution statement

**Manoj Kumar Lenka:** Investigation. **Ayon Chakraborty:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Conceptualization.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ayon Chakraborty reports financial support and administrative support were provided by Indian Institute of Technology Madras. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The data is shared by us and the URL is included in the paper draft.

## References

[1] F. Miao, Y. Huang, Z. Lu, T. Ohtsuki, G. Gui, H. Sari, Wi-Fi Sensing Techniques for Human Activity Recognition: Brief Survey, Potential Challenges, and Research Directions, ACM Comput. Surv. 57 (5) (2025) 1–30.

[2] F. Meneghello, C. Chen, et al., Towards Integrated Sensing and Communications in IEEE 802.11bf Wi-Fi Networks, 2022, arXiv preprint.

[3] E. Cianca, M. De Sanctis, et al., Radios as Sensors, IEEE Internet Things J. (2016).

[4] Y. Ma, G. Zhou, et al., Wi-Fi Sensing with Channel State Information: A Survey, ACM Comput. Surv. (2019).

[5] Y. Zheng, Y. Zhang, et al., Zero-Effort Cross-Domain Gesture Recognition with Wi-Fi, in: ACM International Conference on Mobile Systems, Applications, and Services (ACM MobiSys), 2019.

[6] S. Liu, Y. Zhao, et al., WiCount: A Deep Learning Approach for Crowd Counting Using Wi-Fi Signals, in: IEEE International Symposium on Parallel and Distributed Processing with Applications and IEEE International Conference on Ubiquitous Computing and Communications, ISPA/IUCC, 2017.

[7] W. Jiang, C. Miao, et al., Towards Environment Independent Device Free Human Activity Recognition, in: ACM International Conference on Mobile Computing and Networking (ACM MobiCom), 2018.

[8] X. Zheng, K. Yang, J. Xiong, L. Liu, H. Ma, Pushing the limits of WiFi sensing with low transmission rates, IEEE Trans. Mob. Comput. (2024).

[9] B. Barahimi, H. Singh, H. Tabassum, O. Waqar, M. Omer, RSCnet: Dynamic CSI Compression for Cloud-based WiFi Sensing, in: ICC 2024-IEEE International Conference on Communications, IEEE, 2024, pp. 4179–4184.

[10] Y. He, M. Wei, D. Li, P. Li, H. Li, CFNet: CSI Compression Feedback Network based on WiFi Sensing, Eng. Res. Express 7 (1) (2025) 015221.

[11] A. Dhekne, A. Chakraborty, K. Sundaresan, S. Rangarajan, TrackIO: Tracking First Responders Inside-Out, in: 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019.

[12] A. Sen, A. Mandal, P. Karmakar, A. Das, S. Chakraborty, Passive Monitoring of Dangerous Driving Behaviors Using mmWave Radar, Pervasive Mob. Comput. 103 (2024) 101949.

[13] H. Xue, W. Jiang, et al., DeepMV: Multi-View Deep Learning for Device-Free Human Activity Recognition, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. (2020).

[14] B. Sheng, F. Xiao, et al., Deep Spatial–Temporal Model Based Cross-Scene Action Recognition Using Commodity WiFi, IEEE Internet Things J. (2020).

[15] Y. Gu, X. Zhang, et al., WiGRUNT: WiFi-Enabled Gesture Recognition Using Dual-Attention Network, IEEE Trans. Hum.- Mach. Syst. (2022).

[16] K. He, X. Zhang, et al., Deep Residual Learning for Image Recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (IEEE CVPR), 2016.

[17] G. Menghani, Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better, ACM Comput. Surv. 55 (12) (2023) 1–37.

[18] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, 2015, arXiv preprint.

[19] W. Roth, G. Schindler, B. Klein, R. Peharz, S. Tschiatschek, H. Fröning, F. Pernkopf, Z. Ghahramani, Resource-efficient Neural Networks for Embedded Systems, J. Mach. Learn. Res. 25 (50) (2024) 1–51.

[20] Y. Nimmagadda, Model Optimization Techniques for Edge Devices, in: Model Optimization Methods for Efficient and Edge AI: Federated Learning Architectures, Frameworks and Applications, Wiley Online Library, 2025, pp. 57–85.

[21] A. Chakraborty, Sensing and Networked Systems Engineering (SeNSE) Group, IIT Madras, WISDOM Dataset and Models, URL https://sense.cse.iitm.ac.in/wisdom/.

