Investigation of Learning Robustness Techniques in WiFi Sensing Deep Learning Models

Analyzing the impact of various techniques on model performance

Oleh Grypas[1]
Supervisor(s): Dr. Arash Asadi[1], **Fabian Portner**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

<

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Oleh Grypas
Final project course: CSE3000 Research Project
Thesis committee: Dr. Arash Asadi, Fabian Portner, Dr. Ranga Rao Venkatesha Prasad

Abstract

WiFi sensing has shown great promise in applications such as activity recognition, human identification, and health monitoring. However, models trained on Channel State Information (CSI) data often suffer from poor generalizability due to high variance and inconsistent reporting practices, especially in small-data regimes. Despite increasing attention to deep learning-based approaches, the community lacks standardized guidelines on handling variance and overfitting across architectures and datasets. In this work, we first review existing robustness strategies employed in related fields to identify techniques suitable for WiFi sensing. Based on this survey, we systematically benchmark five representative methods—stability training, mixup, coupled weight decay, early stopping, and label smoothing—selected for their theoretical grounding and prior success in mitigating variance. Our evaluation spans three model types (LeNet, LSTM, CNN+GRU) and two CSI datasets (Widar, NTU-Fi), using stratified 5-fold cross-validation with repeated trials to ensure reliable variance estimation. Results show that (1) stability training consistently improves moderately performing models, (2) coupled weight decay is especially effective for LSTMs, and (3) combining techniques can harm performance in near-saturated scenarios. Our findings offer actionable, architecture- and dataset-specific guidelines for improving robustness and reproducibility in WiFi sensing research.

1 Introduction

WiFi sensing is emerging as a promising non-intrusive technology for various applications. Extensive research has been done into various uses for the technology, and its potential implications for everyday life. Key use cases include human gesture recognition [1], occupancy detection [2], and health monitoring [3] , among others. These recent advances have unveiled highly capable methods which can detect human presence [2] and even estimate breathing rates [3] behind solid barriers.

A significant limitation for WiFi sensing is that deep learning models trained on small datasets often underperform [4]. Common issues include overfitting, sensitivity to noise, and strong dependence on model initialization. While robustness techniques have been explored in other areas like computer vision and speech [5, 6, 7], their effects on WiFi sensing models have not been properly studied. This matters because WiFi CSI data behaves differently from other types of data — it is strongly affected by small changes in the environment [8], lacks clear structure, and is often limited in size[9, 10]. As a result, techniques that work well in other domains may not have the same impact here.

In this research project, we examine the reliability of state-of-the-art training robustness techniques—such as stability training [11], label smoothing [12], and data augmentation [13], and others—in improving model accuracy and robustness when training on small datasets. Our selection of techniques is based on a review of prior work in related areas such as computer vision and speech, where these methods have been shown to reduce overfitting, sensitivity to noise, and dependence on favorable weight initialization. By evaluating these techniques in the context of deep learning-based WiFi sensing, we aim to identify best practices for training stable and robust models, while also following strong experimental reporting standards. This leads us to the following research question:

"How can state-of-the-art learning robustness techniques be leveraged to improve the stability and robustness of deep neural networks for WiFi sensing, particularly under limited data conditions?"

To answer this question, we tested five different training techniques on three different types of models using two WiFi datasets (see Section 3.3 for details).

Our key contributions are summarized as follows:

- We present the first systematic test of these five prominent learning robustness techniques specifically for WiFi sensing, comparing how they perform across different models and datasets.
- We demonstrate a set of best practices when enhancing training with learning robustness techniques under small-data regimes, and how it is context-dependent.
- We identified a set of best practices for reporting model performance when working with limited data, including proper statistical evaluation methods and reliable accuracy reporting that accounts for the high variance typical in small-dataset scenarios.
- We provide clear guidelines for other researchers about when and how to use these techniques, along with all the code and settings needed to reproduce our results.

2 Background

This section reviews key topics in our study: deep learning model architectures in WiFi sensing, evaluation strategies for reliable performance estimation, and techniques for improving robustness and generalization under small-data regimes.

2.1 Channel State Information

Channel State Information (CSI) captures how an environment distorts a signal by measuring amplitude changes, phase shifts, time delays, and frequency shifts—typically as a complex-valued matrix. These parameters let transmitters and receivers compensate for real-world propagation effects (e.g., multipath echoes or movement) to optimize communication. It also provides the foundation for WiFi sensing, as capturing these propagation effects allows us to detect movement [1] and the location [2] of certain objects. However, CSI data introduces critical constraints for deep learning. Publicly available CSI datasets are quite small (typically 10s-100s of samples), and offer narrow diversity (homogeneous environments, restricted activity sets) [9, 10, 14]. These traits amplify overfitting, initialization sensitivity, and gradient instability– core challenges addressed by robustness techniques.

2.2 Deep Learning Architectures In WiFi Sensing

While WiFi sensing research has explored various sophisticated architectures, our study focuses on three fundamental network types available in the SenseFi framework. This choice reflects a key limitation of our work: we are constrained to relatively simple architectures, which may not capture the full complexity of CSI data as effectively as state-of-the-art models.

LeNet-style CNNs are simple convolutional architectures originally designed for digit recognition. While they can detect spatial patterns in 2D CSI matrices, their shallow structure (2-3 layers) limits complex feature learning. The small receptive field misses long-range

dependencies in CSI data, and the limited depth prevents sophisticated feature representations. This simplicity makes LeNet particularly prone to overfitting on small datasets.

Long Short-Term Memory (LSTM) networks were designed to fix the vanishing gradient problem in traditional RNNs, using gating mechanisms to learn long-term dependencies in sequential data. Despite this improvement, LSTMs face challenges in WiFi sensing: they're computationally expensive, can still struggle with very long sequences, and suffer from gradient instability on noisy CSI data. The additional parameters from gating mechanisms increase overfitting risk in small-data scenarios.

Hybrid CNN+GRU architectures combine spatial feature extraction (CNN) with temporal modeling (GRU). While more sophisticated than the previous architectures, this hybrid remains relatively simple compared to modern attention-based approaches. The CNN extracts spatial features from CSI frames, while the GRU models temporal dependencies with fewer parameters than LSTMs. However, the CNN component may miss complex spatial relationships, and the GRU can still struggle with long-term dependencies.

The constraint to these simpler architectures through SenseFi represents a significant limitation of our study, as state-of-the-art WiFi sensing research increasingly relies on more sophisticated models like attention mechanisms, graph neural networks, or advanced transformer architectures that can better capture the complex spatio-temporal relationships in CSI data. Additionally, computational resource limitations further influenced our choice toward these simpler models, as training more complex architectures would have required significantly more computational power than was available for this study.

