

Danmarks
Tekniske
Universitet



GNSS Jamming Detection using Machine Learning and Deep Learning

Technical Project Report

SUPERVISOR

Søren Reime Larsen

AUTHOR

Macarena Burguera Perelló

January 2026

Contents

1	Introduction	2
1.1	Contributions	2
1.2	Report structure	2
2	Background	3
2.1	Baseband model and interference classes	3
2.2	Why baseband detection?	3
3	Data	3
3.1	Synthetic data generation (MATLAB)	4
3.2	Real controlled data: Jammertest 2023	5
3.3	Real labelled data for fine-tuning	5
3.4	In-the-wild test scanning: Rodby highway roadtest	7
4	Methods	7
4.1	End-to-end pipeline	7
4.2	Representations	7
4.2.1	Engineered features (XGB branch)	7
4.2.2	Spectrograms (DL branch)	9
4.3	Models	11
4.3.1	Decision trees and gradient boosting (XGBoost)	11
4.3.2	XGBoost classifiers	11
4.3.3	SE-CNN spectrogram classifier	12
5	Results	13
5.1	Domain gap and confusion patterns	14
5.2	Computational cost	15
5.3	Training dynamics (DL)	16
5.4	Notes on the Interference label	16
6	Discussion	17
6.1	Domain gap: synthetic to real	17
6.2	Wideband and class imbalance	17
6.3	Efficiency trade-offs	17
6.4	Limitations and next steps	17
7	Conclusions	18
	References	19

Abstract

This report presents a complete pipeline for *automatic GNSS jamming detection* using raw receiver baseband recordings. The approach operates directly on sampled IQ data (Septentrio SBF `BBSamples`, 4096 complex samples at $f_s = 60$ MHz) and classifies each block into an operational taxonomy: `NoJam`, `Chirp` (swept interference), `NB` (narrowband), and `WB` (wideband).

Two model families are implemented and compared: (i) feature-based gradient-boosted decision trees (XGBoost, including a 78-feature model and a compact 10-feature variant) and (ii) a deep-learning spectrogram classifier (SE-CNN). Training starts from **20,000** synthetic IQ blocks generated with a MATLAB generator and is then adapted using **18,320** labelled real blocks in the same 4-class taxonomy (including meaconing-like signatures mapped to `WB`).

All models are evaluated under a single fixed controlled validation scenario (Altus06, Day 1, 150 m, **12,689** blocks) to isolate model effects. On this benchmark, training on synthetic data alone leaves a clear performance gap (XGB-78: 91.33% accuracy, 62.07% macro-F1; SE-CNN: 74.47% accuracy, 59.94% macro-F1), while adaptation with labelled real data yields near-perfect performance (XGB-78: 99.73% accuracy, 99.25% macro-F1; SE-CNN: 99.80% accuracy, 99.75% macro-F1). Runtime profiling shows SE-CNN inference around **1.1–1.7 ms** per block, while the XGBoost path is dominated by feature extraction (**25.2 ms** mean) plus inference (**3.5 ms** mean) per block.

Code and data. The full implementation and supporting documentation are available in the project repository [1], with additional details in the `/docs` folder.

1 Introduction

Global Navigation Satellite Systems (GNSS) underpin modern positioning, navigation, and timing (PNT) services. Because GNSS signals are received well below the thermal noise floor, they are inherently vulnerable to radio-frequency interference (RFI), including intentional and unintentional jamming [8].

This project studies **automatic GNSS jamming detection from baseband data**. Instead of using receiver-specific tracking metrics or navigation-domain observables, detection is performed on raw sampled IQ blocks recorded by a Septentrio receiver (`BBSamples` blocks in SBF). This choice emphasizes receiver-independence and allows the same pipeline to be applied to both controlled test campaigns and long-duration “in-the-wild” road recordings [1].

1.1 Contributions

The project delivers:

- A complete offline processing pipeline from SBF recordings to per-block jamming labels (data extraction, feature/spectrogram generation, inference, and reporting) [1].
- A synthetic baseband data generation workflow with controlled parameter coverage (CNR/JSR bins and jamming types) implemented in MATLAB [2].
- Two detection branches: feature-based XGBoost and a spectrogram-based deep network (SE-CNN), including retraining / fine-tuning on **18,320** labelled real blocks from two Jammertest recordings (Section 3.3) [1, 6, 7].
- A controlled validation study on Jammertest 2023, reporting accuracy, macro-F1, confusion matrices, and computational cost for five model variants [1, 5].

1.2 Report structure

The remainder is organized as follows. Section 2 summarizes the interference taxonomy and the baseband formulation. Section 3 describes the data regimes and labelling. Section 4 details the processing pipeline and model families. Section 5 presents the controlled validation results with quantitative comparisons. Section 6 provides a critical analysis, limitations, and implications for operational scanning.

