Universiteit Leiden

# Usage of neuromorphic hardware accelerators for low-latency and low-power RFI mitigation

Nick Boom (s4084217)

Supervisors:
Prof. dr. Rob van Nieuwpoort & Dr. Chris Broekema

Bachelor Thesis– Informatica

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 24, 2026

**Abstract**

In this thesis we investigate real-time RFI flagging in radio telescopes, addressing the limitations of traditional methods which only work offline. While Vision Transformers (ViTs) demonstrate high accuracy, they also bring unscalable computational costs. We evaluate performance and efficiency of two different deep learning model architectures on AI edge hardware (the Axelera Metis edge accelerator): a State Space Model (SSM) and a Convolutional Neural Network (CNN). Due to a compilation compatibility mismatch between the selective scan function of the SSM and the Axelera compiler, development shifted exclusively to the CNN. The CNN model achieves high accuracy, higher than a 400M parameter ViT, while reducing the computational demand by a factor of around five thousand. The Axelera Metis demonstrates a high power efficiency compared to a modern GPU, since it is six times more energy efficient per sample. The throughput on our test platform is not sufficient for real-time flagging, although extrapolating the measured on-chip execution time to a larger Metis configuration suggests the required rate might be achievable.

# Contents

# 1 Introduction

## 1.1 Context & Motivation

Traditional radio astronomy relies heavily on large parabolic dishes, as shown in Figure 1. However, observing at lower frequencies requires another approach due to physical constraints. When the frequency decreases the wavelength of the radio signal increases. For traditional dish-telescopes the radius of the dish determines the minimum frequency the dish can reliably capture. Also factoring in that the dish must be able to be pointed makes it impossible to use dish-telescopes to capture low frequency signals.

Consequently, alternative radio telescopes have been developed. One of those alternative radio telescopes is LOFAR, which stands for low-frequency array. It consists of an array of thousands of simple omnidirectional dipole antennas, made to capture low frequencies in all directions. Figure 2 shows an aerial view of such an array of antennas. Instead of physically pointing the telescope, the steering is done through digital beam forming, which uses the difference in arrival time at the different antennas in the array.

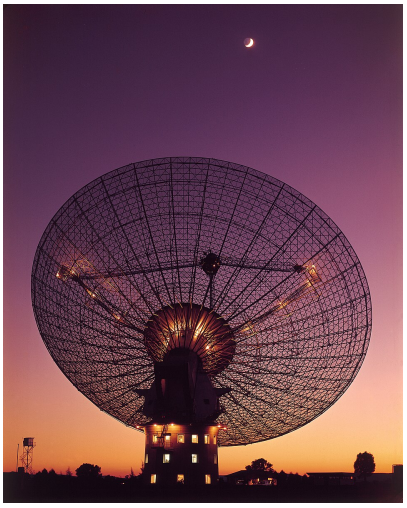

Figure 1: CSIRO's Parkes radio telescope.[1]

## 1.2 The Problem

Due to these antennas (shown in Figure 3) being omnidirectional, capturing other signals is unavoidable, especially since we also use these signals to transmit data wirelessly. These unwanted signals are called Radio Frequency Interference (RFI), and we must remove these signals in order to observe the signal from space we are interested in. Since the amount of data is so astronomically large, due to thousands of antennas, this is not possible to do manually for all signals.

Figure 2: The LOFAR Superterp, the central core of the LOFAR radio telescope array near Exloo, the Netherlands.[2]

Currently, we just record the signal and use a statistical algorithm, named AOFlagger, to flag the RFI afterward. This algorithm has been the industry standard since it was introduced over a decade ago. While there have been updates to the algorithm over the years, the core mechanism remains the same. In 2024 researchers achieved great results when they used vision transformers to flag the RFI [17]. Compared to AOFlagger the accuracy was much better. However, the transformer uses substantially more compute to achieve these results, which makes it impractical to deploy.