2.3 Evaluation Under Limited Data

A brief survey of recent WiFi sensing studies shows that most report only a single-shot accuracy from one train-test split, without any measure of variability or clear partitioning strategy, making it hard to judge how reliable those results really are [10, 15, 16]. In some cases, data leakage issues (e.g., including samples from the test set in the training set) further inflate performance estimates [17, 18]. To address this, we follow better practices from both WiFi sensing and general machine learning literature.

2.3.1 Statistical Reporting Metrics

When reporting performance on classification tasks, in addition to accuracy, other measures should be included, such as precision, recall (or F1-score) and AUC, especially under class imbalance. For continuous outputs, report MSE or MAE along with R^2. Crucially, every reported point estimate should be accompanied by uncertainty quantification: for example, standard deviations or 95% confidence intervals computed over multiple runs [19]. Visualizing the distribution of results (e.g., via boxplots or error bars) helps readers see the spread rather than a single number.

2.3.2 Statistical Validation Approaches

To guard against lucky initialization or optimistic bias, use evaluation splits that avoid leakage and give a better sense of stability. A common choice is stratified k-fold cross-validation [20, 21]: split data into k roughly equal parts while keeping each class's proportion

similar in every part, train on k-1 parts and test on the remaining part, and rotate so each part is used for testing. This ensures that data leakage is prevented, as otherwise the model may "see" similar patterns in both the training and testing sets if they overlap and overestimate real-world performance. This also helps produce a fair performance estimate, especially when classes are imbalanced, since each fold reflects the overall class distribution. For very small datasets, bootstrapping632+ [22, 23] (sampling with replacement and testing on held-out samples) can complement cross-validation by giving another view of variability. It also improves on regular bootstrap by combining training error (optimistic) and out-of-bag error (pessimistic), where, depending on how badly the model overfits, it could rely more on the out-of-bag error. Additionally, since deep models can vary with initialization, reporting performance over several runs with random weight initializations greatly helps prevent reliance on lucky starting points.

In summary, our survey of the WiFi CSI literature found that very few studies reported standard deviations or confidence ranges, nor did they use repeated splits with proper subject separation. Therefore, we recommend the following structure: (1) report averages plus standard deviations (or bootstrap-based intervals) for each metric; (2) use repeated k-fold splits (with subject-wise grouping where relevant) and/or bootstrap sampling; and (3) repeat the previous step with different random seeds to ensure more diverse set of initial weights. This can make an evaluation under limited CSI data more reliable and reproducible.

3 Related Work

Various learning robustness techniques have been proposed to make deep models robust under data scarcity [4, 6, 5]. These strategies include the following:

3.1 Training Robustness Techniques

We organize training robustness techniques into two categories based on their intervention point: data-based techniques directly modify input data or labels (e.g., augmentations or label adjustments), while learning-based techniques alter the model's learning behavior or rules (e.g., optimization constraints or task redesign).

3.1.1 Learning-Based Techniques

Learning-based approaches intervene in the model's training rules or overall workflow—through regularization penalties, early stopping, or specialized training schemes—without changing the underlying data. These methods guide the optimization process or reshape the learning task so that the network adopts simpler, more generalizable patterns. When CSI data are sparse, constraining how the model fits or how quickly it learns can prevent runaway overfitting and yield more consistent WiFi sensing results. [7, 6, 10]

Dropout: Dropout randomly deactivates neurons during training, preventing co-adaptation of features. Srivastava et al. (JMLR'14) showed dropout "significantly reduces overfitting" and improves performance across vision, speech, etc. [24]. By effectively training an ensemble of thinned networks, dropout regularizes any model with many parameters. In WiFi applications with limited CSI samples, dropout helps avoid memorizing the small dataset, forcing the network to learn more robust, distributed features.

L2 Regularization (coupled Weight Decay): This technique reduces overfitting by discouraging large weights in the model. It adds a penalty term ($\lambda||w||^2$) to the loss function during training, which pushes weights toward smaller values. This smoothing effect creates simpler decision boundaries and prevents models from fitting noise in small datasets. For WiFi CSI tasks with limited data, it's a standard method to control model complexity. [25]

Early Stopping: Early stopping monitors validation performance and halts training when improvement ceases. It is a well-known regularization: after enough training steps the model begins to overfit, so stopping early keeps it in the "sweet spot" [26]. This method is especially useful when data are scarce, since it prevents the network from fitting noise. In practice, one trains on CSI data while holding out a validation set; when validation error stops dropping, training stops. The result is a more stable model that generalizes better to new WiFi scenarios.

Transfer Learning: Transfer learning reuses a model (or its weights) pre-trained on one domain for a related task. As noted in the SenseFi WiFi sensing benchmark, "pre-training models on large datasets can be generalized to downstream WiFi-sensing tasks" [10]. In limited-data regimes, one may take a neural network pre-trained on an image or RF dataset and fine-tune it on a small WiFi CSI dataset. The SenseFi study also found that even shallow pre-trained models transfer well for human activity and identification tasks [10]. The general idea is that shared low-level features can be re-used, giving a strong initialization and accelerating learning on scarce WiFi data.

Few-shot: Few-shot learning techniques explicitly train models to learn from very few examples per class. They often use metric learning (e.g., Siamese or prototypical networks) so that classes can be recognized with a few labeled samples [27]. In WiFi sensing, approaches like ReWiS leverage prototypical networks for robust cross-environment activity recognition, while others apply meta-learning to minimize data collection in tasks like indoor positioning [28]. In general, few-shot methods use only a few samples from each category during training and have shown promise for data-efficient WiFi tasks, such as gesture recognition and human activity identification [27]. By leveraging episodic training or contrastive losses, systems like Wi-CHAR dynamically adjust feature prototypes to mitigate noise and cross-domain discrepancies, enabling deployment with minimal labeled data per user or activity (e.g., $>$93% accuracy with five-shot learning) [27].

Self-Supervised Learning: Self-supervised learning (SSL) creates proxy tasks using unlabeled data to learn useful representations. For example, one might train the network to predict signal transformations or segments of CSI without using labels. SSL is known to "outperform supervised learning on less related source datasets" when the self-supervised pretext is well-designed [7]. In practice, a model could be pre-trained on large amounts of unlabeled WiFi signals (e.g. predicting future CSI or reconstructing input) and then fine-tuned on the small labeled set. Such unsupervised pretraining often accelerates convergence and yields more robust features for the downstream WiFi tasks [7].

