Towards a Faster GNSS Interference Classification: a GRU-Based Approach using Spectrograms

Towards a Faster GNSS Interference Classification: a GRU-Based Approach using Spectrograms

Iman Ebrahimi Mehr[1,2], Gianfranco Caputo[2], Dario Salza[2], Maurizio Fantino[2], Fabio Dovis[1]
[1]*Politecnico di Torino*, Turin, Italy
[2]*Fondazione LINKS*, Turin, Italy

***Abstract*—Global Navigation Satellite Systems (GNSS) are critical for Positioning, Navigation, and Timing (PNT) services, but their signals are highly vulnerable to intentional jamming. Traditional interference detection methods, such as spectral analysis and statistical thresholding, struggle to adapt to dynamic interference conditions. Recent advances in Artificial Intelligence (AI) using deep learning techniques, have enabled more robust classification approaches. This study proposes a Gated Recurrent Unit (GRU)-based framework for GNSS interference classification, offering a more lightweight alternative to convolutional neural networks (CNNs), which rely on spectrogram images as inputs. The GRU model directly processes sequential spectrograms, enabling efficient real-time classification with a reduced computational overhead. Experimental validation using datasets collected in a controlled environment, along with an implementation on a low-cost embedded device, demonstrated high classification accuracy and improved efficiency over CNN-based approaches. The findings highlight the potential of AI-driven methods for scalable and real-time GNSS interference monitoring, particularly in resource-constrained environments.**

***Index Terms*—GNSS, Monitoring, Interference, jamming, Classification, Machine Learning, Software Defined Radio, Embedded Systems.**

I. Introduction

The increasing dependence on Global Navigation Satellite Systems (GNSS) for Positioning, Navigation, and Timing (PNT) services has raised concerns over their vulnerability to interference. These signals, essential for applications ranging from telecommunications and transportation to financial transactions and power grid synchronization, are highly susceptible to intentional jamming and unintentional radio frequency interference (RFI). Such disturbances can degrade positioning accuracy below the minimum standards outlined in the respective Service Document Definitions (SDD) or even disrupt signal reception entirely, posing risks to both civilian and critical infrastructure operations [1]. Consequently, there has been a growing research interest in interference detection and classification to enhance system reliability and support the development of effective interference mitigation strategies.

As jamming technology becomes more accessible by the increased availability of low-cost jamming devices, identification of countermeasures to interference in GNSS bands has become a pressing topic. Particularly, chirp jammers, characterized by their frequency-sweeping nature, present a significant challenge in detection and classification. Conventional methods for interference identification often rely on statistical thresholding and spectral analysis of GNSS signals. However, these techniques frequently struggle to adapt to dynamic interference conditions and may not provide sufficient robustness for real-time applications.

Recent advances in artificial intelligence and machine learning (ML) have opened new avenues for interference classification. Deep learning models, especially those utilizing Time-Frequency Representations (TFRs), have demonstrated superior detection capabilities. Time-frequency analysis (TFA) has also been recognized as a powerful tool for interference detection and system monitoring. The work presented in [2] highlights how TFRs such as spectrograms, wavelet transforms, and Wigner-Ville distributions enhance interference classification by capturing non-stationary signal characteristics. Convolutional Neural Networks (CNNs) have been extensively used in this domain, as they effectively analyze spectral patterns from transformed signal data. However, generating spectrogram-based inputs for CNNs introduces a substantial computational burden, adding an extra preprocessing step that makes deployment on embedded systems more time-consuming.

To address these limitations, this study proposes a Gated Recurrent Unit (GRU)-based classification framework, which directly leverages the sequential nature of TFRs, eliminating the need for additional preprocessing steps. The GRU model is designed to efficiently process interference signals in real-time while preserving computational efficiency. By continuously analyzing the evolving frequency characteristics of interference, the proposed approach ensures faster classification and lower resource consumption, making it particularly suitable for embedded GNSS monitoring systems. This methodology operates at the raw signal level (In-phase/Quadrature (IQ) signal samples), which is crucial, especially when mitigation methods must be applied directly to raw signals. To have a fair comparison, this proposed method and three other CNN-based approaches have been deployed into an embedded device for evaluation of their performance.

The remainder of this paper is structured as follows: Section I-A reviews relevant literature, Section II provides a detailed explanation of the proposed methodology and the fundamentals of GRU. Subsequently, Section III describes the experimental setup and the datasets used to evaluate the model, and the implementation of the model on an embedded device. Section IV examines the performance of a GNSS receiver under various jamming scenarios with different power levels. Section V presents the performance evaluation results and

compares the GRU-based approach with CNN alternatives. Finally, we discuss the practical implications of this work and its potential integration into real-world GNSS interference detection frameworks.

A. Related Works

Various machine learning approaches have been explored for GNSS interference detection and classification, leveraging both pre-correlation and post-correlation data processing techniques. Several studies have utilized pre-correlation features, focusing on raw IQ samples or their transformations. Some approaches apply feature extraction from IQ samples, where machine learning classifiers such as Support Vector Machines (SVM), Random Forest, and k-Nearest Neighbors (k-NN) have been used for classification [3]–[8]. To enhance performance, improved optimization techniques such as genetic algorithms have been applied [4]. A different approach involves spectrogram-based analysis, where Convolutional Neural Networks (CNNs) have been extensively employed due to their ability to process image-like data structures efficiently [9]–[15]. Some studies explore alternative representations, such as scalograms derived from wavelet transforms [11] or concatenated images combining multiple signal characteristics, including power spectral density (PSD), histograms, and constellation diagrams [16]. Additionally, hybrid methods combining CNNs with other classifiers like SVM and logistic regression have been proposed to improve classification robustness [16].

Other studies explore post-correlation-based approaches, relying on GNSS receiver observables such as carrier-to-noise density ratio (C/N_0), tracking loop outputs, and satellite elevation. Recurrent neural networks (RNNs), including BiLSTM and U-Net-based architectures, have been applied to analyze these temporal dependencies [17]–[20]. Additionally, autoencoder-based methods combining pre- and post-correlation features have been explored to enhance classification accuracy [21].

