# GNSS jamming detection using bidirectional long short-term memory

**A Reda and T Mekkawy**
Avionics department, Military Technical College, Cairo, Egypt
E-mail: ali.aboelyazeed@ieee.org

**Abstract.** Unmanned Aerial Vehicles (UAVs) require Global Navigation Satellite Systems (GNSS) for navigation and control. The UAVs are unable to follow the way-points of the intended path in jamming conditions without GNSS signals. However, the GNSS receiver is susceptible to many types of jammers, which degrades the quality of service. Jamming detection is crucial for increasing the location's accuracy as a result. The original signal's frequency is shared by the interfering signal. Therefore, traditional techniques have difficulty extracting the features. This paper proposes a Deep Learning (DL) model for signal jamming detection which uses Bidirectional Long-Short Term Memory (Bi-LSTM) to process data sequentially in succeeding leads and classify it into normal and interfering signals. The confusion matrix summarizes the outcomes. The simulation findings demonstrate that the prediction capability and interference detection precision are better than unidirectional Long-Short Term Memory (LSTM).

## 1. Introduction

Global Navigation Satellite Systems (GNSS) can offer precise location and time data, which are necessary for a variety of applications, including autonomous Unmanned Aerial Vehicle (UAV) operations. Although there are other locating systems, GPS and Galileo is the one that is most frequently utilized in most applications, particularly in UAVs [1]. However, the satellites of GNSS are allocated very far from the earth, therefore the received signal strength is often weak and below the noise threshold. Therefore, it is susceptible to human and natural interferences. The GNSS receiver may completely fail to lock the satellite signals because both man-made interferences (jamming and spoofing signals) and natural interferences (pulse interference, multi-path, and Radio Frequency Interference (RFI)) [2].

Many conventional techniques have been suggested to identify and reduce GNSS interference. The majority of GNSS systems employ Direct-Sequence Spread Spectrum (DSSS), which disperses GNSS signals over a larger bandwidth [3]. However, as a result of the cellular telecommunications industry's fast growth, the spectrum is becoming more and more crowded and is almost saturated. Signal harmonics from other systems operating in the same frequency band substantially impair the ability to determine the user's position, velocity, and time with accuracy [4]. For the purpose of detecting GNSS interference, a joint hybrid algorithm based on the Choi-Williams Transform (CWT) and Smoothed Pseudo Wigner-Ville Distribution (SPWVD) was proposed in [5]. This algorithm effectively removes the cross-terms present in the bi-linear distribution and improves localization and aggregation. Although traditional jamming detection methods have advantages, they are only useful in specific jamming environments and

under particular channel media conditions. In addition, because of the explosive growth of the wireless telecommunications industry, the next-generation GNSS jammers' accuracy makes it challenging for conventional algorithms to detect jamming signals accurately. Therefore, more effective jamming signal identification techniques are needed.

Deep Learning (DL) models have been extensively utilized in a variety of artificial intelligence domains, such as image recognition [6], image classification [7], and time-series data forecasting [8]. Also, Convolutional Neural Network (CNN) combined with Long-Short Term Memory (LSTM) was used for classifying transient RFI [9], which achieved efficient results for identifying the sources of transient RFI signals. In our research, we focus on the Bi-LSTM model that is applied for GNSS jamming detection, and compare the performance of the Bi-LSTM model with LSTM. The results show that the accuracy of Bi-LSTM is better than LSTM. To evaluate the performance of our model, we generate the confusion matrix to calculate accuracy (AC) and F1-score, and also calculate the root mean square error between the actual and predicted values. The main contributions of this paper are: we implement the GNSS jamming detection model using two different DL models; LSTM, and Bi-LSTM, then train and evaluate the performance of these models using two different datasets; GPS, and Galileo containing Continuous Wave (CW)and Chirp jamming signals. We compare the accuracy of the LSTM, and Bi-LSTM models using the confusion matrix. Finally, we prove that our model can detect efficiently the jamming signals for the received GNSS signals in dynamic and static scenarios.