Multi-Task Learning: In multi-task learning (MTL), a single model is trained on several related tasks simultaneously, sharing representations. Caruana (1997) described MTL as learning tasks in parallel "while using a shared representation" so that each task benefits from the others. The Wikipedia entry notes that MTL improves accuracy by exploiting commonalities between tasks. For WiFi sensing, one could jointly train on multiple activities (e.g. walking, falling) or modalities (e.g. gesture plus room occupancy) so that the model learns features useful across tasks. This shared learning acts as an inductive bias, effectively regularizing the model and reducing overfitting when each task's data is limited [7].

3.1.2 Data-Based Techniques

Data-based methods act directly on the training examples—either by expanding the dataset or by softly modifying the labels—so that the model sees a wider variety of inputs during learning. In small-data regimes, where collecting more CSI measurements is expensive or impractical, these approaches simulate new conditions, smooth noisy labels, or expose the network to harder examples. By enriching and perturbing the data, they help the model learn invariances and avoid overfitting to the limited original samples, which is crucial when channel variations and environmental noise can otherwise lead to wildly inconsistent WiFi sensing performance. [4, 10, 6, 5]

MixUp Data Augmentation: MixUp augments data by forming convex combinations of input pairs and their labels, effectively smoothing decision boundaries between samples. Zhang et al. showed that training on these mixed inputs improves generalization and robustness – e.g. reducing memorization of noisy labels and boosting resistance to adversarial perturbations [13]. In WiFi sensing with limited CSI data, MixUp can simulate intermediate channel states, yielding a richer training set and more stable feature learning.

Random-Transformation Augmentation: Standard augmentations (e.g. random flips, crops, rotations, noise) and specialized variants (e.g. random erasing, CutOut) create diverse training examples without new labels. For instance, random erasing replaces a random image patch with noise, teaching the model to ignore occlusions [7]. Such random input-level transforms expand WiFi datasets by mimicking physical variations (like different user orientations or noise patterns), reducing overfitting on scarce data. In general, data warping methods significantly boost predictive performance under data scarcity [7].

Stability Training: This technique explicitly regularizes the model to produce similar outputs for slightly perturbed inputs. For example, Zheng et al. (CVPR'16) introduced stability training, which penalizes output changes under small distortions (e.g. slight compression, scaling, or noise) [11]. By encouraging invariance to input jitter, stability training makes the network's representation more robust to real-world signal fluctuations. In WiFi sensing, where channel measurements can vary subtly (due to environment or hardware noise), stability training helps maintain consistent predictions even when the CSI inputs are perturbed.

Adversarial Training: In adversarial training, the model is trained on both clean and adversarially perturbed examples. Goodfellow et al. (ICLR'15) generated "worst-case" input perturbations and added them to training, which noticeably reduced test errors under attack [29]. In practice, crafting adversarial CSI samples (small changes to channel phases/amplitudes) and training the network on them makes the model robust to malicious or unintentional distortions. Adversarial training thus improves stability by explicitly defending against small but worst-case input changes [30].

Label Smoothing: Instead of one-hot targets (0 or 1), label smoothing assigns soft labels (e.g. 0.9/0.1) during training [12]. Müller et al. (NeurIPS'19) explain that smoothing "prevents the network from becoming over-confident" and generally improves generalization and calibration [31]. In low-data regimes, smoothing guards against fitting noise: the model learns to hedge its bets rather than memorize small peculiarities of the limited training set. For WiFi sensing, label smoothing can mitigate label noise (e.g. uncertain activity annotations) and yield a more stable classifier under dataset scarcity.

Each of these training-time methods complements standard learning. By adjusting the architecture or altering the training data and loss, one can often achieve better robustness

and generality without fundamentally changing the model class. These practical techniques — pretraining, augmentation, noise injection, and regularization — collectively help the model perform well even under domain shifts or attacks. In summary, these training-based techniques are widely cited as solutions to data scarcity in deep learning [6, 4]. Each approach can make a model more robust by either effectively increasing/augmenting the data or by guiding learning to generalize better from limited examples.

3.2 Architecture Robustness Techniques

Robustness can also arise from the network design itself. When data are limited, the choice of architecture and its complexity plays a critical role [6]. Key design strategies include:

Simplified Model Complexity: Limiting the depth or width of the network helps prevent overfitting on small datasets [6]. Alzubaidi et al. report that very deep or wide models tend to memorize scant data, whereas "non-intricate" (simpler) CNNs often perform better in low-data regimes [32]. Reducing layers or channels can improve generalization, at the cost of representational power. Thus, architectures should be carefully scaled to the data size: extremely deep ResNets or Transformers may be ill-suited unless paired with massive augmentation or pretraining.

Regularization Layers and Mechanisms: Incorporating dropout, batch normalization, or weight decay into the architecture provides strong built-in regularization [6]. In fact, even with scarce data, dropout remains effective at reducing overfitting [6]. Similarly, adding noise layers or explicit sparsity constraints can make models more robust. These components are architectural choices that mitigate variance when each parameter can only be supported by few examples.

Generative/Adversarial Architecture Modules: Some architectures embed generative networks to address data scarcity. For instance, a GAN-based branch can be attached to the model to generate synthetic CSI conditioned on class labels, effectively learning to expand the dataset [10]. In WiFi sensing, adversarial networks have been used not only to augment data but also to perform domain adversarial training: by having the model include a domain-classifier adversarially, it learns features invariant to changing environments (e.g. room layout or device shift) [10]. Variational autoencoder components have likewise been integrated (e.g. for CSI compression) to capture data distributions and improve robustness.

Attention and Transformer Blocks: Attention mechanisms allow the model to focus on the most informative parts of the input. Vision Transformers (ViT) and similar blocks have shown power in many domains, but they come with huge parameter counts. In practice, SenseFi finds that ViT-like models do not outperform much simpler CNN/GRU models on WiFi tasks when training data are limited [10]. The large number of parameters makes Transformers "not really attractive" for supervised learning on small CSI datasets [10]. In general, one should avoid over-parameterized architectures (like vanilla Transformers) unless ample data or heavy regularization is available.

The above design principles guide how to train such networks with small data in mind. Notably, simplifying the network may matter more than adding complexity: studies show that with limited CSI samples, shallower CNNs with strong regularization can outperform very deep nets [10]. Embedding domain knowledge (e.g. physics-inspired layers) is also a potential strategy, though most work relies on generic modules. Overall, careful architectural choices – balancing capacity and inductive biases – are crucial for stability when training

WiFi models on small datasets.