A few studies focus on time-series-based signal analysis, where deep learning architectures like LSTMs and Transformers process sequential features such as signal entropy and instantaneous frequency variations [22]. Transformers and CNNs have also been combined for feature extraction from spectrogram and PSD representations [12].

Additionally, deep learning-based hybrid models have been explored for GNSS signal processing and prediction tasks. The work in [23] introduces a CNN-GRU hybrid approach for GNSS deformation monitoring, demonstrating the benefits of combining spatial feature extraction from CNNs with temporal sequence modeling from GRUs.

Despite significant advancements, challenges remain in optimizing computational efficiency and real-time performance, particularly for embedded applications. Our study builds upon these prior works by implementing a GRU-based approach that processes spectrogram representations while minimizing inference latency and computational costs, making it suitable for real-time GNSS monitoring.

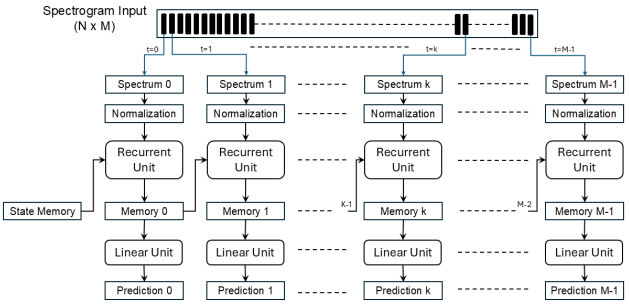


Fig. 1: Proposed methodology

II. METHODOLOGY

Since most jamming signals exhibit variations in their instantaneous frequency over time (commonly referred to as chirp signals), a time-frequency representation (TFR) is an appropriate tool to analyze and understand their behavior. In this research, the Short-Time Fourier Transform (STFT) was chosen for TFR due to its ability to effectively handle non-stationary signals and its lower computational complexity. The STFT employs a fixed-length window function to segment the signal into shorter frames, where the Fourier transform is computed separately for each frame. This approach ensures that frequency variations are captured while maintaining sufficient time resolution, making it well-suited for detecting and analyzing chirp signals [24].

The resulting TFR is represented as a data matrix containing the sequence of spectral components across time. This matrix can be directly fed into a recurrent neural network (RNN), such as a Gated Recurrent Unit (GRU) or Long-Short Term Memory (LSTM) network. These architectures are designed to process sequential data by consuming one element at a time and updating an internal memory representation of the sequence observed so far. This memory enables the model to learn temporal dependencies and extract meaningful patterns from the data.

For this study, the GruNet model was selected due to its ability to effectively process sequential data and capture temporal dependencies, which are critical for analyzing non-stationary signals such as jamming signals. A many-to-many approach is employed, where the memory representation is passed at each step into one or more linear layers to make predictions. The recursive model infers both the presence and the type of jamming interference based on the sequence of spectra processed. This methodology allows for robust classification of jamming signals and provides insights into their temporal dynamics.

The GruNet architecture is built around a GRU, which offers an efficient and robust framework for modeling time-series data. The concept block diagram of the proposed method depicted in Fig. 1. The GRU processes input sequences of a predefined dimensionality (128, corresponding to the number

of FFT points in each analysis window) using a hidden state with 256 hidden units. To enhance temporal modeling capacity, the model was designed with two configurations analyzed: one and two stacked GRU layers. After the GRU processes the input, the output undergoes a Parametric Rectified Linear Unit (PReLU) activation, introducing non-linearity and allowing the model to adaptively learn activation parameters. The final layer is a fully connected linear layer, which maps the GRU's output to six output classes, corresponding to the different types of jamming signals.

The input data to the model is the STFT calculated over 100 microseconds of IQ signal samples, employing 128 FFT points in each analysis window, which provides detailed frequency resolution. To ensure smooth transitions and continuity between consecutive analysis windows, an overlap of 99% (approximately 127 samples) is used. The input is first normalized to ensure numerical stability and consistency, involving a log transformation followed by standardization using the mean and standard deviation computed during the initial training forward pass. The GRU processes the normalized input, initializing its hidden states as zeros for each sequence. The model's output, which captures temporal features from the input data, is passed through the PReLU activation function and the final linear layer to produce predictions.

In this research, we compared the performance of the proposed model (GruNet) with a CNN-based approach. The implementation of the CNN-based model is fully aligned with the methodology proposed in [25] [26], with the only modification being the replacement of the CNN architectures. While the original work focused on the ResNet model, we further analyzed MobileNet and EfficientNet in addition to ResNet. MobileNet and EfficientNet were selected due to their demonstrated performance on the ImageNet dataset. ResNet is well-known for its high accuracy in classification tasks, whereas MobileNet and EfficientNet were chosen for their superior efficiency in terms of inference time, making them more suitable for deployment on resource-constrained systems.

A. GRU Fundamentals and Mathematical Framework

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. [27]. The GRU is similar to the long short-term memory (LSTM) model [28], as both use gating mechanisms to input or forget certain features [29]. However, GRUs lack a context vector or output gate, which results in fewer parameters compared to LSTMs. GRUs have demonstrated comparable performance to LSTMs in tasks such as polyphonic music modeling, speech signal modeling, and natural language processing. Studies have shown that gating mechanisms significantly enhance performance, although no consensus exists on whether GRUs or LSTMs are superior [30], [31].

The GRU operates by updating its hidden state h_t at each time step t, based on the input x_t and the previous hidden state h_{t-1}. It achieves this through two gates: the update gate z_t and the reset gate r_t. These gates control the flow of information within the GRU, allowing it to retain relevant information and forget irrelevant or outdated data. The equations governing the GRU are as follows:

1) Reset Gate (r_t): Determines how much past information to forget.

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{1}$$

where σ is the sigmoid activation function. The reset gate determines how much of the previous hidden state information to forget.