## 2. System model

### *2.1. GNSS Signal Representation*

The model of the signal that a GNSS receiver receives at its input in an environment with interference can be described as [10]:

$$y_{RF}(t) = \sum_{i=1}^{N_s} r_{RF,i}(t) + \eta_{RF}(t) \tag{1}$$

where $r_{RF,i}(t)$ denotes the signal received from the $i^{th}$ GNSS satellite, $Ns$ denotes the number of satellites visible in the sky, and $\eta_{RF}(t)$ is the interfering signal received by the receiver. Generally speaking, $r_{RF,i}(t)$ can be written as [5]:

$$r_{RF,i}(t) = A_i \; c_i(t-\tau_i)\, d_i(t-\tau_i) \cos\left[2\pi(f_{RF} + f_{d,i}) + \varphi_{RF,i}\right] \tag{2}$$

where $A_i$ is the amplitude of the signal received from the $i^{th}$ GNSS satellite, $\tau_i$ is the code phase delay caused by transmitted channel delay, the navigation message is $d_i(t)$, the pseudo random noise (PRN) periodic code sequence is $c_i(t-\tau_i)$, the GNSS signal's carrier frequency is $f_{RF}$, the Doppler frequency is $f_{d,i}$, and the initial carrier phase offset is $\varphi_{RF,i}$. However, $\eta_{RF}(t)$ can be created as:

$$\eta_{RF}(t) = J_{RF}(t) + w_{RF}(t) \tag{3}$$

where the natural interference signal, or $w_{RF}(t)$, is produced by the thermal noise of the GNSS receiver. Typically, it is zero-mean stationary additive white noise (AWGN), and $J_{RF}(t)$ is the malicious interference signal, which can be produced by several types of jammers, including those that produce pulse, chirp, multi-CW, continuous wave (CW), and so on. We mainly discuss linear chirp jamming and CW jamming in this study.

In the first case, the malicious interference signal, $J_{RF}(t)$, in (3) is frequency modulated with nearly constant amplitude in a sweep jamming (linear chirp jamming) scenario. Sweep jamming reduces the precision of satellite navigation and the capacity to identify placement by performing a linear scan for the frequency band to lock the active carrier frequencies. It

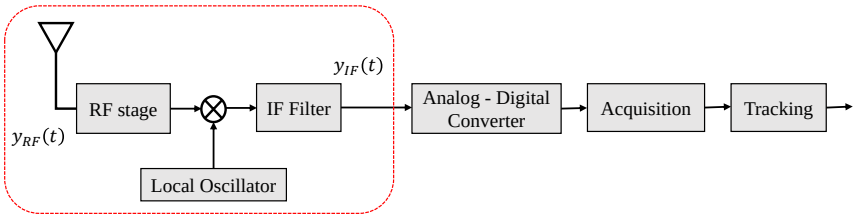



Figure 1: Block diagram of GNSS receiver.

then sends a high-power signal over this locked frequency. The expression for the linear chirp jamming signal is:

$$j_{RF}(t) = A_{inst}(t) \cos\left[\int_0^t f_{inst}(t)\, dt + \varphi_0\right] \tag{4}$$

where $A_{inst}(t)$ is the jamming signal's amplitude, and $\varphi_0$ is the signal's initial carrier phase, which may be a random variable with a uniform distribution across the range $[-\pi, +\pi)$. The instantaneous frequency jamming signal is the linear chirp instantaneous frequency, where $f_{inst}(t) = f_0 + kt$, where $k$ is the chirp rate, $f_0$ represents the carrier frequency, and $t \in [0, t_j]$, $t_j$ denotes the frequency sweep period for the jamming signal.

In the second case, a Malicious interference signal, $J_{RF}(t)$ in (3) can be represented as a CW jamming signal which is a special case of the sweep jamming such that $f_{inst}$ is constant. In this scenario $J_{RF}(t)$ can be expressed as [10]:

$$j_{RF}(t) = A_{CW}(t) \cos\left[2\pi f_{CW}(t)\, t + \varphi_0\right] \tag{5}$$

where $f_{CW}(t)$ is the CW jamming signal's carrier frequency, $A_{CW}(t)$ represents its amplitude factor, and $\varphi_0$ is its initial carrier phase. The received signal, $y_{RF}(t)$, is demodulated in the GNSS receiver using a local oscillator to an Intermediate Frequency (IF) level, where it is then passed through an IF filter. The signal is then converted to digital form via an analog-to-digital converter before being transferred to the acquisition and tracking blocks, as depicted in figure 1. Therefore, The received signal at IF can be represented as [5]:

$$y_{IF}(t) = \sum_{i=1}^{N_s} r_i(t) + \eta(t) = \sum_{i=1}^{N_s} A_i \widetilde{c}_i(t - \tau_i)\, d_i(t - \tau_i) \cos\left[2\pi(f_{IF} + f_{d,i}) + \varphi_i\right] + \eta(t) \tag{6}$$

where $r_i(t)$ is the received signal from the GNSS, $\eta(t)$ is the interference signal from the GNSS, $f_{IF}$ is the receiver's IF, and $\widetilde{c}_i(t - \tau_i)$ is the spreading code sequence that has been filtered in the front-end of the receiver. For the purpose of simplicity, we neglect the filter's influence, or $\widetilde{c}_i(t) \approx c_i(t)$.

The interaction between positive and negative frequency parts of the spectrum causes some cross-terms. Therefore, an analytic form of the received GNSS signal is used:

$$y_a(t) = y_{IF}(t) + j\, \hat{y}_{IF}(t) \tag{7}$$

where the analytic signal $y_a(t)$ includes an imaginary part $\hat{y}_{IF}(t)$ which representing the Hilbert transform of $y_{IF}(t)$ and a real part $y_{IF}(t)$ denoting the original signal of GNSS.

### *2.2. Problem Definition*

The included feature vectors for a specific time are extracted for the prediction model by analyzing the received IF GNSS signal, $y_a(t)$. The extracted feature vectors are represented as $\mathbf{Y}$, where $\mathbf{Y} \in \mathbb{C}^{N*T}$ is a matrix of $N$ features across time steps $T$ as follows:

$$\mathbf{Y} = \begin{bmatrix} x_1^1 & \cdots & x_T^1 \\ \vdots & \ddots & \vdots \\ x_1^N & \cdots & x_T^N \end{bmatrix}_{N*T} = \begin{bmatrix} \boldsymbol{y}_1 & \ldots & \boldsymbol{y}_T \end{bmatrix} 1 * T \tag{8}$$

where $x$ denoted the feature variable and $\boldsymbol{y}_i$ is the $i^{\text{th}}$ time sample with $N$ feature variables. Two sets of input datasets are created: one for training and the other for testing. In order to define the training set, $\boldsymbol{Z} = (\ \boldsymbol{y}_1, \boldsymbol{y}_2, .., \boldsymbol{y}_K)$, where $K \in T$, and the testing-set represents the others input data as $\boldsymbol{R} = (\ \boldsymbol{y}_{K+1}, \boldsymbol{y}_{K+2}, .., \boldsymbol{y}_T)$. In this study, our challenge is to accurately identify the interfering term by predicting the value of the next time-step signal $\boldsymbol{y}_{T+1}$ with the lowest possible error.

## 3. Jamming Detection based Deep Learning models

The Recurrent Neural Network (RNN) is a memory model that helps in predicting time series data. The model uses feedback from the preceding output of a particular layer to be used as input of the current layer, these inputs pass through hidden layers that contain activation functions, and after that, the inputs are forwarded to the output layers. However, in our case, we have long data sequences, K time-steps, which results in the learning using the RNN model being unachievable, because of the gradient vanishing problem. Therefore, for our jamming detection problem, we use the LSTM model to solve the gradient vanishing problem [11]. Then, Bi-LSTM is trained in both forward and backward directions of the input data, which improves the accuracy of the model. Also, the model's performance is impacted by the proper preparation of the incoming data. To enhance the model's output and generalizability, we therefore pre-process the data using zero-order hold approaches.

### *3.1. LSTM model*

LSTM is a particular RNN model that can be used for long data sequences [12]. For a long period of time series data, LSTM performs better at prediction than RNN because it can remember long-term dependencies. Additionally, it employs advanced gating techniques that include input, forget, and output gates, three nonlinear gates, as shown in figure 2, to solve the vanishing gradient problem. Following is a description of the relationship between the current and subsequent states (cell state and concealed state): LSTM is a special model of RNN which is applicable for long data sequences.