2 Background

2.1 Baseband model and interference classes

Let $x[n] \in \mathbb{C}$ denote a block of complex baseband samples at sampling rate f_s . Under nominal conditions, the received signal can be modeled as

$$x[n] = s_{\text{GNSS}}[n] + w[n], \quad (1)$$

where $s_{\text{GNSS}}[n]$ represents the aggregate GNSS signal content (below noise floor in open-loop baseband) and $w[n]$ is complex thermal noise. In the presence of interference, an additional term $j[n]$ appears:

$$x[n] = s_{\text{GNSS}}[n] + j[n] + w[n]. \quad (2)$$

The project uses an operational taxonomy aligned with common GNSS interference patterns [1, 5]:

- **NoJam:** noise-like background without significant interference.
- **NB (narrowband):** one or more tones or narrow spectral peaks within the receiver bandwidth.
- **Chirp:** swept interference with time-varying carrier frequency, often producing diagonal structures in a time-frequency representation.
- **WB (wideband):** noise-like broadband jamming spanning a substantial fraction of the band.

2.2 Why baseband detection?

Baseband detection has three practical advantages for this project: (i) it avoids dependence on proprietary receiver internals, (ii) it enables consistent processing of both synthetic and real recordings, and (iii) it supports long-duration scanning with uniform block-level decisions [1]. The main downside is the need to process large data volumes and the sensitivity to domain shift (synthetic vs. real), which is treated explicitly in this work.

3 Data

The workflow separates **training**, **validation** (controlled, labelled), and **test scanning** (unlabelled, in-the-wild) regimes [1]. This separation matters: validation quantifies general-

ization on labelled data under controlled conditions, while scanning evaluates operational behavior (rates, persistence, and event structure) without ground truth.

3.1 Synthetic data generation (MATLAB)

Synthetic baseband IQ is generated with a dedicated MATLAB generator [2]. The generator synthesizes a multi-channel GNSS-like baseband signal, injects a jammer waveform, applies front-end impairments, and exports complex IQ blocks together with per-block metadata. In total, 2000 samples per class were generated for the **training** subset, whereas 1500 samples per class were generated for both **test** and **validation** subsets, leading to a total of **20000** synthetic samples. Table 1 summarizes the configuration used in this project (sampling rate, block length, label set, and C/N₀/JSR ranges).

Table 1: Synthetic dataset configuration used for training data generation in MATLAB.

Parameter	Value
Sampling rate	$f_s = 60$ MHz
Block length	4096 complex samples per block
Primary classes	NoJam, NB, Chirp, WB
C/N ₀ range	30–70 dB-Hz
JSR range	30–80 dB (40–80 for WB)
Band presets	Randomly drawn per block from the generator’s supported bands (e.g., L1CA/L2C/L5/E1OS)
RF impairments	DC offset, IQ imbalance, carrier-frequency offset / phase noise, filter ripple, AGC/ADC quantization

At a high level, each generated block follows the steps described in the generator documentation [2]:

1. Draw a band preset (e.g., L1CA/L2C/L5/E1OS) and generate a sum of PRN-like channels;
2. Draw C/N₀ and JSR from configurable bins and scale noise and jammer power accordingly;
3. Generate a jammer waveform (Chirp, NB, WB, or NoJam) from a jammer-family factory;
4. Apply RF front-end impairments (e.g., DC offset, IQ imbalance, CFO/phase noise, filter ripple, AGC/ADC quantization);
5. Save the complex IQ vector together with metadata (band, label, C/N₀, JSR, and basic power statistics).

Spectrogram preprocessing and deep-learning training settings are described in the Methods section (Table 9).

3.2 Real controlled data: Jammertest 2023

Real labelled data is drawn from the Jammertest 2023 campaign, a controlled live-interference test event with predefined testcases [5]. Labels are produced with an external labelling tool that aligns interference periods to SBF `BBSamples` blocks [4]. This report uses a fixed validation subset (Altus06, Day 1, 150 m) so that all models are compared under identical conditions [1].

Table 2: Altus06 validation subset (150 m) label distribution (excluding `Interference`).

Class	Count	Share [%]
NoJam	8470	66.76
Chirp	2843	22.41
NB	1342	10.58
WB	32	0.25
Total	12687	100.00

3.3 Real labelled data for fine-tuning

To close the synthetic–real domain gap without changing the 4-class taxonomy, the models are **fine-tuned** on two additional labelled SBF recordings from the Jammertest 2023 campaign. Both recordings are converted to labelled NPZ blocks with the same extraction and labelling tooling as the validation subset [4, 1]. Across both files, the fine-tuning pool contains **18,348** labelled blocks at $f_s = 60$ MHz (GPS week 2280). Because the model heads are defined over the four main classes (`NoJam/Chirp/NB/WB`), the auxiliary label `Interference` is treated as “other” and excluded from the 4-class training/evaluation pool, leaving **18,320** blocks.

Table 3 summarizes the two recordings (UTC time range and number of labelled blocks). Tables 4–6 provide the corresponding label distributions.