2) Update Gate (z_t): controls the balance between retaining the previous hidden state and incorporating new candidate information.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2}$$

3) Candidate Hidden State ($\tilde{h}_t$): Represents a new candidate for the current hidden state.

$$\tilde{h}_t = \tanh(W_h x_t + r_t \odot (U_h h_{t-1}) + b_h) \tag{3}$$

Here, $\odot$ represents the element-wise multiplication. The reset gate r_t selectively determines the contribution of the past hidden state h_{t-1}.

4) Final Hidden State (h_t): The final output of the GRU cell for the current time step.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{4}$$

The update gate z_t combines the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$ to produce the final hidden state h_t.

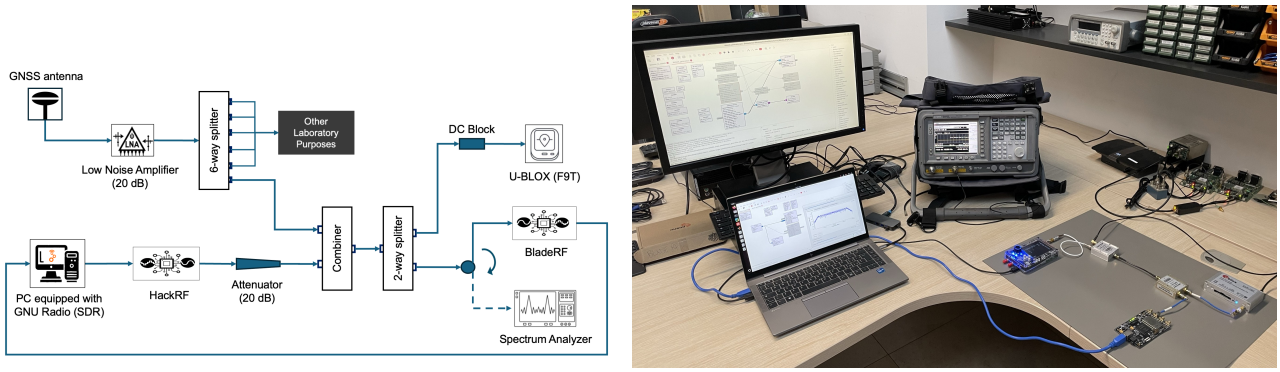
The GRU employs two primary activation functions. The first is the sigmoid function ($\sigma(x)$), which maps input values to a range between 0 and 1. This function is used in the reset and update gates to control the flow of information by compressing values into this range. The second is the hyperbolic tangent function ($\tanh(x)$), which maps input values to a range between -1 and 1. This function is used for computing the candidate hidden state, allowing the GRU to effectively represent both positive and negative values. Compared to LSTMs, which also use gating mechanisms, GRUs are simpler due to the absence of an output gate. This reduction in complexity makes GRUs computationally more efficient, although their performance may vary depending on the task.

III. Data Gathering and Practical Implementation

A. Laboratory Testbed for Data Acquisition

The laboratory testbed was designed to simulate controlled environments for generating, capturing, and analyzing GNSS signals under jamming conditions. Figure 2 provides an overview of the experimental setup.

The GNSS signal path begins with a Calibrated Survey GNSS Triple-band antenna (IP67 - ARDESIMPLE). The antenna is powered by a low noise amplifier (LNA), which amplifies the signal by 20 dB. The amplified signal is then routed to



(a) (b)

Fig. 2: Laboratory testbed. Schematic Diagram of the Experimental GNSS Jamming Testbed (a) and Laboratory Setup for GNSS Jamming Experiments (b).

a 6-way splitter, distributing the signal to various components and devices across the laboratory. The channel power of the GNSS signal, after passing through all the components shown in the schematic (Figure 2a), was measured using a spectrum analyzer. The measurement indicated a signal strength of -80.60 dBm over a 10 MHz bandwidth, corresponding to a power spectral density of -150.60 dBm/Hz. Note that the noise floor of the spectrum analyzer in this testbed is -94.70 dBm over 10 MHz or -164.70 dBm/Hz.

The jamming signal was generated using the software-defined radio (SDR) paradigm. Pre-generated binary files, representing various types of jamming signals, were transmitted using a HackRF SDR (hackrf one portapack). These jamming signals belong to the chirp family and include Linear Wideband Fast (LWF), Linear Narrowband (LN), Triangular (TRI), Triangular Wave (TW), and Tick-shaped (TICK) signals. Comprehensive details regarding the behavior and characteristics of these specific jamming signals are provided in [25] and [5]. These binary files were created with a MATLAB script, featuring a sampling rate of 20 MHz, 16-bit quantization, and a duration of 100 milliseconds. The HackRF device communicates with the computer via the OSMOCOM library, ensuring seamless control and signal transmission. To simulate continuous jamming, the HackRF repeatedly transmitted the same binary file in a loop. To prevent leakage of unwanted signal power from the HackRF output, even when the transmission gain is set to zero, a 20 dB attenuator was placed at the output. For controlling the power of the transmitted jamming signal, the IF gain (LNA gain) was used, while the RF gain was fixed at zero. This setup allowed precise control over the signal power while ensuring minimal signal leakage.

The GNSS and jamming signals are combined using a signal combiner and further split using a 2-way splitter to facilitate multiple outputs for simultaneous testing of various devices. The devices under test include a U-BLOX F9T GNSS receiver and a BladeRF 2.0 micro (Nuand). The BladeRF provides In-phase and Quadrature (IQ) signal samples, which is used for the classification task. The PC equipped with GNU Radio serves as the central hub for signal processing, communicating with the HackRF to transmit the jamming signal and with the BladeRF to receive the GNSS-jammed signal.