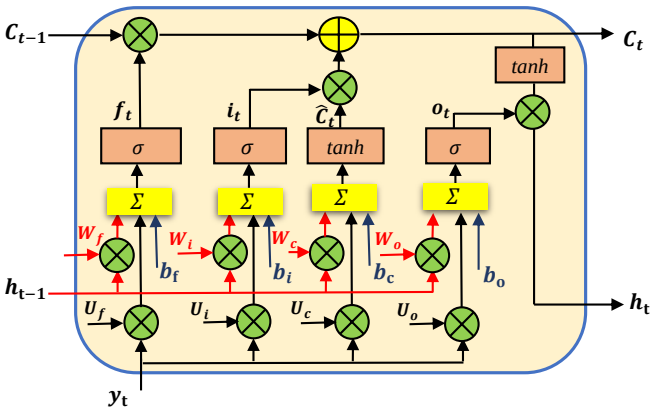



Figure 2: LSTM Structure.

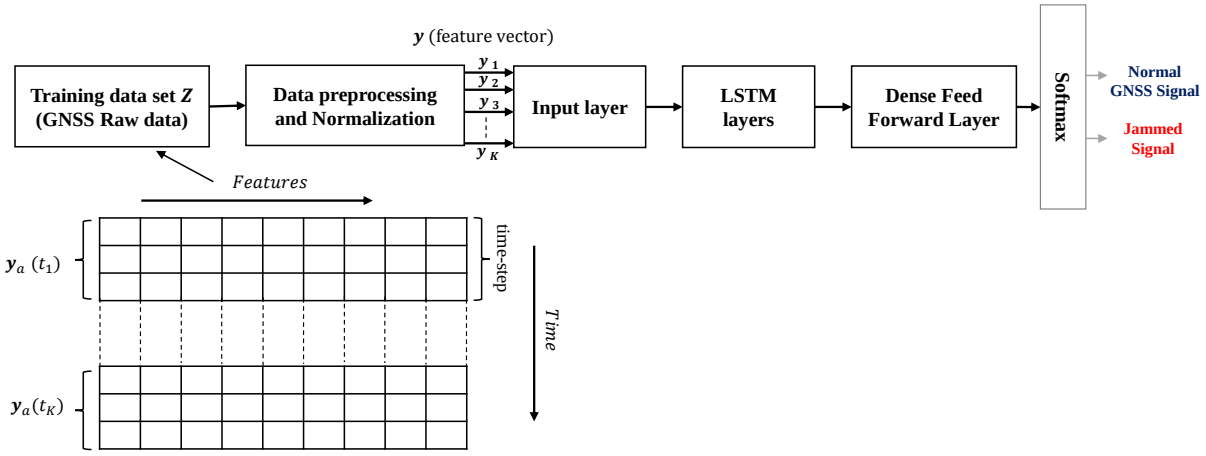



Figure 3: Jamming Detection based LSTM model.

$$f_t = \sigma(\boldsymbol{y}_t \boldsymbol{U}_f + h_{t-1} \boldsymbol{W}_f + b_f) \tag{9a}$$
$$i_t = \sigma(\boldsymbol{y}_t \boldsymbol{U}_i + h_{t-1} \boldsymbol{W}_i + b_i) \tag{9b}$$
$$o_t = \sigma(\boldsymbol{y}_t \boldsymbol{U}_o + h_{t-1} \boldsymbol{W}_o + b_o) \tag{9c}$$
$$\hat{C}_t = tanh(\boldsymbol{y}_t \boldsymbol{U}_c + h_{t-1} \boldsymbol{W}_c + b_c) \tag{9d}$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t \ * \hat{C}_t\right) \tag{9e}$$
$$h_t = tanh\,(C_t) * o_t \tag{9f}$$

The current and previous time steps are represented by $t$ and $t-1$, The non-linear activation function is $\sigma$. For ease of use, we use the subscripts $f$, $i$, $o$, and $c$ to designate the forget gate, input gate, output gate, and candidate memory, respectively.$\boldsymbol{y}_t$ specifies the input feature vector. The weight matrices for the connections between inputs and hidden layers are $\boldsymbol{U}$ and $\boldsymbol{W}$, respectively. A bias parameter is $b$. The forget function, $f_t$, filters out information that is not useful. The input gate is denoted by $i_t$, the output gate is denoted by $o_t$, the hidden state is denoted by $h_t$, the cell state is denoted by $C_t$, and the applicant hidden state is denoted by $hatC_t$ that is computed based on the current input and the previous hidden state.