Table 3: Overview of the real labelled datasets used for fine-tuning. Time ranges are in UTC. “4-class blocks” excludes samples labelled as `Interference`.

Dataset	SBF file	Time range [UTC]	Duration	Blocks	4-class blocks
Alt01004 (roadside)	alt01004.sbf	2023-09-20 14:05:37–15:13:21	1h 07m 44s	12193	12192
Alt06 (meaconing afternoon)	alt06 - Meaconing afternoon.sbf	2023-09-19 12:01:51–15:15:02	3h 13m 11s	6155	6128
Combined	—	—	—	18348	18320

Table 4: Fine-tuning set 1 (Alt01004, roadside test) label distribution.

Class	Count	Share [%]
NoJam	9433	77.36
Chirp	1359	11.15
NB	1400	11.48
WB	0	0.00
Interference	1	0.01
Total	12193	100.00

Table 5: Fine-tuning set 2 (Alt06, “Meaconing afternoon”) label distribution.

Class	Count	Share [%]
NoJam	4322	70.22
Chirp	0	0.00
NB	41	0.67
WB	1765	28.68
Interference	27	0.44
Total	6155	100.00

Table 6: Combined fine-tuning label distribution (Alt01004 + Alt06).

Class	Count	Share [%]
NoJam	13755	74.97
Chirp	1359	7.41
NB	1441	7.85
WB	1765	9.62
Interference	28	0.15
Total	18348	100.00

Why these two files. The two recordings are complementary:

- **Alt01004 (roadside test):** provides a large volume of real **Chirp** and **NB** events but contains no **WB** blocks (Table 4).
- **Alt06 (“Meaconing afternoon”):** while recorded during a meaconing exercise, the baseband snapshots exhibit strong broadband energy elevation and are therefore treated as **WB** interference for the purpose of this project. This file contributes **1,765 WB** blocks (28.68%), substantially increasing the coverage of real wideband examples (Table 5).

Overall, the combined fine-tuning pool contains **4,565** jammer blocks across **Chirp**/NB/WB (24.92% of the 4-class pool) and **13,755** NoJam blocks (75.08%). This balance is intentionally skewed toward clean data, reflecting operational scanning conditions, while still ensuring that the fine-tuned models see a meaningful number of real jammer examples—especially WB, which is rare in the fixed validation subset.

3.4 In-the-wild test scanning: Rodby highway roadtest

To assess operational behavior over long recordings, the trained classifiers are applied to an unlabelled roadtest dataset acquired on a public highway. Because no ground truth is available, this stage is interpreted via detection statistics (e.g., counts and temporal event structure), not via accuracy metrics [1].

4 Methods

4.1 End-to-end pipeline

The pipeline is implemented as an offline processing chain [1]:

1. **SBF parsing:** read Septentrio SBF files and extract each `BBSamples` block.
2. **Block serialization:** store blocks as compressed `.npz` samples, preserving metadata (timestamp, LO frequency, sampling rate).
3. **Pre-processing:** sanity checks and optional normalization.
4. **Representation:** either (a) compute engineered features or (b) compute a spectrogram tensor.
5. **Inference:** apply the chosen model family and log per-block predictions and probabilities.
6. **Reporting:** confusion matrices and metric summaries for labelled datasets; incident logs and plots for scanning runs.

4.2 Representations

4.2.1 Engineered features (XGB branch)

The feature-based branch extracts 78 scalar features per block to summarize both time- and frequency-domain structure [1]. The design goal is to capture signatures that separate the four main classes: (i) NoJam (noise-like), (ii) NB (spectrally concentrated), (iii) Chirp (time-varying instantaneous frequency), and (iv) WB (broadband elevation of the noise floor).

Concretely, the extractor combines:

- **Time-domain summaries:** amplitude and envelope statistics, including measures of modulation and impulsiveness.
- **Frequency-domain summaries:** FFT/PSD descriptors (e.g., spectral flatness, centroid/spread, and peak salience).
- **Time–frequency cues:** compact STFT-derived statistics that respond to chirp-like slopes and centroid drift.

Key extraction parameters are fixed and recorded (Table 7).

Table 7: Key feature-extraction parameters used by the engineered-feature pipeline.

Parameter	Value
Block length	4096 complex samples
Preprocess	remove mean; normalize by standard deviation
FFT length	4096
PSD (Welch)	$n_{\text{perseg}} = 128$
Peak search	minimum prominence 2 dB
STFT	$N_{\text{FFT}} = 256$, window length 256, hop 64

A reduced 10-feature variant (Table 8) is also evaluated to study the performance–efficiency trade-off.

Table 8: Reduced feature set (10 features) used by the XGB-10 model.

#	Feature name
1	nb_peak_salience
2	spec_peakiness_ratio
3	spec_flatness
4	spec_centroid_Hz
5	spec_spread_Hz
6	stft_centroid_std_Hz
7	stft_centroid_absderiv_med_Hzps
8	chirp_slope_Hzps
9	chirp_r2
10	env_mod_index

4.2.2 Spectrograms (DL branch)

The deep-learning branch operates on a short-time Fourier transform (STFT) spectrogram computed per block. Table 9 summarizes the STFT settings, data augmentation, and SE-CNN training hyperparameters.