B. Dataset

Using the testbed described in Section 3, an IQ dataset of jammed GNSS signals was generated. The dataset comprises six classes, one representing the GNSS signal in the absence of jamming, and five corresponding to the presence of chirp jamming signals. For each class of jammed signal, various jamming power levels were considered, ranging from -85 dBm to -55 dBm in increments of 5 dB, simulating different scenarios where the jammer is either near or far from the tested devices. To ensure that the output power of the HackRF matched the desired levels, the input signal to the BladeRF was measured using a spectrum analyzer at each power level before starting the IQ file recording. The IQ files were recorded with a duration of approximately 10 minutes, a sampling frequency of 10 MHz, and 8-bit quantization for both the real and imaginary parts. Each jamming class comprises multiple files, corresponding to the different jamming power levels described above. To increase the number of training examples, each IQ file was segmented into smaller portions, resulting in 42,000 samples per class. The dataset was then divided into three subsets: 50% was used for training, 15% for validation (to tune hyperparameters and select the best model), and the remaining 35% for unbiased evaluation of the final model. An example of the time-frequency representation using STFT for the clean signal and each jammed IQ signal under –55 dBm jamming power is presented in Figure 3. To support reproducibility and further research, we have published two versions of the dataset described in this study. The first version features a sampling frequency of 10 MHz, while the second version offers a higher sampling frequency of 20 MHz. Both datasets are accessible through the the Zenodo repository [32].

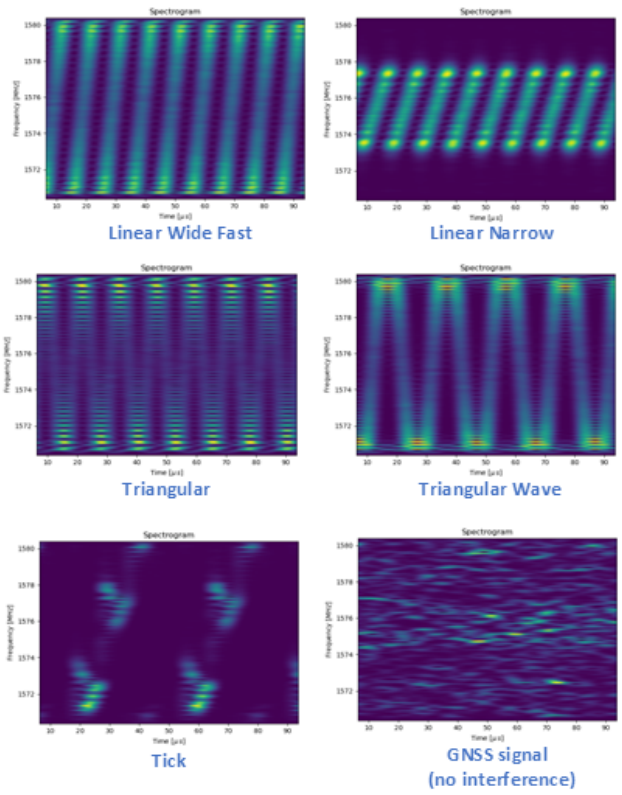


Fig. 3: STFT of clean and jammed IQ signals (–55 dBm of jamming power).

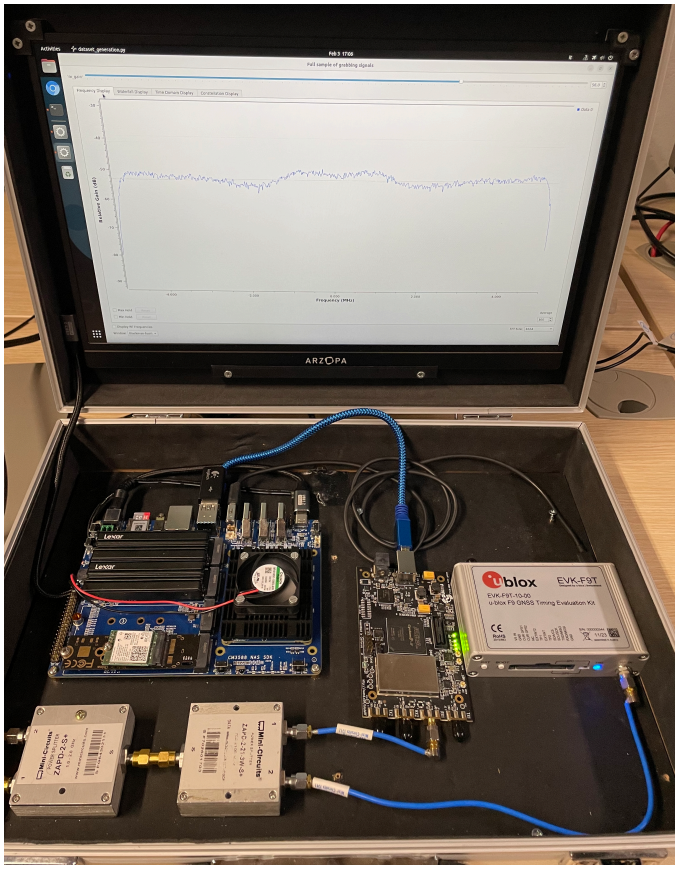


Fig. 4: Prototype of jamming classification

C. Practical Implementation of Models on Embedded Systems

For the embedded implementation of the jamming classification system, a low-cost embedded device with a neural computation unit, the Rockchip CM3588, has been selected as the primary hardware platform. The CM3588 is a high-performance system-on-chip (SoC) designed for edge computing, featuring an 8-core CPU, and an integrated NPU (Neural Processing Unit) for accelerating AI tasks, making it well-suited for real-time signal processing and classification applications. To utilize the hardware's AI capabilities effectively, we leveraged the RKNN-Toolkit, a development framework provided by Rockchip. The RKNN-Toolkit simplifies the deployment of neural network models on Rockchip devices by offering APIs and tools for model optimization, conversion, and inference on the NPU. This toolkit enables efficient execution of AI models with reduced latency and power consumption.