Here, the LSTM architectural model that we utilized in our study classifies the input GNSS signal by determining if it is a jammed signal or a normal GNSS signal. The input GNSS data first passes to an input layer with n dimensions, followed by a fully connected layer and the soft-max function. The input raw data in figure 3 paths through pre-processing, where the data is transformed into a more comprehensible form to facilitate the feature extraction process, after which normalized data was processed. Time series feature vectors (one vector per time-step) are inputted into the LSTM model, which transforms them into probability vectors at the output layer for detection. Consequently, we have $K$ (time-steps) feature vectors for our dataset, which comprises GNSS and jamming signal as illustrated in Section 2.2, for the training dataset $\boldsymbol{Z}$. This model is trained using two different GNSS systems; GPS and Galileo. Each training process' input feature vector, $\boldsymbol{y}$, contains 18 variables (6 features x 3 signals). the $n$ hidden units of the LSTM layer. For the purpose of jamming detection, the output of the LSTM layer is transmitted to the fully-connected layer. Finally, soft-max activation layer is used to classify the signal.

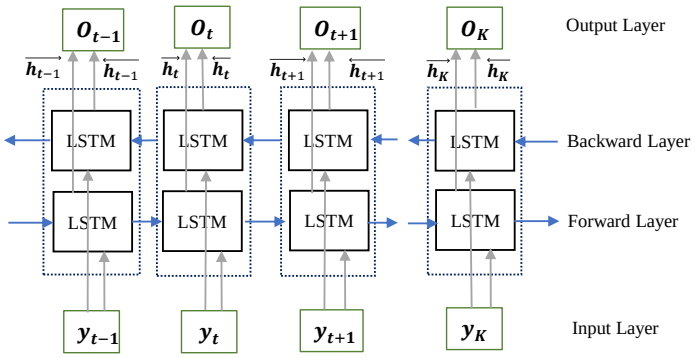



Figure 4: Bi-LSTM Structure.

*3.2. Bi-LSTM model*

As shown in figure 4, the Bi-LSTM model has two sequence information LSTM hidden layers: a forward layer (future to past) and a backward layer (past to future). Bi-LSTM, in contrast to ordinary LSTM, allows inputs to stream in both directions, allowing information to be exploited from both sides [13]. The backward layer, $\overleftarrow{h_t}$, and the forward hidden layer, $\overrightarrow{h_t}$,are defined as follows,respectively:

$$\overrightarrow{h_t} = tanh\left(\boldsymbol{W}_{y\overrightarrow{h}}\, y_t + \boldsymbol{W}_{\overrightarrow{h}\overrightarrow{h}} + b_{\overrightarrow{h}}\right) \tag{10}$$

$$\overleftarrow{h_t} = tanh\left(\boldsymbol{W}_{y\overleftarrow{h}}\, y_t + \boldsymbol{W}_{\overleftarrow{h}\overleftarrow{h}} + b_{\overleftarrow{h}}\right) \tag{11}$$

The weight matrices between the input and hidden layers and between two different hidden layers are represented by $W_{y\overrightarrow{h}}$ and $W_{\overrightarrow{h}\overrightarrow{h}}$, respectively. The bias vectors for the forward and backward hidden layers are represented by $b_{\overrightarrow{h}}$ and $b_{\overleftarrow{h}}$, respectively. By utilising (10) and (11), the outputs sequences of Bi-LSTM model, $\boldsymbol{O}_t$ can be described as following:

$$\boldsymbol{O}_t = \boldsymbol{W}_{\overrightarrow{h}O}\overrightarrow{h_t} + \boldsymbol{W}_{\overleftarrow{h}O}\overleftarrow{h_t} + b_O \tag{12}$$

where $\boldsymbol{W}_{\overrightarrow{h}O}$, $\boldsymbol{W}_{\overleftarrow{h}O}$, and $b_O$ are the weight matrices connected the output layer with forward and backward hidden layers, and the output bias vector, respectively.