Table 9: Spectrogram preprocessing and SE-CNN configuration used in this work (from `train/train_eval_cnn_spectrogram.py`).

Parameter	Value
Raw block length	4096 complex samples per <code>BBSamples</code> block (Septentrio)
Net input length	$N = 2048$ complex samples (crop/pad to <code>TARGET_LEN</code>)
STFT	Hann window, $N_{\text{FFT}} = 256$, window $L = 256$, hop $H = 64$
Frequency axis	Two-sided STFT (<code>return_onesided=False</code>) with <code>fftshift=True</code>
Spectrogram	Log-power $S = \log(X ^2 + \epsilon)$ with $\epsilon = 10^{-8}$ (single channel)
Normalization	Per-sample z-score
Time frames	$T = \lfloor \frac{N-L}{H} \rfloor + 1 = 29$ (freq \times time: 256×29)
Augmentation (IQ)	amplitude jitter (<code>AMP_JITTER=True</code>), random global phase (<code>PHASE_JITTER=True</code>)
Time shift (IQ)	circular shift up to ± 64 samples (<code>TIME_SHIFT=True</code>)
CFO jitter (IQ)	implemented but disabled by default (<code>CFO_JITTER=False</code>)
CNN channels	$[32, 64, 128, 192]$ with <code>MaxPool2d(2)</code> between stages
Activation	GELU (conv blocks and head)
SE block	reduction ratio $r = 16$ (channel attention)
Head	global average pooling \rightarrow FC 256 \rightarrow dropout 0.2 \rightarrow classifier
Imbalance handling	class-weighted cross-entropy and a <code>WeightedRandomSampler</code> (training)
Optimizer	AdamW ($\eta = 10^{-3}$, weight decay 10^{-3})
Batch size / epochs	256 / 80 (early stopping patience 60)
Scheduler	<code>ReduceLROnPlateau</code> (factor 0.5, patience 30)

Fine-tuning on real labelled data. To reduce the synthetic–real domain gap, the SE-CNN is fine-tuned on the labelled real datasets described in Section 3.3. In total, **18,320** blocks from the four main classes (`NoJam/Chirp/NB/WB`) are used, obtained by combining a roadside recording rich in `Chirp/NB` with a `WB`-heavy recording (“Meaconing afternoon”) that is treated as `WB` interference for this project. Fine-tuning keeps the original 4-class head and applies gentle updates (lower learning rate), optionally freezing the backbone for the

first epochs and using limited rehearsal to avoid catastrophic forgetting.

Table 10: Fine-tuning configuration for adapting the SE-CNN to real labelled data (from `retrain/retrain_dl.py`).

Parameter	Value
Target labels	Four-class head preserved: NoJam/Chirp/NB/WB
Preprocessing	Reuse checkpoint preprocessing by default (<code>USE_MODEL_CONFIG_FOR_PREPROC=True</code>)
Batch size / epochs	128 / 50
Optimizer	AdamW ($\eta = 2 \times 10^{-4}$, weight decay 10^{-3})
Early stopping	patience 25 (monitor validation loss)
Scheduler	ReduceLROnPlateau (factor 0.5, patience $\max(2, \lfloor 25/2 \rfloor) = 12$)
Freezing	freeze backbone for first 5 epochs (<code>FREEZE_BACKBONE=True</code> , <code>FREEZE_EPOCHS=5</code>); train head only
Loss / imbalance	class-weighted cross-entropy computed from real split counts
Augmentation	enabled (<code>AUGMENT=True</code>); time shift on (<code>TIME_SHIFT=True</code>); CFO jitter off (<code>CFO_JITTER=False</code>)
Rehearsal (optional)	can mix a capped subset of synthetic samples (e.g., WB-only) to reduce catastrophic forgetting

For a discrete-time complex baseband block $x[n]$ of length N , the STFT is

$$X[m, k] = \sum_{n=0}^{L-1} x[n + mH] w[n] e^{-j2\pi kn/N_{\text{FFT}}},$$

where L is the window length, H the hop size, N_{FFT} the FFT size, m the frame index, and k the frequency-bin index. In this project, each `BBSamples` block contains 4096 complex samples, but the DL model crops/pads to a fixed input length $N = 2048$ (Table 9). With $L = 256$, $H = 64$, and $N_{\text{FFT}} = 256$, the time axis has

$$T = \left\lfloor \frac{N - L}{H} \right\rfloor + 1 = 29$$

frames, yielding an $N_{\text{FFT}} \times T$ time–frequency grid. The classifier input is a normalized log-power spectrogram,

$$S[m, k] = \log(|X[m, k]|^2 + \varepsilon),$$

followed by per-sample normalization (z-score) to reduce sensitivity to absolute gain/AGC variations [1].