The machine learning models for jamming classification include the GruNet model, developed using the PyTorch library [33], and various CNN models, implemented with TensorFlow (Keras) [34]. To deploy these models on the RK3588, it is necessary to convert them into a binary format compatible with the RKNN-Toolkit. This process involves two steps:

1) Exporting models to `.onnx` Format: The trained models are first exported from PyTorch or TensorFlow into the ONNX (Open Neural Network Exchange) format. ONNX is an open standard for representing machine learning models, enabling interoperability between various deep learning frameworks. It serves as an intermediate representation, simplifying the transition between training and deployment environments.
2) Converting ONNX to RKNN Format: Using the RKNN-Toolkit APIs, the ONNX models are transformed into the `.rknn` format, which is optimized for execution on the Rockchip NPU. While it is possible to directly convert models from TensorFlow's `.keras` format to `.rknn`, this approach requires manually defining input and output sizes and names, which can be time-consuming and error-prone. The ONNX-based workflow provides a more streamlined and flexible solution.

At the hardware level, the laboratory setup has been miniaturized into a portable prototype, housed within a briefcase. In this prototype, the bladeRF output is directly connected to the embedded CM3588 device, replacing the PC used in earlier experiments for the generation of dataset. The embedded device performs real-time classification of GNSS jamming signals using the deployed models. This prototype implementation is illustrated in Figure 4

IV. Evaluating GNSS Receiver Resilience to Jamming Attacks

The purpose of this evaluation is to assess the resilience of GNSS receivers under various jamming conditions through controlled experiments. By simulating different types of jamming signals and varying their intensity, this study provides insights into how these signals affect GNSS receiver performance. This analysis highlights the impact of the signals, which are the focus of our classification task, on real-world GNSS receivers, thereby linking the evaluation to practical scenarios and applications. Key metrics were considered to evaluate the impact of jamming on GNSS receivers:

1) **Channel Power of GNSS and Jamming Signals**: Measured over a 10 MHz bandwidth, this metric quantifies the signal strengths of both the legitimate GNSS signal and the interfering jamming signal. Note that the channel power of the GNSS signal remains constant during the entire experiment and is equal to -80 dBm, while the jamming signal varies from -85 dBm to -55 dBm. The channel power was measured using an averaging window and has a tolerance of ±0.5 dB.
2) **Jamming-to-Signal Ratio (JSR)**: Defined as the difference between the channel power of the jamming signal (P_{J}) and the GNSS signal (P_{S}), the JSR is a critical metric that indicates the relative intensity of interference. Since we have measured both these parameters during the experiment, JSR can be calculated as:

$$\text{JSR (dB)} = P_{\text{J,dBm}} - P_{\text{S,dBm}}$$

3) **Position precision (2D and 3D)**: The precision of receiver position estimations is assessed in both two-dimensional and three-dimensional space. These metrics reflect the degradation in navigation performance caused by jamming.
4) **Automatic Gain Control (AGC)**: AGC measures the percentage of the receiver's gain adjustment, which provides insights into the receiver's internal response to varying signal strengths and interference levels. A high AGC value indicates increased effort by the receiver to maintain signal stability under jamming conditions.

TABLE I: Evaluation Metrics for GNSS Receiver Resilience to Jamming

Jamming Type	Power (dBm)	JSR (dB)	2D Acc. (m)	3D Acc. (m)	AGC (%)
LWF	-85	-5	0.82	1,29	38.6
LWF	-80	0	0.90	1,44	38.6
LWF	-75	5	0.96	1.64	34.3
LWF	-70	10	1.01	1.75	25.7
LWF	-65	15	1.35	2.68	21.4
LWF	-60	20	2.47	4.83	12.9
LWF	-55	25	Blind	Blind	8.6
LN	-85	-5	0.79	1.41	42.9
LN	-80	0	0.82	1.43	38.6
LN	-75	5	0.86	1.43	38.6
LN	-70	10	1.01	1.66	34.3
LN	-65	15	1.27	1.99	21.4
LN	-60	20	3.08	4.37	17.1
LN	-55	25	Blind	Blind	11.5
TRI	-85	-5	0.81	1.36	38.6
TRI	-80	0	0.88	1.49	38.6
TRI	-75	5	0.90	1.60	34.3
TRI	-70	10	1.01	1.81	25.7
TRI	-65	15	1.18	2.20	17.1
TRI	-60	20	1.26	2.55	12.9
TRI	-55	25	1.51	2.84	8.6
TW	-85	-5	0.87	1.50	38.6
TW	-80	0	0.88	1.51	38.6
TW	-75	5	0.95	1.51	34.3
TW	-70	10	1.06	1.71	25.7
TW	-65	15	1.41	2.24	17.1
TW	-60	20	2.44	3.55	12.9
TW	-55	25	Blind	Blind	8.6
TICK	-85	-5	0.88	1.51	38.6
TICK	-80	0	0.89	1.53	38.6
TICK	-75	5	0.92	1.71	34.3
TICK	-70	10	0.93	1.71	25.7
TICK	-65	15	1.36	2.40	21.4
TICK	-60	20	2.11	3.49	12.9
TICK	-55	25	Blind	Blind	8.6

These metrics are instrumental in evaluating the performance degradation caused by different types of jamming signals and provide a foundation for developing strategies to mitigate their effects in real-world GNSS systems.

The evaluation metrics reveal critical insights into the impact of jamming signals on GNSS receiver, particularly as the Jamming-to-Signal Ratio (JSR) increases. A clear trend emerges across all jamming types: at JSR values of 25 dB, the receiver enters a "blind" state, where it can no longer provide position solutions. This result demonstrates the severe effect of strong jamming signals on GNSS performance. As the JSR increases, shown in Figure 5, both 2D and 3D position accuracy deteriorate significantly, and the receiver struggles to maintain reliable navigation. While all waveforms negatively impact the GNSS receiver, the extent of degradation varies between jamming types. Linear waveforms, LWF and LN, demonstrate rapid degradation, with position accuracies collapsing at higher JSR values. Waveforms like triangular and triangular Wave exhibit slightly less severe effects at lower JSR values but converge with other waveforms as the interference becomes stronger. The AGC further reflects the receiver's internal struggle to maintain signal stability under jamming. As the JSR increases, AGC values progressively decrease, demonstrating the receiver's diminishing ability to compensate for stronger jamming signals. For instance as shown in shown in Figure 6,, in the case of LWF jamming, AGC drops from 38.6% at JSR = -5 dB to 9.0% at JSR = 25 dB. This decline signifies that as AGC saturates, the receiver becomes incapable of maintaining signal quality, eventually leading to blind conditions.