[22] D. Halperin, W. Hu, et al., Tool release: Gathering 802.11n traces with Channel State Information, ACM SIGCOMM Comput. Commun. Rev. (2011).

[23] Y. Xie, Z. Li, et al., Precise Power Delay Profiling with Commodity WiFi, in: ACM International Conference on Mobile Computing and Networking (ACM MobiCom), 2015.

[24] X. Jiao, W. Liu, et al., openwifi: A Free and Open-source IEEE 802. 11 SDR implementation on SoC, in: IEEE Vehicular Technology Conference, VTC2020-Spring, 2020.

[25] Z. Jiang, T.H. Luan, et al., Eliminating the Barriers: Demystifying WiFi Baseband Design and Introducing the PicoScenes WiFi Sensing Platform, IEEE Internet Things J. (2022).

[26] F. Gringoli, M. Schulz, et al., Free Your CSI: A Channel State Information Extraction Platform For Modern WiFi Chipsets, in: International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, 2019.

[27] M. Zhu, S. Gupta, To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017, arXiv preprint.

[28] B. Jacob, S. Kligys, et al., Quantization and Training of Neural Networks for Efficient Integer-arithmetic-only Inference, in: IEEE Conference on Computer Vision and Pattern Recognition (IEEE CVPR), 2018.

[29] C. Xiao, Y. Lei, et al., DeepSeg: Deep-Learning-Based Activity Segmentation Framework for Activity Recognition Using WiFi, IEEE Internet Things J. (2021).

[30] Z. Chen, L. Zhang, et al., WiFi CSI Based Passive Human Activity Recognition Using Attention Based BLSTM, IEEE Trans. Mob. Comput. (2019).

[31] H. Zou, Y. Zhou, et al., DeepSense: Device-Free Human Activity Recognition via Autoencoder Long-Term Recurrent Convolutional Network, in: IEEE International Conference on Communications, IEEE ICC, 2018.

[32] F. Wang, W. Gong, et al., On Spatial Diversity in Wi-Fi based Human Activity Recognition: A Deep Learning-Based Approach, IEEE Internet Things J. (2019).

[33] C. Li, M. Liu, et al., WiHF: Enable User Identified Gesture Recognition with WiFi, in: IEEE Conference on Computer Communications, IEEE ICC, 2020.

[34] X. Zhang, C. Tang, et al., WiFi-Based Cross-Domain Gesture Recognition via Modified Prototypical Networks, IEEE Internet Things J. (2022).

[35] S.M. Hernandez, E. Bulut, WiFi Sensing on the Edge: Signal Processing Techniques and Challenges for Real-World Systems, IEEE Commun. Surv. Tutorials (2022).

[36] J. Yang, X. Chen, et al., EfficientFi: Toward Large-scale Lightweight WiFi Sensing via CSI Compression, IEEE Internet Things J. (2022).

[37] S.M. Hernandez, E. Bulut, Lightweight and Standalone IoT Based WiFi Sensing for Active Repositioning and Mobility, in: International Symposium on "A World of Wireless, Mobile and Multimedia Networks", WoWMoM, 2020.

[38] S.S. Saha, S.S. Sandha, et al., Machine Learning for Microcontroller-class Hardware – A Review, IEEE Sens. J. (2022).

[39] G. Menghani, Efficient deep learning: A survey on making deep learning models smaller, faster, and better, ACM Comput. Surv. (2023).

[40] H. Cai, C. Gan, et al., TinyTL: Reduce memory, not parameters for efficient on-device learning, in: Advances in Neural Information Processing Systems, 2020.

[41] M. Abadi, A. Agarwal, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, URL https://www.tensorflow.org/.

[42] R. David, J. Duke, et al., TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems, 2020, arXiv preprint.

[43] S. Gayen, Y. Shankar, A. Chakraborty, Improving Network Resource Utilization for Distributed Wireless Sensing Applications, in: Proceedings of the Twenty-Fifth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (ACM MobiHoc), 2024, pp. 452–457.

[44] N. Semiconductor, Power Profiler Kit II, URL https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2.

[45] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. (1997).

[46] S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, in: International Conference on Machine Learning, 2015.

[47] G.E. Hinton, N. Srivastava, et al., Improving Neural Networks by Preventing Co-adaptation of Feature Detectors, 2012, arXiv preprint.

[48] G. Strang, Linear algebra and its applications, 2006.

[49] Y. Matsubara, M. Levorato, et al., Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Comput. Surv. (2022).