4.3 Models

4.3.1 Decision trees and gradient boosting (XGBoost)

A *decision tree* partitions the feature space using a sequence of binary threshold tests. Each internal node selects a feature x_j and a threshold t and routes the sample left/right depending on whether $x_j < t$. Leaves store class scores, so the tree implements a piecewise-constant approximation of the decision boundary.

XGBoost builds an *ensemble* of such trees by gradient boosting [6]. The model prediction is the sum of M trees,

$$\hat{y}(x) = \sum_{m=1}^M f_m(x), \quad f_m \in \mathcal{F},$$

and training minimizes a regularized objective of the form

$$\mathcal{L} = \sum_i \ell(y_i, \hat{y}(x_i)) + \sum_{m=1}^M \Omega(f_m),$$

where $\ell(\cdot)$ is the multiclass loss and $\Omega(\cdot)$ penalizes tree complexity (e.g., number of leaves and leaf weights). Boosting adds trees sequentially so that each new tree fits the current residual structure (using first- and second-order gradient information in XGBoost).

In practice, key hyperparameters control capacity and regularization: `max_depth` limits tree depth, `n_estimators` sets the number of boosting rounds, `learning_rate` shrinks each added tree, and `subsample/colsample_bytree` add stochasticity to reduce overfitting.

4.3.2 XGBoost classifiers

Two feature-based classifiers are evaluated. **XGB-78** uses the full 78-feature set. **XGB-10** uses the reduced set in Table 8. The base synthetic XGB-78 model uses the hyperparameters in Table 11; the “inc-tuned” variants are obtained by continuing training on the real fine-tuning pool described in Section 3.3, as implemented in the repository scripts [1].

Table 11: Best hyperparameters for the synthetic-trained XGB-78 model (as used in the repository implementation).

Hyperparameter	Value
max_depth	4
learning_rate	0.2
n_estimators	400
subsample	1.0
colsample_bytree	1.0

4.3.3 SE-CNN spectrogram classifier

The spectrogram model is a compact convolutional neural network augmented with Squeeze-and-Excitation (SE) blocks [7]. The architecture is parameterized by the channel list in Table 9 (here [32, 64, 128, 192]) and consists of:

- **Convolutional backbone:** four stages with channel widths [32, 64, 128, 192]. Each stage applies two 3×3 convolutions with BN+GELU, followed by MaxPool2d(2).
- **Channel recalibration (SE):** after selected stages, an SE module rescales channels based on global context.
- **Classifier head:** global average pooling, a fully connected layer of size 256, dropout (0.2), and a final softmax over the four classes.

What the SE block does. Given a convolutional feature map $U \in \mathbb{R}^{C \times H \times W}$, SE performs:

1. **Squeeze (global context):**

$$z_c = \frac{1}{HW} \sum_{i=1}^H \sum_{j=1}^W U_{c,i,j}, \quad c = 1, \dots, C.$$

2. **Excitation (gating):** a small MLP produces channel weights

$$s = \sigma(W_2 \delta(W_1 z)),$$

where δ is GELU and σ is the sigmoid.

3. **Scale:** channels are reweighted as $\tilde{U}_{c,i,j} = s_c U_{c,i,j}$.

In this application, the SE mechanism helps the network emphasize frequency bands and time–frequency patterns that are most discriminative for each jammer type (e.g., narrow peaks for NB or broadband elevation for WB). Two variants are considered: a synthetic-trained base model and a fine-tuned model trained further on real labelled Jammertest data [1].

5 Results

All five models are evaluated on the same controlled validation subset (Altus06, Day 1, 150 m, 12,689 blocks) [1]. To avoid distortion by rare labels, the two samples marked **Interference** are excluded from the main metrics.

Evaluation metrics. All results in Table 12 are computed on the 4-class taxonomy NoJam/Chirp/NB/WB; the two blocks labelled **Interference** are excluded because they are too few to yield stable statistics.

The confusion matrix (Figures in Section 5) counts how often each true class (rows) is predicted as each class (columns). The diagonal entries are correct classifications; off-diagonal entries show the dominant confusions.

Let y_i be the true class and \hat{y}_i the predicted class for block i , with $i = 1, \dots, N$. The overall accuracy is the fraction of correctly classified blocks:

$$\text{Acc} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}[y_i = \hat{y}_i].$$

Accuracy is easy to interpret, but it can hide poor performance on rare classes when the dataset is imbalanced.

To describe performance per class c , we use the standard one-vs-rest definitions: $\text{TP}_c = \#(y = c \wedge \hat{y} = c)$, $\text{FP}_c = \#(y \neq c \wedge \hat{y} = c)$, $\text{FN}_c = \#(y = c \wedge \hat{y} \neq c)$. From these, precision and recall are

$$\text{Prec}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}, \quad \text{Rec}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}.$$

Precision answers “when the model raises class c , how often is it correct?” Recall answers “of all true blocks of class c , how many did the model detect?” The per-class F1 score combines both:

$$\text{F1}_c = \frac{2 \text{Prec}_c \text{Rec}_c}{\text{Prec}_c + \text{Rec}_c}.$$

Finally, macro-F1 averages these scores across classes, giving each class equal weight:

$$\text{Macro-F1} = \frac{1}{4} \sum_{c \in \{\text{NoJam}, \text{Chirp}, \text{NB}, \text{WB}\}} \text{F1}_c.$$

This makes macro-F1 a better summary than accuracy when we care about *all* jamming types, not only the dominant label.