4 Gap And Motivation

Despite substantial progress in general deep-learning strategies for small-data scenarios [6, 4], domain-specific applications like WiFi CSI sensing lack systematic, controlled comparisons of robustness methods. Prior works often discuss robustness techniques only within broader pipelines or focus on other modalities (e.g., images, remote sensing). Existing benchmarks such as RobustART [7], RobustBench [33], and ARES-Bench [34] address adversarial or noise robustness but in large-data regimes and different input types, leaving small-data WiFi sensing unexplored.

In our work, we apply five widely used robustness techniques across multiple model families—spanning convolutional, recurrent, and hybrid architectures—to WiFi CSI tasks. Because we evaluate each technique identically on several architectures, our results identify practices that hold generally, rather than being tied to a single network design. This "architecture-agnostic" evaluation yields actionable guidelines for diverse WiFi sensing models under limited data. To our knowledge, this is the first reproducible benchmark comparing multiple robustness methods across architectures and datasets in WiFi CSI sensing, motivating our study.

5 Methodology

This section describes the experimental setup, including datasets, models, learning robustness techniques, training strategies, and evaluation methods. Our implementation builds upon the SenseFi benchmark [10], which provides a standardized framework for WiFi sensing using deep learning. It includes multiple model architectures and four publicly available datasets. We extend the codebase by integrating additional robustness techniques and statistical evaluation tools.

5.1 Dataset

We use two open-source human activity recognition datasets from SenseFi: NTU-Fi [10] and Widar [9]. These both provide CSI inputs but differ in baseline difficulty: Widar typically yields lower baseline accuracy, while NTU-Fi models often achieve high accuracy. This allows us to examine whether robustness techniques can improve weaker performance in one case and preserve high performance in another.

5.2 Models

Following SenseFi, we evaluate three relatively simple architectures: LeNet (a CNN, referred to as CNN-5 in SenseFi), an LSTM-based RNN, and a hybrid CNN+GRU. This allows us to see whether the effects of these learning robustness techniques can be preserved across different architectures, which may be more effective for certain designs. We note that this choice reflects both the SenseFi framework's available implementations and practical constraints: limited computational resources make very deep or large models difficult to train, especially on modest hardware. Moreover, simpler architectures can have an inherent advantage in small-data regimes by reducing overfitting, though confirming this benefit fully would require further study with varied model sizes.

5.3 Learning Robustness Techniques

Given our use of relatively simple, lightweight models (LeNet, LSTM, CNN+GRU) and limited compute budget, we focused on robustness methods that are independent of architecture and incur modest overhead. We selected five techniques—stability training, L2 regularization (coupled weight decay), early stopping, label smoothing, and MixUp—because they (1) can be applied uniformly across different network types without modifying the model structure; (2) require minimal extra computation or changes to the training pipeline, making long experiments feasible; (3) collectively address key challenges in small-data WiFi sensing: stability to input noise (stability training), overfitting control (coupled weight decay, early stopping, label smoothing), and decision-boundary smoothing/generalization (MixUp); and (4) fit within our time constraints, whereas other promising methods (e.g., extensive adversarial training loops or self-supervised pretraining on large unlabeled corpora) would demand substantial setup changes or compute beyond our current resources. This selection balances broad coverage of robustness goals with practical feasibility for reproducible experiments under limited data and hardware. Full details and code are included in Appendix A.

5.4 Evaluation

Each model variant is evaluated with regular stratified 5-fold cross-validation which was implemented in the sklearn library. We report the mean and standard deviation of accuracy over the five folds. Since F1-score behaved similarly to accuracy in preliminary tests, we omit it from the main tables and include detailed F1 results in Appendix B. To account for randomness in training (e.g., weight initialization), each configuration is run twice with different random seeds, and we average accuracy across those runs.

5.5 Experimental Design

We structure our experiments in two stages to isolate individual effects and explore useful combinations of the robustness methods introduced earlier. All experiments use the same base training pipeline from SenseFi, differing only in which techniques are applied.

First, we establish a baseline by training each model on each dataset without any added robustness methods. This provides a reference point for measuring improvements.

Next, in the individual ablation stage, we apply each robustness technique separately, one at a time, to the baseline setup. This shows how much benefit each method brings on its own for a given model and dataset.

Finally, we conduct combination experiments to see if certain methods work well together. Based on prior findings and practical considerations, we select these groups:

- **Generalization** – Using weight decay, early stopping, and label smoothing to strongly limit overfitting.
- **Stable MixUp** – Since both stability training and MixUp improve robustness by introducing changes to the data, combining them can lead to cumulative improvement.
- **Stable Stopping** – Pairing the single most effective method from ablation (stability training) with early stopping to check if stopping earlier further improves robustness.
- **Mixed Trio** - applying MixUp, stability training and early stopping together to see if their effects accumulate.

- **Full Combination** – applying all robustness methods at once to observe whether maximal intervention helps or if interactions interfere.

6 Results

Our evaluation approach highlights the importance of proper statistical reporting when working with small datasets. That is why we report mean accuracy ± standard deviation over stratified 5-fold cross-validation (each configuration run twice with different seeds). F1-score trends closely mirror accuracy (detailed in Appendix B), so we discuss accuracy here and refer to F1 only when it reveals additional insights. All numeric values below come from Tables 1 and 2 in the results file.

Table 1: Performance of each method on the Widar dataset. Values are mean accuracy ± standard deviation.

Method	**LeNet**	**LSTM**	**CNN+GRU**
Baseline	0.6741 ± 0.0062	0.6021 ± 0.0066	0.6799 ± 0.0089
Stability training	0.7055 ± 0.0058	0.6045 ± 0.0046	0.6884 ± 0.0164
Mixup	0.6740 ± 0.0094	0.6091 ± 0.0059	0.6732 ± 0.0145
Weight Decay	0.6849 ± 0.0065	0.6369 ± 0.0054	0.6853 ± 0.0058
Early Stopping	0.6836 ± 0.0090	0.6068 ± 0.0063	0.6821 ± 0.0065
Label Smoothing	0.6754 ± 0.0082	0.5821 ± 0.0068	0.6861 ± 0.0126
Generalization	0.6835 ± 0.0068	0.6238 ± 0.0119	0.6858 ± 0.0079
Stable MixUp	0.6810 ± 0.0091	0.6222 ± 0.0054	0.6931 ± 0.0086
Stable Stopping	0.7041 ± 0.0089	0.6050 ± 0.0067	0.6837 ± 0.0204
Mixed Trio	0.6287 ± 0.0078	0.5311 ± 0.0134	0.4071 ± 0.0289
Full Combination	0.6949 ± 0.0113	0.6848 ± 0.0053	0.6516 ± 0.0197

6.1 Which Methods Worked Well?

On the Widar dataset, stability training consistently improved accuracy beyond its usual variation. For example, LeNet rose from about 67.4% ±0.6% to 70.6% ±0.6%, and CNN+GRU rose from 68.0% ±0.9% to 68.8% ±1.6%, indicating a reliable gain. Weight decay also helped, especially for LSTM (from about 60.2% ±0.7% to 63.7% ±0.5%), showing that penalizing large weights reduces overfitting on this smaller dataset. MixUp, label smoothing, and early stopping alone did not move accuracy beyond one standard deviation, so their individual benefit on Widar was limited.