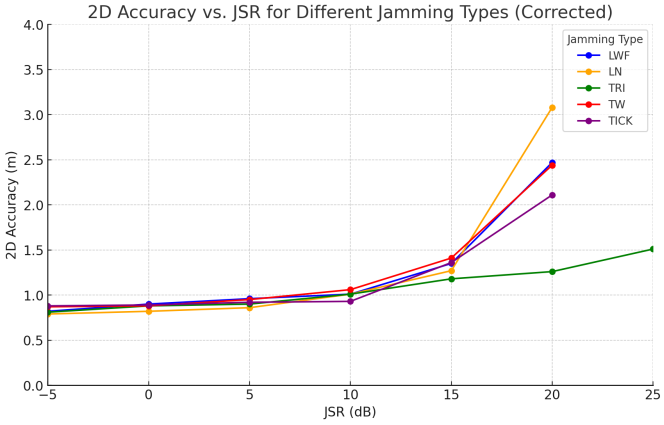


Fig. 5: 2D Accuracy vs. JSR for Different Jamming Types

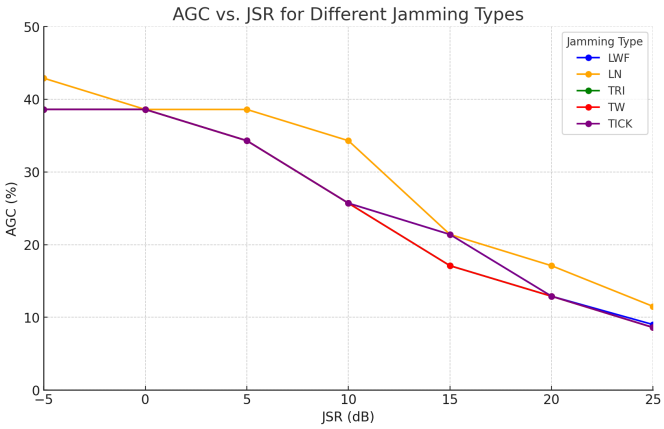


Fig. 6: AGC vs. JSR for Different Jamming Type

V. Experimental Findings and Analysis

This section presents the outcomes of the experiments, which were carried out in accordance with the methodology outlined in Section II and utilizing the datasets described in Section III-B. To ensure a fair and consistent comparison across the different ML models, all experiments were conducted using an identical training dataset and test samples.

A. Predictive Performance of models

This subsection focuses on the evaluation of the predictive capabilities of the AI models, presenting key performance metrics such as accuracy, F1-score, precision, and recall. These metrics provide a comprehensive understanding of each model's performance. Accuracy reflects the overall proportion of correct predictions, while precision measures the proportion of true positive predictions out of all positive predictions made by the model. Recall, on the other hand, quantifies the model's ability to identify all relevant positive instances in the dataset. The F1-score, as a harmonic mean of precision and recall, balances these two metrics to provide a single measure of performance that accounts for both false positives and false negatives [35]. Table II summerize the evaluation performance of proposed methodology comparing to CNN-based approach. As mentioned in Section II, the GruNet model was analyzed with two configurations: one and two stacked GRU layers. Since both showed similar performance, the metrics for the 1-layer GRU (GruNet-1L) are reported here.

TABLE II: Performance Metrics of Different Models

Model	Accuracy (%)	F1-Score	Precision	Recall
GruNet-1L	99.98	99.98	99.98	99.98
ResNet	99.94	99.94	99.95	99.94
EfficientNet	99.02	99.02	99.01	99.03
MobileNet	99.61	99.61	99.62	99.61

The performance comparison of different models, as presented in Table II, shows that GruNet-1L achieves the highest accuracy, precision, recall, and F1-score, all at 99.98%. These results indicate that the model performs well in this classification task, with consistent predictions across different classes, as reflected in the minimal difference between precision and recall. While all models perform well, GruNet-1L demonstrates a small advantage, suggesting that its architectural design, which incorporates recurrent structures, and its simpler model structure compared to CNN-based architectures, may contribute to its performance.

In our analysis, we used a time duration of 100 microseconds to compute the STFT of the IQ signal, which served as the input to the classification model. This choice is motivated by the characteristics of commonly used GNSS jamming signals, such as chirp jammers, whose sweep rates are typically below 50 microseconds. Therefore, a 100-microsecond window is generally sufficient to capture meaningful time-frequency features for classification. It is important to note that every machine learning model is optimized to perform best on the data distribution it was trained with. When the characteristics of the input jamming signals—such as periodicity or sweep rate—differ significantly from the training data, the model's certainty in classification naturally decreases. As a result, a classification model that aims to be robust across a wide variety of jamming types and parameters must be trained (or fine-tuned) on representative samples of those signal classes. To illustrate this, we tested the trained model on previously unseen jamming signals that exhibited different periodicity and waveform structures. In this scenario, the classifier did not provide high-confidence predictions for any specific class (i.e., classification certainty remained below 40%). However, it still indicated the presence of interference, acting more as a jamming detector than a fine-grained classifier. This observation highlights the potential for classification models to serve dual roles depending on the input distribution, providing both identification and coarse detection of anomalous signals.

B. Performance Analysis on Embedded Devices

This subsection provides the results for inference and preprocessing time of each model obtained by deploying the models onto an embedded device explained in Section III-C. To ensure reliable measurements, the results were derived

through a Monte Carlo analysis, where the average values were calculated over 500 iterations. For a fair comparison, identical input samples were used for all models during evaluation. It is important to note that the reported inference times do not include the preprocessing step, which involves reading the binary file and converting raw IQ signal samples into the required input format for the model. As shown in Table III and IV, the preprocessing time for GruNet is 14.17 milliseconds, whereas for CNN-based models, it is 148.50 milliseconds. This difference is primarily due to the additional step required to convert the spectrogram matrix into an image for CNN processing.