Because false alarms on clean data are operationally costly, we also report the false-alarm rate on **NoJam**:

$$\text{FAR}_{\text{NoJam}} = \frac{\#(y = \text{NoJam} \wedge \hat{y} \neq \text{NoJam})}{\#(y = \text{NoJam})}.$$

Additionally, Table 12 lists recall for NB and WB (detection rate for two practically important jammer classes), and the mean per-block runtime (XGBoost includes feature extraction and inference; SE-CNN reports STFT computation and network inference).

Table 12: Controlled validation results on Altus06 (150 m). Metrics are computed over the four primary classes (NoJam/Chirp/NB/WB), excluding the 2 blocks labelled **Interference**.

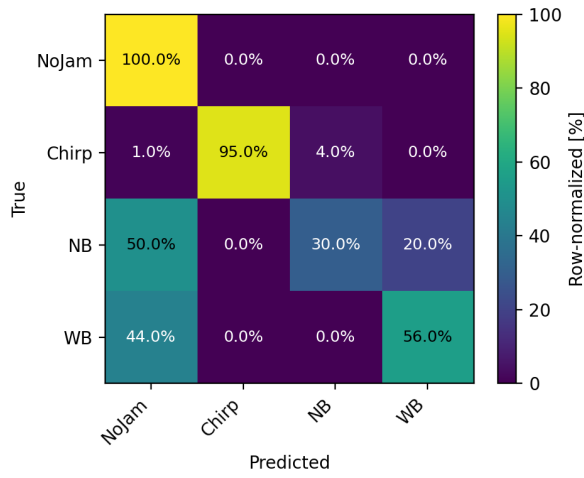
Model	Acc [%]	Macro-F1 [%]	FAR _{NoJam} [%]	Rec _{NB} [%]	Rec _{WB} [%]	Time [ms]
XGB-78 (trained on synthetic)	91.33	62.07	0.15	30.48	56.25	35.94
XGB-78 (inc-tuned on real)	99.73	99.25	0.17	98.66	96.88	35.94
XGB-10 (inc-tuned on real)	99.50	83.39	0.33	99.25	21.88	18.44
SE-CNN (trained on synthetic)	74.47	59.94	32.51	65.13	100.00	1.69
SE-CNN (fine-tuned on real)	99.80	99.75	0.26	99.85	100.00	1.12

5.1 Domain gap and confusion patterns

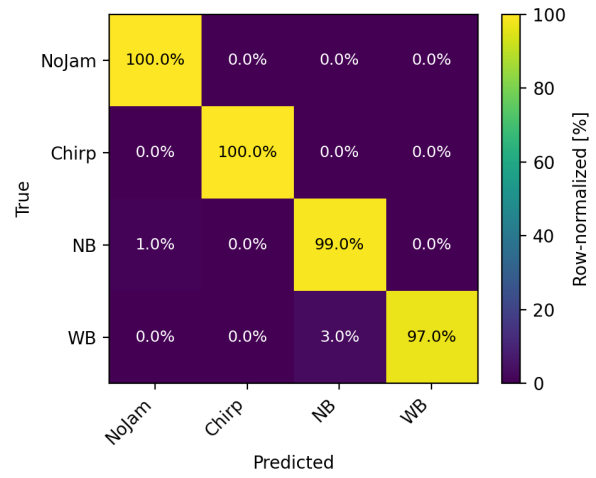
Figure 1 summarizes row-normalized (per-class recall) confusion matrices over the four main classes for the two model families. The synthetic-only models exhibit a clear domain gap:

- **XGB (synthetic-only)**: large recall loss on NB (30.48%).
- **SE-CNN (synthetic-only)**: high false-alarm rate on NoJam (32.51%).

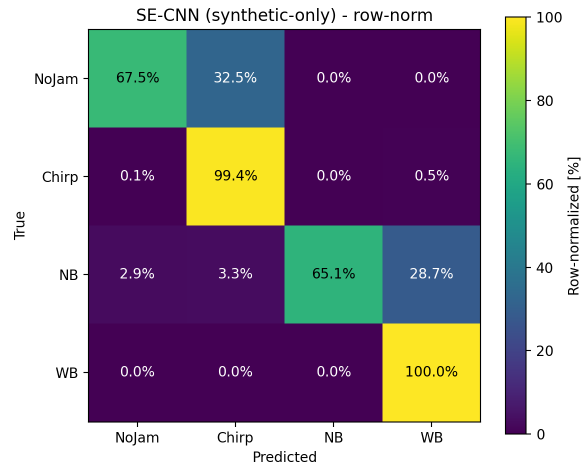
After fine-tuning on the real labelled pool described in Section 3.3, both families reach near-ceiling performance with uniformly high recall across classes.