On NTU-Fi, individual techniques behaved as expected: none caused significant drops in accuracy beyond the normal variation. Accuracy changes for stability training, weight decay, MixUp, label smoothing, or early stopping alone stayed within ±0.5%, matching the standard deviation. This matches our expectation that single methods would not harm performance on a near-saturated task.

Table 2: Performance of each method on the NTU-Fi dataset. Values are mean accuracy ± standard deviation.

Method	**LeNet**	**LSTM**	**CNN+GRU**
Baseline	0.9950 ± 0.0038	0.9963 ± 0.0054	0.9764 ± 0.0134
Stability Training	0.9963 ± 0.0031	0.9963 ± 0.0046	0.9847 ± 0.0158
Mixup	0.9967 ± 0.0038	0.9867 ± 0.0096	0.9681 ± 0.0146
Weight Decay	0.9942 ± 0.0049	0.9979 ± 0.0029	0.9542 ± 0.0167
Early Stopping	0.9954 ± 0.0060	0.9983 ± 0.0029	0.8333 ± 0.2278
Label Smoothing	0.9979 ± 0.0029	0.9967 ± 0.0038	0.9583 ± 0.0123
Generalization	0.9983 ± 0.0029	0.9925 ± 0.0085	0.8979 ± 0.0688
Stable Mixup	0.9963 ± 0.0041	0.9950 ± 0.0043	0.9344 ± 0.0244
Stable Stopping	0.9938 ± 0.0035	0.9913 ± 0.0077	0.8094 ± 0.0933
Mixed Trio	0.9875 ± 0.0171	0.9433 ± 0.0253	0.8750 ± 0.1197
Full Combination	0.9963 ± 0.0050	0.9654 ± 0.0155	0.7719 ± 0.2766

6.2 Surprising Or Counterintuitive Results

We expected that mixing methods might hurt accuracy on NTU-Fi, and indeed using several techniques at once often caused drops beyond the normal range. On Widar, however, combining three or more techniques sometimes caused a clear accuracy drop. For instance, stability training with MixUp and early stopping led to roughly a 4.5% decrease. This shows that stacking methods can actually hurt learning even on moderate datasets.

6.3 How Do Models Differ In Their Benefit?

Different architectures showed different patterns. LeNet on Widar benefited most from stability training, with smaller gains from weight decay. It was also least harmed by combining methods, suggesting simpler models may better tolerate extra regularization. LSTM on Widar gained notably from weight decay and, in combination, from stronger regularization, implying temporal models may overfit more and need tighter control. CNN+GRU on Widar saw some improvement from stability training but was most vulnerable when multiple methods were combined, dropping accuracy more sharply. On NTU-Fi, none of the individual methods harmed any model beyond normal variation, but combinations caused larger drops for CNN+GRU and LSTM than for LeNet, indicating that larger or hybrid architectures can be more sensitive to over-regularization when data are already sufficient.

6.4 Which Technique Offers Best Overall Stability?

Stability training stands out as the most reliable single method on the moderate-baseline dataset: it improved accuracy beyond the standard deviation for all models on Widar without causing large downsides. Weight decay is especially helpful for LSTM models. MixUp and label smoothing had limited individual benefit but sometimes added small gains when

paired with stability training for certain models. Early stopping alone did not yield clear benefits. On NTU-Fi, no single technique improved accuracy beyond normal variation, so the safest approach there is to use minimal or no added methods unless validation suggests otherwise.

7 Conclusions and Future Work

We evaluated five robustness techniques—stability training, mixup, label smoothing, early stopping, and coupled weight decay—across three model architectures (LeNet, LSTM and CNN+GRU) on two WiFi CSI datasets (Widar and NTU-Fi), using fixed hyperparameters for reproducibility. Our goal was to see how each method behaves under a controlled setup and how different models react.

On the weaker-performing dataset (Widar), stability training stood out for both LeNet and CNN+GRU, consistently improving accuracy and macro-F1 with modest regularization. LSTM also benefited but showed its best gains when using coupled weight decay or the full combination of techniques, likely because its temporal nature needed slightly stronger regularization to generalize well on limited data. Aggressive stacks (e.g., combining mixup with early stopping) sometimes led to underfitting, especially with CNN+GRU when early stopping cut training too soon.

On the near-saturated dataset (NTU-Fi), most robustness methods tended to degrade performance if applied without caution. Stability training alone caused minor drops for some models but was less harmful than heavier combos; LSTM again showed that coupled weight decay and full-combo settings could be tuned to maintain or slightly improve its performance, while LeNet and CNN+GRU generally saw diminishing returns or small declines. This saturation effect underscores that when baseline accuracy is already very high, extra regularization or augmentation can easily push the model off its peak.

Because we held hyperparameters constant, these findings reflect each technique's raw effect under a fixed training regime; in practice, per-technique tuning might shift the exact gains or losses. Still, the patterns suggest starting with stability training for LeNet- or CNN-based setups on moderate baselines, and considering coupled weight decay or light full-combo for LSTM if initial experiments show underfitting otherwise. Always monitor validation carefully: for example, when combining mixup and early stopping, you may need to relax the stopping criterion to avoid premature cut-off. Although the exact reason for this combination led to worse results is unknown.

Limitations: We tested only two datasets and three models, with two repeats per configuration; loss scales differ by architecture, so we compare relative changes within each model. Results may vary with more runs, different hyperparameters, or other architectures (e.g., Transformer-based).

7.1 Future Work

As it stands, the current setup can be improved. It's possible to add more learning robustness methods, and test more different combinations, or run them on more repetitions. Although each of them drastically affects the time it takes to get results, which is the main reason that no further improvements were made. On top of that, it is possible to do hyperparameter tuning, as even though reasonable values are already used, it can help make the research more believable if previous validation of the hyperparameters is done. On top of that, it's possible to improve the current setup by doing partitioning of the dataset, to see

if using an identical dataset, with just half the data would affect the results.