TABLE III: Models inference performance using only CPU

Model	Number of trainable parameters	Inference Time (ms)	Preprocessing Time (ms)	Task Completion Time (ms)
GruNet-1L	298,246	64.92	14.17	79.09
ResNet	32,033,828	262.39	148.50	410.89
EfficientNet	10,933,920	134.28	148.50	282.78
MobileNet	9,150,244	80.79	148.50	229.29

The inference performance results, summarized in Table III, further emphasize the efficiency of the proposed GRU-based model. A key observation is the significant difference in the number of trainable parameters between GruNet-1L and the CNN-based architectures. Specifically, GruNet-1L has only 298,246 parameters, which is drastically lower than ResNet (32M parameters), EfficientNet (10.9M parameters), and MobileNet (9.1M parameters). This substantial reduction in model complexity highlights the lightweight nature of the GRU-based approach, making it more suitable for resource-constrained environments.

The CPU inference time has been obtained by deploying the models using ONNX runtime on the embedded device. Since this library does not support the NPU resources available on the device, inference is performed solely on the CPU. Examining the CPU inference times in Table III, GruNet-1L achieves the fastest inference time at 64.92 ms, making it the most efficient model for CPU-based inference. MobileNet follows with 80.79 ms, which is slightly slower than GruNet-1L but still significantly faster than the other CNN-based models. EfficientNet and ResNet exhibit much higher inference times, with 134.28 ms and 262.39 ms, respectively. ResNet, in particular, has the slowest execution on a CPU, likely due to its deeper architecture, which requires more computations per inference. EfficientNet, though optimized for efficiency, still takes more than twice the time of GruNet-1L.

TABLE IV: Models inference performance using NPU and CPU

Model	Inference Time (ms)	Preprocessing Time (ms)	Task Completion Time (ms)
GruNet-1L	83.39	14.17	97.56
ResNet	36.33	148.50	184.83
EfficientNet	28.18	148.50	176.68
MobileNet	12.41	148.50	160.91

Table IV presents the inference time of different models using both NPU and CPU. These times are measured when all the models has been deployed using the RKNN library for inference task, which is optimized for NPU acceleration on the board. However, as seen in the results, the inference time of GruNet-1L is higher compared to CNN-based models. This difference suggests that the execution efficiency of GruNet-1L on the NPU may be influenced by architectural factors, as RKNN is designed with a focus on convolutional operations. In contrast, CNN-based models exhibit significant acceleration on the NPU, with ResNet achieving 36.33 ms on the NPU compared to 262.39 ms on the CPU, highlighting the speedup provided by NPU execution. However, comparing these results reveals that, even though GruNet does not benefit from NPU acceleration, its overall task completion time remains lower than that of CNN-based approaches. This is primarily due to the significantly faster preprocessing step required for GruNet, which compensates for its longer inference time.

VI. Conclusion

This study presented a GRU-based framework for GNSS interference classification, addressing the challenges posed by chirp jammers and other dynamic interference sources. Unlike conventional CNN-based approaches that rely on spectrogram images as inputs, the proposed model directly processes sequential time-frequency representation (TFR) data, reducing computational complexity while maintaining high classification accuracy. Experimental results demonstrated the effectiveness of the GRU model, particularly in embedded environments, where it achieved comparable and superior performance to CNN-based architectures while significantly reducing preprocessing time. Furthermore, an evaluation of GNSS receiver resilience under various jamming scenarios provided insights into the impact of different interference types on navigation performance.

Future work will focus on further optimizing the proposed approach for embedded platforms, with particular emphasis on improving the execution efficiency of GruNet on dedicated NPUs. While initial results indicate that CNN-based models benefit more from current NPU acceleration, investigating optimized implementations tailored to recurrent architectures will be key to maximizing hardware utilization. By leveraging real-time classification outputs, future GNSS receivers could dynamically adapt their signal processing techniques to suppress or compensate for interference effects. This includes the exploration of adaptive filtering, interference excision techniques, and predictive countermeasures based on detected interference patterns. Expanding the classification framework to address a wider range of jamming sources, including multi-source interference scenarios and potential spoofing threats, will further enhance its applicability to real-world operational conditions.

ACKNOWLEGGMENT

The Ph.D. work of I. Ebrahimi Mehr is supported by the grant DOT1332092 CUP E11B21006430005 funded within

the Italian Programma Operativo Nazionale (PON) Ricerca e Innovazione 2014-2020, Asse IV "Istruzione e ricerca per il recupero" con riferimento all'Azione IV.4 "Dottorati e contratti di ricerca su tematiche dell'innovazione" e all'Azione IV.5 "Dottorati su tematiche green" DM 1061/2021. This work was supported by the European Union's Horizon Europe framework programme under the project ATLANTIS with grant agreement No.101073909. This paper is part of the project NODES which has received funding from the MUR – M4C2 1.5 of PNRR funded by the European Union - NextGenerationEU (Grant agreement no.ECS00000036)

REFERENCES

[1] F. Dovis, *GNSS Interference Threats and Countermeasures*, ser. The GNSS technology and application series. Artech House, 2015.

[2] M. G. Amin, D. Borio, Y. D. Zhang, and L. Galleani, "Time-frequency analysis for GNSSs: From interference mitigation to system monitoring," *IEEE Signal Processing Magazine*, vol. 34, no. 5, pp. 85–95, 2017.

[3] J. Xu, S. Ying, and H. Li, "GPS interference signal recognition based on machine learning," *Mobile Networks and Applications*, vol. 25, 12 2020.

[4] J. Shu, Y. Liao, and X. Luan, "An interference recognition method based on improved genetic algorithm," in *2021 7th International Conference on Computer and Communications (ICCC)*, 2021, pp. 496–500.

[5] W. Qin and F. Dovis, "Situational awareness of chirp jamming threats to GNSS based on supervised machine learning," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, no. 3, pp. 1707–1720, 2022.