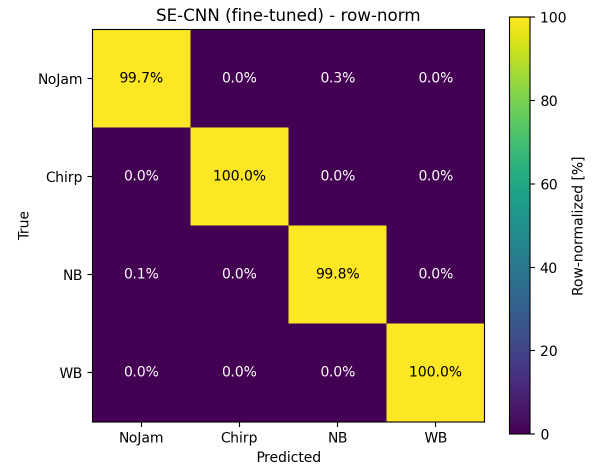
(a) XGB-78 (synthetic-only)



(b) XGB-78 (fine-tuned)



(c) SE-CNN (synthetic-only)



(d) SE-CNN (fine-tuned)

Figure 1: Row-normalized confusion matrices (NoJam/Chirp/NB/WB). Interference is excluded for consistency with Table 12.

5.2 Computational cost

Figure 2 compares the mean per-block processing time recorded by the evaluation scripts. The DL branch is substantially faster in the provided setup (mean below 2 ms/block), while the XGB branch is dominated by feature extraction (tens of ms/block) [1].

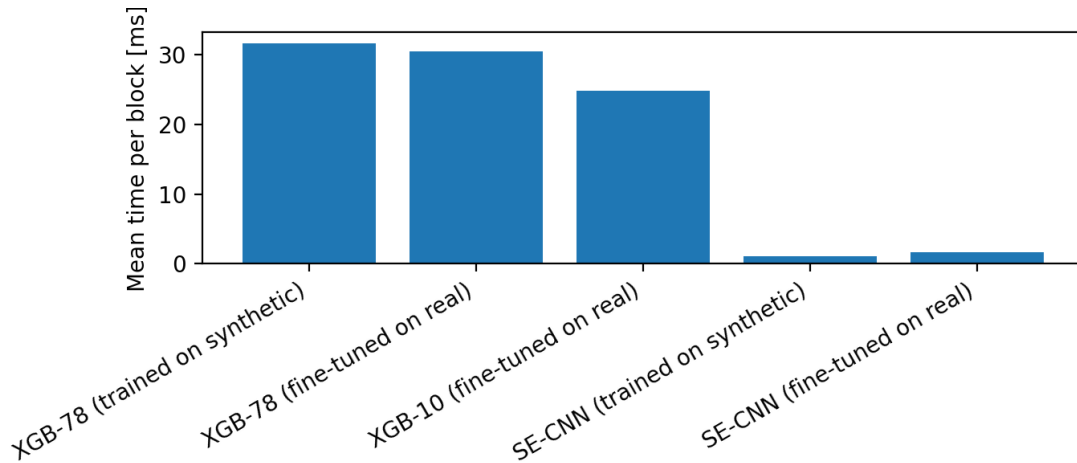


Figure 2: Mean processing time per block for each evaluated model (as logged by the evaluation scripts).

5.3 Training dynamics (DL)

Figure 3 shows training curves for the synthetic base model and the fine-tuning run. The fine-tuned model selects best epoch 41 based on validation loss (see the run `summary.txt` in the fine-tuning artifacts directory) [1].

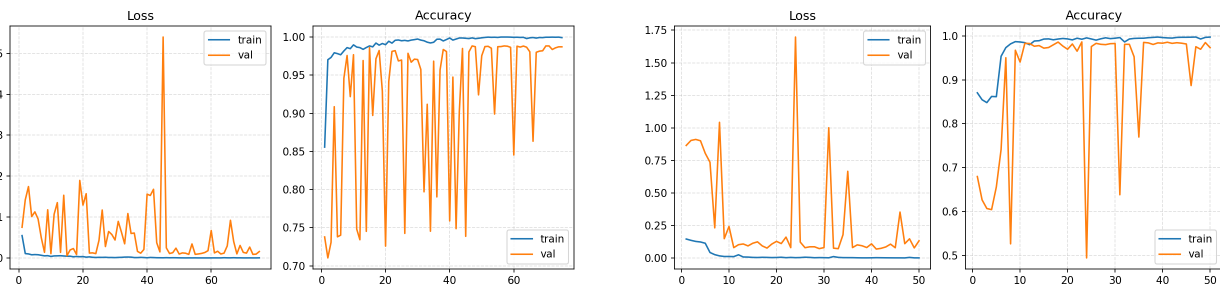


Figure 3: DL training curves: synthetic base run (left) and fine-tuning run (right).