Overall, there is no one-size-fits-all. Tailor robustness techniques to your dataset's baseline performance and the specific model: LeNet and CNN+GRU tended to benefit most from stability training on moderate baselines, while LSTM showed its best gains under coupled weight decay or a broader combination of methods. On highly saturated tasks like NTU-Fi, apply these methods conservatively or skip some to avoid hurting an already strong baseline.

8 Responsible Research

8.1 Ethics

Since for this research topic we only use data that is open source, and thus freely available to everyone, there is no need to ensure that any data is anonymized or kept secret.

8.2 CoC

In line with commonly accepted standards of academic integrity, we have adhered to the principles of the Netherlands Code of Conduct for Research Integrity, with particular emphasis on ensuring reproducibility and transparency in this work.

8.3 LLMs

We used LLMs only to spot typos, fix phrasing inconsistencies, maintain a formal tone, and offer general feedback on clarity. They were not asked to generate hypotheses, analyze data, or draw conclusions. Every suggestion was carefully reviewed and approved by the authors before any changes. We stayed aware that LLM outputs can be imperfect or biased, so we kept full human oversight. This ensured that all substantive content and scientific integrity remained the authors' responsibility, while still benefiting from automated help in proofreading and style.

8.4 Reproducibility

To ensure the reproducibility of this setup, we make sure to explicitly state any and all libraries, datasets, codebases and hypervalues used in this research, as well as extra bits of code that are deemed to improve the reproducibility of the experiment.

8.4.1 SenseFi codebase + modifications

As the basis for our code, the SenseFi codebase was used https://github.com/xyanchen/WiFi-CSI-Sensing-Benchmark and was further modified as described in the methodology. At the moment of writing this (June 2025) the most recent version of the SenseFi codebase is used, and since the research is concluded it is highly unlikely that the codebase will change. All code related to the implementation of the methodology can be found in Appendix A.

8.4.2 Reproducibility

The following seeds were used for making sure that the results could be reproduced if necessary: 43889, 41225 The implementation of the function that ensured that the seeds were used can be found in Appendix A.

8.4.3 Environment

Here are the specifications of the system that the code was run on:

- System + Hardwark
  - OS: Ubuntu 24.04.2 LTS
  - kernel version: Linux 6.11.0-26-generic
  - GPU: NVIDIA GeForce RTX™ 3060 Laptop GPU
  - CPU: 12th Gen Intel® Core™ i7-12700H × 20
  - RAM: 16GB
- Python environment
  - Python version: 3.10.18
  - Pip version: 23.0.1
  - Libraries used:
    - numpy 2.2.6 used for manipulating values in lists
    - tocrh 2.7.1 used for training the model
    - scipy 1.15.3 (was used for calculating statistics)
    - scikit-learn 1.7.0 formerly sklearn. Was used for the implementation of the k-fold cross validator and calculating the F1 score.

A Appendix A: Learning robustness code implementation

```
# Replace your criterion
class LabelSmoothingCrossEntropy(nn.Module):
    def __init__(self, smoothing=0.1):
        super().__init__()
        self.smoothing = smoothing

    def forward(self, pred, target):
        n_class = pred.size(1)
        one_hot = torch.zeros_like(pred).scatter(1, target.view(-1, 1)
            ↪ , 1)
        one_hot = one_hot * (1 - self.smoothing) + self.smoothing /
            ↪ n_class
        return torch.mean(torch.sum(-one_hot * torch.log_softmax(pred,
            ↪  1), 1))
```

```
def set_seed(seed):
    """Set seed for reproducibility across all random number
       ↪ generators."""
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    # Make CuDNN deterministic
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

def mixup_data(inputs, labels, generator, alpha=alphaNum):
    '''Returns mixed inputs, pairs of targets, and lambda'''
    if alpha > 0:
        beta_dist = Beta(alpha, alpha)
        lam = beta_dist.sample().item()  # uses global torch generator
    else:
        lam = 1

    batch_size = inputs.size(0)
    index = torch.randperm(batch_size, generator=generator).to(inputs.
       ↪ device)

    mixed_inputs = lam * inputs + (1 - lam) * inputs[index, :]
    labels_a, labels_b = labels, labels[index]
    return mixed_inputs, labels_a, labels_b, lam


def train(model, tensor_loader, num_epochs, learning_rate, crt, device
    ↪ , stabi, dataug, l2, early, seed, save_path, resume=False):

    model = model.to(device)
    # Use mixed precision for speed (if you have a modern GPU)
    scaler = torch.amp.GradScaler('cuda')

    if l2:
        optimizer = torch.optim.Adam(model.parameters(), lr=
           ↪ learning_rate, weight_decay=1e-4)
    else:
        optimizer = torch.optim.Adam(model.parameters(), lr=
           ↪ learning_rate)

    # Try to resume from checkpoint
    start_epoch = 0
    best_loss = float('inf')
    no_improve_count = 0

    if resume:
        checkpoint_path = save_path / 'checkpoint_latest.pth'
```

```
    checkpoint_info = load_model_checkpoint(model, optimizer,
        ↪ checkpoint_path)
    if checkpoint_info:
        start_epoch = checkpoint_info['epoch'] + 1
        best_loss = checkpoint_info.get('loss', 0.0)
        print(f"Resuming from epoch {start_epoch}, best loss: {
            ↪ best_loss:.4f}")

# Create a generator for reproducible random operations during
    ↪ training
# Note: We set this once at the beginning, not during training
    ↪ loop
generator = torch.Generator()
generator.manual_seed(seed)

for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0
    epoch_accuracy = 0
    epoch_total = 0
    for data in tensor_loader:
        inputs, lbs = data
        inputs = inputs.to(device, non_blocking=True)
        lbs = lbs.type(torch.LongTensor).to(device, non_blocking=
            ↪ True)

        optimizer.zero_grad()
        with torch.amp.autocast('cuda'):
            # FIXED: Apply data augmentation BEFORE forward pass
            if dataug:
                inputs, lbs_a, lbs_b, lam = mixup_data(inputs, lbs
                    ↪ , alpha=alphaNum, generator=generator)
                outputs = model(inputs)  # Forward pass with mixed
                    ↪  inputs
                ce_loss = lam * crt(outputs, lbs_a) + (1 - lam) *
                    ↪ crt(outputs, lbs_b)
            else:
                outputs = model(inputs)
                ce_loss = crt(outputs, lbs)

            # Stability training (after stabilityStart portion of
                ↪ training)
            if epoch > int(stabilityStart * num_epochs) and stabi:
                # Use original inputs for stability training, not
                    ↪ mixed inputs
                if dataug:
                    # Need to use original inputs for stability,
                        ↪ not mixed
                    orig_inputs = data[0].to(device, non_blocking=
                        ↪ True)
                    prt_inputs = orig_inputs + torch.randn_like(
                        ↪ orig_inputs) * noiseAmount
```

```
                    orig_outputs = model(orig_inputs)
                    prt_outputs = model(prt_inputs)
                    stability_loss = torch.nn.functional.mse_loss(
                        ↪ orig_outputs, prt_outputs)
                else:
                    prt_inputs = inputs + torch.randn_like(inputs)
                        ↪  * noiseAmount
                    prt_outputs = model(prt_inputs)
                    stability_loss = torch.nn.functional.mse_loss(
                        ↪ outputs, prt_outputs)

                loss = ce_loss + stability * stability_loss
            else:
                loss = ce_loss

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        epoch_loss += loss.item() * inputs.size(0)
        predict_y = torch.argmax(outputs, dim=1).to(device)
        if dataug:
            # For mixup, we'll use the primary labels for accuracy
                ↪  calculation
            epoch_accuracy += (predict_y == lbs_a.to(device)).sum
                ↪ ().item()
        else:
            epoch_accuracy += (predict_y == lbs.to(device)).sum().
                ↪ item()
        epoch_total += lbs.size(0) if not dataug else lbs_a.size
            ↪ (0)

    epoch_loss = epoch_loss/len(tensor_loader.dataset)
    epoch_accuracy = epoch_accuracy/epoch_total

    improvement_threshold = 1e-4  # Require meaningful improvement
    if early:
        # Training loss usually plateaus/increases when
            ↪ overfitting starts
        if epoch_loss < (best_loss - improvement_threshold):
            best_loss = epoch_loss
            save_model_checkpoint(model, optimizer, epoch,
                ↪ epoch_loss, epoch_accuracy, best_loss,
                ↪ save_path, True)
            no_improve_count = 0
        else:
            no_improve_count += 1
            if no_improve_count >= 10:  # Stop after at least 5
                ↪ epochs without regular improvement
                print("achieved peak loss!!!")
                break
    else:
```

```
            if epoch_loss < best_loss:
                best_loss = epoch_loss
                save_model_checkpoint(model, optimizer, epoch,
                    ↪ epoch_loss, epoch_accuracy, best_loss,
                    ↪ save_path, True)

        if epoch == 0 or (epoch+1) % 40 == 0:
            print('Epoch:{}, Accuracy:{:.4f},Loss:{:.9f}'.format(epoch
                ↪ +1, float(epoch_accuracy), float(epoch_loss)))

    save_model_checkpoint(model, optimizer, epoch, epoch_loss,
        ↪ epoch_accuracy, best_loss, save_path, False)
    print('Final Epoch:{}, Accuracy:{:.4f},Loss:{:.9f}'.format(epoch
        ↪ +1, float(epoch_accuracy), float(epoch_loss)))

    # At the end, load the best saved model
    best_checkpoint_path = save_path / 'model_best.pth'
    if best_checkpoint_path.exists():
        checkpoint = torch.load(best_checkpoint_path)
        model.load_state_dict(checkpoint['model_state_dict'])
        print(f"Loaded best model from epoch {checkpoint['epoch']}")
        print(f"The model has the f1_score: {checkpoint['f1_score']},
            ↪ acc: {checkpoint['accuracy']} and loss: {checkpoint['
            ↪ loss']}")
    return model
```