## 4. Simulation Results and Discussions

*4.1. Dataset*

The data obtained in a sequence with time steps are collected to form the GNSS signal datasets, which is a sequential time series data. The dataset is provided in RINEX/OBS files as a result of the Ultrahack Galileo innovation challenge [14]. Using the Georinex Python module, we examined this dataset. Three distinct types of receivers: a professional receiver (NovAtel Propak6), a low-cost receiver (U-Blox M8T), and an Android device receiver (Sony Xperia XZ) are used to collect the data, which includes readings for GPS and Galileo satellites, under various conditions (static and kinematic Rx). In addition to natural jamming from the environment, jamming signals (Chirp - CW) are used during the collection of these data to separate the three categories of jamming data used by GPS and Galileo: static, kinematic, and natural. We extracted the features from our Rinex datasets, the feature extraction are: Frequency Band ($f_{RF}$), Channel or Code, Pseudo Range, Carrier Phase ($\varphi_{RF,i}$), Doppler Frequency ($f_{d,i}$), and Signal Strength ($r_{RF,i}(t)$) for GNSS signal and $J_{RF}(t)$ for jammer signal, each feature calculated from the equations in system model as shown in section 2.2. Our normal GNSS data includes

three GPS frequency bands (L1: 1575.42, L2: 1227.60, and L5: 1176.45 MHz), as well as three Galileo frequency bands (E1:1575.42, E5a:1176.45, and E5:1191.795MHz). As a result, we were able to collect multi-variant data for the GPS and Galileo training datasets with 8,668 and 7,719 samples, respectively, and 18 variables for each input feature vector $\boldsymbol{y}$.

We split the dataset into a 25% validation set and a 75% training set. To assess the model's performance, we also employed four unseen test data files that may or may not have jamming (some are affected by natural jamming). TensorFlow and scikit-learn based Python's Keras library is used to encode our models, while the Georinex Python library is used to pre-process the GNSS dataset, which running on the 64-bit version of the Windows 11 operating system. It is implemented on an ASUS laptop with a GeForce RTX 3060 6GB GDDR6 graphics card, 16GB of RAM, and an Intel Core i7-11800H processor.

*4.2. Performance analysis*

There are different ways to evaluate the performance of the model. We used the confusion matrix that depends on the following four measures:

- True-Positive (TP): jamming signal that is correctly identified by model as interference.
- True-Negative (TN): normal GNSS signal that is correctly identified as normal.
- False-Positive (FP): normal GNSS signal that identified by the model as interference.
- False-Negative (FN): jamming signal that is identified by the model as normal signal.

We used the above outcomes to evaluate our prediction model performance by computing the following parameters: Accuracy $= \frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}$ which defined as the percentage of the correct predictions overall. Precision $= \frac{\mathrm{TP}}{\mathrm{TP+\ FP}}$ is defined to indicate how the model is efficient for predicting a specific class. Moreover, the Recall $= \frac{\mathrm{TP}}{\mathrm{TP+\ FN}}$ is calculated to represent how often the model is able to detect a specific class.

There is an alternative evaluation matrix called F1-Score, which takes into account not only the number of prediction errors, but also look at the type of error. It is a harmonic mean of the Recall and Precision and can be expressed as follows:

$$\mathrm{F1-Score} = 2 * \frac{\mathrm{Recall\ .\ \ Precision}}{\mathrm{Recall} + \ \mathrm{Precision}} \tag{13}$$

Table 1: Evaluation Parameters for LSTM & Bi-LSTM using GPS & Galileo datasets.

| Dataset | **Model** | Accuracy (%) | Precision (%) | Recall (%) | F1-Score (%) | Test-accuracy (%) | Training time (Sec) |
|---|---|---|---|---|---|---|---|
| GPS | LSTM | 95.94 | 95.23 | 95.6 | 95.41 | 85.8 | **786** |
| | Bi-LSTM | 97.6 | 97.13 | 97.39 | 97.26 | 87.3 | 1252 |
| Galileo | LSTM | 95.6 | 94.95 | 95.13 | 95.04 | 85.72 | **778** |
| | Bi-LSTM | 97.14 | 96.59 | 96.95 | 96.77 | 87 | 1244 |