5.4 Notes on the Interference label

Only 2 blocks carry the label **Interference** in the validation set [1]. Across models, these samples are inconsistently mapped (often to **NB** or **Chirp**), suggesting that a separate “other interference” class would require more labelled examples to be meaningful.

6 Discussion

6.1 Domain gap: synthetic to real

The most important empirical finding is the magnitude of the domain gap. Both model families degrade strongly when trained only on synthetic IQ: the synthetic XGB model reaches 91.33% accuracy but fails on NB recall (30.48%), while the synthetic DL model collapses mainly due to false alarms ($\text{FAR}_{\text{NoJam}}=32.51\%$). After fine-tuning on real labelled Jammertest data, both families reach $\approx 99.7\text{--}99.8\%$ accuracy with macro-F1 above 99.2% (Table 12).

A plausible interpretation is that the synthetic generator, while controlling high-level parameters (CNR, JSR, jammer type), cannot fully capture receiver-dependent effects present in real SBF recordings (front-end filtering, quantization, AGC behavior, and multi-emitter environments). Fine-tuning acts as a compact domain-adaptation mechanism that aligns the learned decision boundaries to real-world statistics.

6.2 Wideband and class imbalance

The validation set contains only 32 WB blocks (0.25%), which makes WB metrics sensitive to a few misclassifications. In contrast, the fine-tuning pool contains **1,765** real WB blocks (Table 5), which helps calibrate the decision boundary for wideband signatures even though WB remains rare in the fixed validation scenario.

This is visible in the XGB-10 model: overall accuracy remains high (99.50%), but WB recall drops sharply (21.88%), indicating that the reduced feature set loses some WB-discriminative cues. In contrast, the fine-tuned SE-CNN maintains perfect WB recall in this subset.

6.3 Efficiency trade-offs

In the logged evaluation setup, the DL branch achieves sub-2 ms/block mean processing time, while the XGB branch costs roughly 25–32 ms/block (Figure 2). For long-duration scanning, this difference is operationally significant. However, feature-based models retain two practical advantages: (i) interpretability (feature importance and failure attribution) and (ii) easier deployment on CPU-only targets if spectrogram inference is not accelerated.

6.4 Limitations and next steps

This report focuses on one fixed controlled validation scenario to enable clean model comparisons. Future work should:

- expand controlled validation across more receivers, distances, and days from Jammertest to quantify robustness;
- improve synthetic realism (e.g., better modeling of receiver front-end and band-limited noise) to reduce reliance on real fine-tuning;
- collect sufficient labelled examples for an “other interference” class to treat **Interference** explicitly;
- evaluate end-to-end scanning performance on the roadtest dataset using incident-level metrics (duration, clustering, temporal consistency) rather than block-wise counts.

7 Conclusions

This project demonstrates that reliable GNSS jamming detection can be achieved directly from baseband recordings using both feature-based ML and spectrogram-based DL. Across a controlled Jammertest 2023 validation scenario (Altus06, 150 m), the best models achieve $\approx 99.8\%$ accuracy and macro-F1 above 99.7%, but only after fine-tuning on real labelled data.

The central takeaway is that **domain adaptation is not optional** for baseband interference classification: synthetic-only training is insufficient to generalize robustly to real receiver recordings. With a modest amount of labelled real data, however, both XGB and SE-CNN can reach near-ceiling performance. For operational long-duration scanning, the fine-tuned DL model is preferred in this repository due to its substantially lower per-block compute cost.

Reproducibility

All scripts, model artifacts, evaluation CSVs, and the complete documentation are provided in the repository [1]. In particular, `docs/usage.md` and `docs/1_Pipeline.md` define the exact command entry points and expected data formats.

References

- [1] M. Burguera, *GNSS-jamming-classifier* (GitHub repository), 2025. Available: <https://github.com/macaburguera/GNSS-jamming-classifier>
- [2] M. Burguera, *GNSS_generator* (GitHub repository), 2025. Available: https://github.com/macaburguera/GNSS_generator
- [3] D. Pascual, *GNSS-matlab* (GitHub repository). Available: <https://github.com/danipascual/GNSS-matlab>
- [4] M. Burguera, *sbj-labeller* (GitHub repository), 2025. Available: <https://github.com/macaburguera/sbj-labeller>
- [5] Jammertest, *Testplan for the event “Jammertest 2023”*, Rev. A7, 2023. Available: <https://jammertest.no/content/files/2024/03/Jammertest-2023---Testplan.pdf>
- [6] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’16)*, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.
- [7] J. Hu, L. Shen, and G. Sun, “Squeeze-and-Excitation Networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7132–7141.
- [8] E. D. Kaplan and C. J. Hegarty (eds.), *Understanding GPS/GNSS: Principles and Applications*, 3rd ed. Artech House, 2017.