A Appendix B: F1 scores

A.1 Widar

Table 3: Performance of each method on the Widar dataset. Values are mean F1-score ± standard deviation.

Method	**LeNet**	**LSTM**	**CNN+GRU**
Baseline	0.6149 ± 0.0049	0.5035 ± 0.0112	0.4597 ± 0.0220
Stability training	0.6510 ± 0.0098	0.5065 ± 0.0079	0.4782 ± 0.0402
Mixup	0.6189 ± 0.0184	0.5220 ± 0.0081	0.3967 ± 0.0307
Weight Decay	0.6308 ± 0.0084	0.5505 ± 0.0051	0.3988 ± 0.0134
Early Stopping	0.6299 ± 0.0067	0.5075 ± 0.0102	0.4711 ± 0.0268
Label Smoothing	0.6171 ± 0.0095	0.4707 ± 0.0059	0.4844 ± 0.0365
Generalization	0.6287 ± 0.0078	0.5311 ± 0.0134	0.4071 ± 0.0289
Stable MixUp	0.6237 ± 0.0163	0.5340 ± 0.0073	0.4387 ± 0.0272
Stable Stopping	0.6500 ± 0.0113	0.5056 ± 0.0098	0.4633 ± 0.0443
Mixed Trio	0.6362 ± 0.0095	0.5503 ± 0.0158	0.3917 ± 0.0221
Full Combination	0.6408 ± 0.0139	0.6100 ± 0.0061	0.3373 ± 0.0219

A.2 NTU-Fi

Table 4: Performance of each method on the NTU-Fi dataset. Values are mean F1-score ± standard deviation.

Method	**LeNet**	**LSTM**	**CNN+GRU**
Baseline	0.9950 ± 0.0038	0.9962 ± 0.0054	0.9764 ± 0.0134
Stability Training	0.9962 ± 0.0031	0.9962 ± 0.0046	0.9845 ± 0.0162
Mixup	0.9967 ± 0.0038	0.9867 ± 0.0096	0.9672 ± 0.0152
Weight Decay	0.9942 ± 0.0049	0.9979 ± 0.0029	0.9541 ± 0.0165
Early Stopping	0.9954 ± 0.0060	0.9983 ± 0.0029	0.8130 ± 0.2675
Label Smoothing	0.9979 ± 0.0029	0.9967 ± 0.0038	0.9581 ± 0.0124
Generalization	0.9983 ± 0.0029	0.9925 ± 0.0085	0.8820 ± 0.0981
Stable Mixup	0.9962 ± 0.0041	0.9950 ± 0.0043	0.9336 ± 0.0251
Stable Stopping	0.9937 ± 0.0035	0.9912 ± 0.0077	0.8044 ± 0.1031
Mixed Trio	0.9874 ± 0.0174	0.9433 ± 0.0253	0.8607 ± 0.1465
Full Combination	0.9962 ± 0.0050	0.9653 ± 0.0155	0.7585 ± 0.3014

References

[1] M. Al-qaness and F. Li, "Wiger: Wifi-based gesture recognition system," *International Journal of Geo-Information (IJGI)*, vol. 5, p. 92, 06 2016.