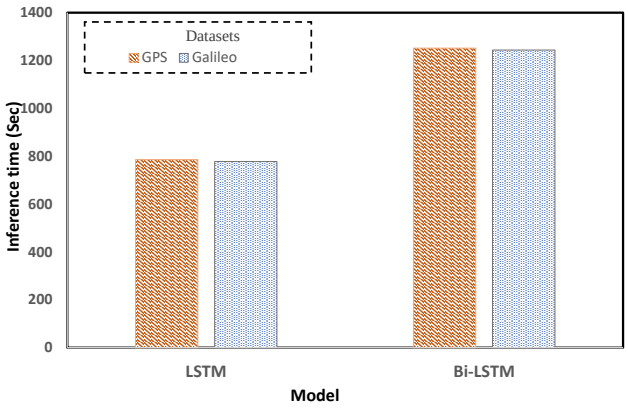


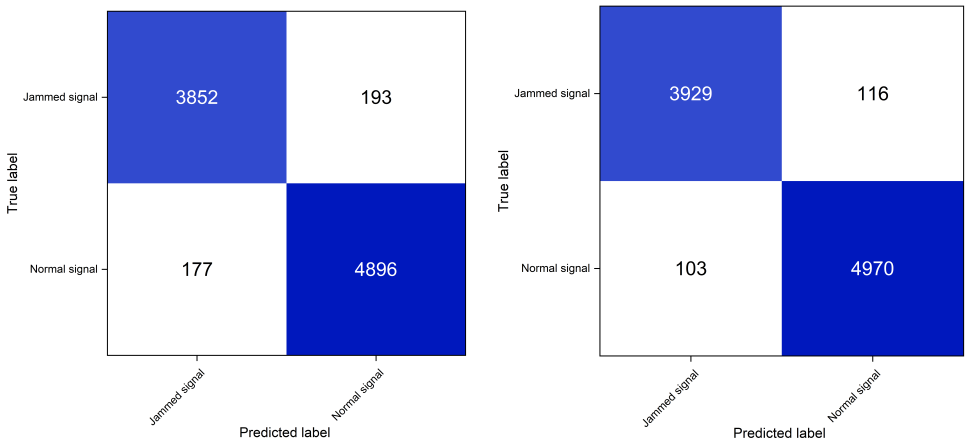

Figure 5: Training time for GPS & Galileo datasets.

*4.3. Experiment procedures and results*

In our jamming detection problem, we designed the Bi-LSTM model that contains two LSTM hidden layers as shown in figure 4 as a prediction model. In the training phase, we use tangent (tanh) as an activation function, mean square error (MSE) as an objective function, and the Learning rate (Lr) set as 0.001. We evaluate our model using different batch size hyperparameters (32, 64, 128, 256, and 256) and optimizer methods: stochastic gradient descent (SGD), Adam, Adagrad, and Adamax. We notice that the model underfits with 32 and 64 batch sizes, whereas the model overfits with 256 and 512 batch sizes, which lowers validation accuracy. Thus, as 128 batch size provides the maximum accuracy, we employ it. We can see that the Adam optimizer has the highest validation accuracy by comparing the optimizer methods. Therefore, we used the Adam optimizer with a 128 batch size to compute the confusion matrices.

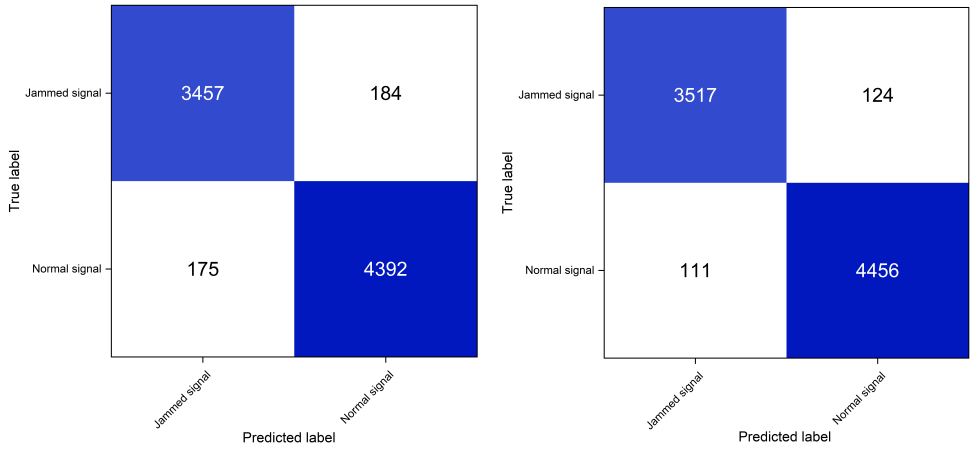
The final test process contains two different datasets: GPS and Galileo. We evaluate the performance of the implemented DL models (LSTM, and Bi-LSTM) using the confusion matrix, some parameters are obtained by this matrix, such as accuracy, recall, precision, and F1-score.