[6] M. Nicola, G. Falco, R. Morales Ferre, E.-S. Lohan, A. de la Fuente, and E. Falletti, "Collaborative solutions for interference management in GNSS-based aircraft navigation," *Sensors*, vol. 20, no. 15, 2020.

[7] D. Jdidi, T. Brieger, T. Feigl, D. Contreras Franco, J. van der Merwe, A. Rügamer, J. Seitz, and W. Felber, "Unsupervised disentanglement for post-identification of GNSS interference in the wild," in *Proceedings of the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022), Denver, Colorado,*, 09 2022.

[8] Y. Ding and K. Pham, "GNSS interference identification beyond jammer classification," in *2023 IEEE Aerospace Conference*, 2023, pp. 1–8.

[9] R. Ferre, A. Fuente, and E. S. Lohan, "Jammer classification in GNSS bands via machine learning algorithms," *Sensors*, vol. 19, p. 4841, 11 2019.

[10] X. Chen, D. He, X. Yan, W. Yu, and T.-K. Truong, "GNSS interference type recognition with fingerprint spectrum DNN method," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, no. 5, pp. 4745–4760, 2022.

[11] A. Elango, S. Ujan, and L. Ruotsalainen, "Disruptive GNSS signal detection and classification at different power levels using advanced deep-learning approach," in *2022 International Conference on Localization and GNSS (ICL-GNSS)*, 2022, pp. 1–7.

[12] T. Brieger, N. Raichur, D. Jdidi, F. Ott, T. Feigl, J. van der Merwe, A. Rügamer, and W. Felber, "Multimodal learning for reliable interference classification in GNSS signals," *Proceedings of the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022), Denver, Colorado,*, pp. 3210–3234, 10 2022.

[13] P. Wu, H. Calatrava, T. Imbiriba, and P. Closas, "Jammer classification with federated learning," in *2023 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 04 2023, pp. 228–234.

[14] F. Ott, L. Heublein, N. L. Raichur, T. Feigl, J. Hansen, A. Rügamer, and C. Mutschler, "Few-shot learning with uncertainty-based quadruplet selection for interference classification in GNSS data," *ArXiv*, vol. abs/2402.09466, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267681906

[15] W. Zhong, H. Xiong, Y. Hua, D. H. Shah, Z. Liao, and Y. Xu, "TSFANet: Temporal-spatial feature aggregation network for GNSS jamming recognition," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–13, 2024.

[16] C. J. Swinney and J. C. Woods, "GNSS jamming classification via CNN, transfer learning & the novel concatenation of signal representations," in *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, 2021, pp. 1–9.

[17] B. Yang, C. Dong, B. Gao, Y. Liu, W. Cui, and F. Gao, "Research on GNSS interference recognition based on ROI of correlation peaks," *International Journal of Satellite Communications and Networking*, vol. 40, no. 5, pp. 330–342, 2022.

[18] H. Nasser, G. Berz, M. Gómez, A. Fuente, J. Fidalgo, W. Li, M. Pattinson, P. Truffer, and M. Troller, "GNSS interference detection and geolocalization for aviation applications," in *Proceedings of the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, 10 2022, pp. 192–216.

[19] N. Raichur, T. Brieger, D. Jdidi, T. Feigl, J. van der Merwe, B. Ghimire, F. Ott, A. Rügamer, and W. Felber, "Machine learning-assisted GNSS interference monitoring through crowdsourcing," in *Proceedings of the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, 09 2022.

[20] V. Ivanov, M. Scaramuzza, and R. C. Wilson, "Deep temporal semi-supervised one-class classification for GNSS radio frequency interference detection," *Journal of Navigation*, p. 1–23, 2024.

[21] J. R. van der Merwe, D. C. Franco, D. Jdidi, T. Feigl, A. Rügamer, and W. Felber, "Low-cost COTS GNSS interference detection and classification platform: Initial results," in *2022 International Conference on Localization and GNSS (ICL-GNSS)*, 2022, pp. 1–8.

[22] D. Fu, X. Li, W. Mou, M. Ma, and G. Ou, "Navigation jamming signal recognition based on long short-term memory neural networks," *Journal of Systems Engineering and Electronics*, vol. 33, no. 4, pp. 835–844, 2022.

[23] Y. Xie, J. Wang, H. Li, A. Dong, Y. Kang, J. Zhu, Y. Wang, and Y. Yang, "Deep learning CNN-GRU method for GNSS deformation monitoring prediction," *Applied Sciences*, vol. 14, no. 10, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/10/4004

[24] C. Mateo and J. Talavera, "Short-time Fourier transform with the window size fixed in the frequency domain," *Digital Signal Processing*, vol. 77, pp. 13–21, 2018.

[25] I. E. Mehr and F. Dovis, "A deep neural network approach for classification of GNSS interference and jamming," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 61, no. 2, pp. 1660–1676, 2025.

[26] I. E. Mehr, O. Savolainen, L. Ruotsalainen, and F. Dovis, "Dual-stage deep learning approach for efficient interference detection and classification in GNSS," in *Proceedings of the 37th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2024)*, Baltimore, Maryland, September 2024, pp. 3336–3347. [Online]. Available: https://doi.org/10.33012/2024.19686

[27] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[29] D. Britz, "Recurrent neural network tutorial, part 4—implementing a GRU and LSTM RNN with Python and Theano," https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-4/, 2015, accessed: 2025-01-27.

[30] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light gated recurrent units for speech recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 92–102, 2018.

[31] Y. Su and C.-C. J. Kuo, "On extended long short-term memory and dependent bidirectional recurrent neural network," *Neurocomputing*, vol. 356, pp. 151–161, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231219306289

[32] I. E. Mehr, G. Caputo, D. Salza, M. Fantino, and F. Dovis, "Dataset of jammed signals with 10 and 20 MHz sampling frequency," 2025. [Online]. Available: https://zenodo.org/communities/atlantis-project/records?q=&f=resource_type%3Adataset&l=list&p=1&s=10&sort=newest

[33] J. Ansel and et al., "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024.

[34] M. Abadi and et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.