[2] W. Wang, F. Nikseresht, V. G. Rajan, J. Gao, and B. Campbell, "Enabling ubiquitous occupancy detection in smart buildings: A wifi ftm-based approach," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, 2023, pp. 256–260.

[3] X. Wang, C. Yang, and S. Mao, "On csi-based vital sign monitoring using commodity wifi," *ACM Trans. Comput. Healthcare*, vol. 1, no. 3, 2020. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3377165

[4] A. Safonova, G. Ghazaryan, S. Stiller, M. Main-Knorn, C. Nendel, and M. Ryo, "Ten deep learning techniques to address small data problems with remote sensing," *International Journal of Applied Earth Observation and Geoinformation*, vol. 125, p. 103569, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156984322300393X

[5] I. H. Rather, S. Kumar, and A. H. Gandomi, "Breaking the data barrier: a review of deep learning techniques for democratizing ai with small datasets," *Artificial Intelligence Review*, vol. 57, no. 9, p. 226, 2024. [Online]. Available: https://doi.org/10.1007/s10462-024-10859-3

[6] L. Alzubaidi, J. Bai, A. Al-Sabaawi, J. Santamaría, A. S. Albahri, B. S. N. Al-dabbagh, M. A. Fadhel, M. Manoufali, J. Zhang, A. H. Al-Timemy, Y. Duan, A. Abdullah, L. Farhan, Y. Lu, A. Gupta, F. Albu, A. Abbosh, and Y. Gu, "A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications," *Journal of Big Data*, vol. 10, no. 1, 2023.

[7] S. Tang, R. Gong, Y. Wang, A. Liu, J. Wang, X. Chen, F. Yu, X. Liu, D. Song, A. Yuille, P. H. S. Torr, and D. Tao, "Robustart: Benchmarking robustness on architecture design and training techniques," 2022. [Online]. Available: https://arxiv.org/abs/2109.05211

[8] S. Tan, J. Yang, and Y. Chen, "Enabling fine-grained finger gesture recognition on commodity wifi devices," *IEEE Transactions on Mobile Computing*, vol. PP, pp. 1–1, 12 2020.

[9] Y. Zhang, Y. Zheng, K. Qian, G. Zhang, Y. Liu, C. Wu, and Z. Yang, "Widar3.0: Zero-effort cross-domain gesture recognition with wi-fi," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 8671–8688, 2022.

[10] J. Yang, X. Chen, D. Wang, H. Zou, C. X. Lu, S. Sun, and L. Xie, "Sensefi: A library and benchmark on deep-learning-empowered wifi human sensing," *Patterns*, vol. 4, no. 3, 2023.

[11] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4480–4488.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[13] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2018. [Online]. Available: https://arxiv.org/abs/1710.09412

[14] F. Wang, J. Feng, Y. Zhao, X. Zhang, S. Zhang, and J. Han, "Joint activity recognition and indoor localization with wifi fingerprints," *IEEE Access*, vol. 7, pp. 80 058–80 068, 2019.

[15] L. Sharma, C.-H. Chao, S.-L. Wu, and M.-C. Li, "High accuracy wifi-based human activity classification system with time-frequency diagram cnn method for different places," *Sensors*, vol. 21, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/11/3797

[16] A. M. Khalili, A.-H. Soliman, M. Asaduzzaman, and A. Griffiths, "Wi-fi sensing: Applications and challenges," 2023. [Online]. Available: https://arxiv.org/abs/1901.00715

[17] D. Varga, "Mitigating data leakage in a wifi csi benchmark for human action recognition," *Sensors*, vol. 24, no. 24, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/24/8201

[18] ——, "Critical analysis of data leakage in wifi csi-based human action recognition using cnns," *Sensors*, vol. 24, no. 10, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/10/3159

[19] M. Bayle, G. Blanchard, S. Clémençon, and C. Houdré, "Cross-validation confidence intervals for test error," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[20] R. Bey, R. Goussault, F. Grolleau, M. Benchoufi, and R. Porcher, "Fold-stratified cross-validation for unbiased and privacy-preserving federated learning," *Journal of the American Medical Informatics Association*, vol. 27, no. 8, p. 1244–1251, Jul. 2020. [Online]. Available: http://dx.doi.org/10.1093/jamia/ocaa096

[21] Z. Qiu, Y. Chen, and Y. Wu, "A comprehensive review of cross-validation techniques in machine learning," *Information Fusion*, vol. 97, p. 101933, 2024.

[22] B. Efron and R. T. and, "Improvements on cross-validation: The 632+ bootstrap method," *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560, 1997.

[23] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 14, no. 2, 1995, pp. 1137–1145.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[25] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, J. Moody, S. Hanson, and R. Lippmann, Eds., vol. 4. Morgan-Kaufmann, 1991. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf

[26] L. Prechelt, *Early Stopping - But When?* Springer Berlin Heidelberg, 1998, pp. 55–69. [Online]. Available: https://doi.org/10.1007/3-540-49430-8_3

[27] Z. Wang, J. Li, W. Wang, Z. Dong, Q. Zhang, and Y. Guo, "Review of few-shot learning application in csi human sensing," *Artificial Intelligence Review*, vol. 57, no. 8, 2024.

[28] Z. Ma and K. Shi, "Few-shot learning for wifi fingerprinting indoor positioning," *Sensors*, vol. 23, no. 20, p. 8458, 2023.

[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015. [Online]. Available: https://arxiv.org/abs/1412.6572

[30] Y. Pan, Z. Zhou, W. Gong, and Y. Fang, "Sat: A selective adversarial training approach for wifi-based human activity recognition," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 12 706 – 12 716, 2024.

[31] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?" 2020. [Online]. Available: https://arxiv.org/abs/1906.02629

[32] L. Alzubaidi, J. Zhang, A. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, 03 2021.

[33] F. Croce, M. Andriushchenko, V. Sehwag, E. Debenedetti, N. Flammarion, M. Chiang, P. Mittal, and M. Hein, "Robustbench: a standardized adversarial robustness benchmark," 2021. [Online]. Available: https://arxiv.org/abs/2010.09670

[34] C. Liu, Y. Dong, W. Xiang, X. Yang, H. Su, J. Zhu, Y. Chen, Y. He, H. Xue, and S. Zheng, "A comprehensive study on robustness of image classification models: Benchmarking and rethinking," 2023. [Online]. Available: https://arxiv.org/abs/2302.14301