The first dataset used in the test phase is the GPS dataset, the results and the comparison can be shown in figure 6 and Table 1. Based on figure 5, the GPS dataset training process




(a) LSTM (b) Bi-LSTM

Figure 6: Confusion matrices for LSTM & Bi-LSTM using GPS dataset.




(a) LSTM (b) Bi-LSTM

Figure 7: Confusion matrices for LSTM & Bi-LSTM using Galileo dataset.

takes(786 s, and 1252 s) with 1400 epochs and batch size 128, therefore the LSTM is faster than the Bi-LSTM model. In figure 6 (a), and (b), a summary of the prediction findings for the two DL models is presented in the form of a confusion matrix, where each model more accurately predicts the jammed signals. As shown in Table 1, the Bi-LSTM model outperforms LSTM models with a performance of 97.6% validation accuracy, 97.13% precision, 97.39% recall, and 97.26% F1-score. The test performance of the Galileo dataset, the confusion matrix of Bi-LSTM in figure 7(b) has a lower FP and FN compared to the LSTM confusion matrix in figure 7(a). The Bi-LSTM model also outperforms LSTM models with a performance of 97.14% validation accuracy, 96.59% precision, 96.95% recall, and 96.77% F1-score.

The test-accuracy is the major indicator of the model's performance because we test the models with data that they have never experimented it before. Based on the outcomes of our study, the test-accuracy of Bi-LSTM for GPS and Galileo datasets is (87.3% and 87% respectively) outperformed the LSTM for GPS and Galileo datasets with (85.8% and 85.72% respectively) as shown in Table 1.

## 5. Conclusion

GNSS is mainly used for various applications of UAV, such as autonomous operations, thus the GNSS jamming attacks detection and mitigation is a very important challenge. In this paper, we implemented a system that detects GNSS jamming based on two different DL models: LSTM, and Bi-LSTM. We used two different datasets; GPS and Galileo which contain (Chirp - CW) jamming signals for training our models. These models' performances were evaluated in our research using a confusion matrix. Based on our results, the performance of the Bi-LSTM model is better than LSTM, but the computational complexity and training time is higher than the LSTM model.

## Reference

[1] Ferreira R, Gaspar J, Sebastião P and Souto N 2020 *Wireless Personal Communications* **115** 2705–2727
[2] Wu R, Wang W, Lu D, Wang L and Jia Q 2018 *Adaptive interference mitigation in GNSS* (Springer)
[3] Gao G X, Sgammini M, Lu M and Kubo N Jun 2016 *Proc. IEEE* **104** 1327–1338
[4] Wildemeersch M and Fortuny-Guasch J 2010 *EC Joint Research Centre, Security Tech. Assessment Unit, Tech. Rep*
[5] Sun K, Zhang M and Yang D 2016 *IEEE Transactions on Vehicular Technology* **65** 9057–9071

[6] Kong J, Wang H, Wang X, Jin X, Fang X and Lin S 2021 *Computers and Electronics in Agriculture* **185** 106134
[7] Zheng Y Y, Kong J L, Jin X B, Wang X Y, Su T L and Zuo M 2019 *Sensors* **19** 1058
[8] Jin X B, Gong W T, Kong J L, Bai Y T and Su T L 2022 *Entropy* **24** 335
[9] Czech D, Mishra A and Inggs M 2018 *Astronomy and computing* **25** 52–57
[10] Sun K and Zhang T 2021 *Sensors* **21** 1714
[11] Schmidhuber J 2015 *Neural Networks* **61** 85–117
[12] Yu Y, Si X, Hu C and Zhang J 2019 *Neural computation* **31** 1235–1270
[13] Huang Z, Xu W and Yu K 2015 *arXiv preprint arXiv:1508.01991*
[14] Ultahack, European GNSS Agency (GSA), and Ublox GALILEO INNOVATION CHALLENGE (2019, Nov.) URL `https://ultrahack.org/galileoinnovationchallenge`