[1]Photo by division, CSIRO, licensed under CC BY 3.0, via Wikimedia Commons. Source: https://commons.wikimedia.org/wiki/File:CSIRO_ScienceImage_4350_CSIROs_Parkes_Radio_Telescope_with_moon_in_the_background.jpg

[2]Photo by LOFAR / ASTRON, licensed under CC BY 3.0, via Wikimedia Commons. Source: https://commons.wikimedia.org/wiki/File:LOFAR_Superterp.jpg

Due to these transformers being so compute-intensive, it is only applicable on the recorded signal after the correlator, just like how AOFlagger is used. Another way is to flag the RFI in real-time on the antenna stations. This comes with the advantage of lower data transfer and more granular flagging, since the RFI is flagged in a station's signal directly, instead of the correlated signal of multiple stations. This requires a low-power yet accurate and high-throughput flagger. In order to successfully flag in real-time, the flagger must keep up with the time resolution of the telescope. In imaging mode, which is the most common case, this is a five microsecond interval, or 200 kHz, although LOFAR supports resolutions as fine as 5 nanoseconds.[1]

## 1.3 Research Objective

The primary objective for this research is to evaluate the feasibility of low-power edge hardware accelerators (specifically, the Axelera Metis) to run deep learning models to flag the RFI in real-time. While deep learning models have demonstrated state-of-the-art accuracy in RFI flagging, their high computational complexity traditionally requires large and expensive GPU clusters. This thesis aims to bridge this gap by exploring whether optimized neural networks can be used to handle the high throughput of LOFAR, in real-time with minimal energy overhead. This work investigates the hardware compatibility of different architectures, specifically state space models and convolutional neural networks, and measures the critical trade-offs between accuracy and operational efficiency. We will compare the performance and energy efficiency to Graphics Processing Units (GPUs). Results show that state-space models are capable of detecting RFI at a fraction of the computational power normal vision transformers require, but the CUDA-optimized implementation cannot be directly exported to ONNX, which means they are not able to run on the Axelera Metis. We develop a Convolutional Neural Network (CNN) which rivals the accuracy of the vision transformer at a fraction of the computational power. This CNN does run on the Axelera Metis at roughly six times less energy per sample compared to a GPU (NVIDIA RTX A4000). At the end we will answer the following question: **How does the performance of neuromorphic hardware compare to the current state-of-the-art, in terms of latency and power efficiency for RFI mitigation?**

Figure 3: A low-band antenna (LBA) of the Low-Frequency Array (LOFAR). The LOFAR cabin containing the electronics is visible in the background right.[2]

## 1.4 Thesis overview

In Section 2 we provide the necessary background for the rest of the thesis; Section 3 discusses related work which inspired the methods evaluated in this thesis; in Section 4 we describe the setup of the experiments and why we chose these; Section 5 describes the implementation and results of

[1] These figures were provided in personal communication with our supervisors.

[2] Photo by A. R. Offringa, licensed under CC BY-SA 3.0, via Wikimedia Commons. Source: https://commons.wikimedia.org/wiki/File:A_low-band_antenna_of_LOFAR.jpg

the experiments described in the approach; Section 6 concludes. This is a bachelor thesis made under supervision of Prof. dr. Rob van Nieuwpoort and Dr. Chris Broekema under the Leiden Institute of Advanced Computer Science (LIACS).

# 2 Background

## 2.1 LOFAR

### 2.1.1 The Pipeline

The antenna receives the signal from the real world and converts it into a voltage. The voltage is then sampled by an Analog-to-Digital Converter (ADC) which converts it into a stream of discrete digital samples. This is then combined with the signals of other antennas in the same array. Next, a fixed window of values is combined into a group, which then gets processed by the Fast Fourier Transform (FFT). The FFT returns a complex number of the form $a + bi$, with $a$ being the real part and $b$ the imaginary part. With these parts together we can calculate the magnitude with the Pythagorean theorem ($\sqrt{a^2 + b^2}$). This magnitude is where the RFI will be flagged. For flagging and human interpretation we create a spectrogram. There are multiple ways to achieve this but in this paper it will have the following structure: the frequency is plotted on the x-axis of a graph, with the y-axis capturing the time dimension. At the coordinates there is a color; this color represents the relative magnitude compared to all other points. Low values are represented by cool colors with intense spikes represented by warmer colors. In Figure 4 an example is shown.

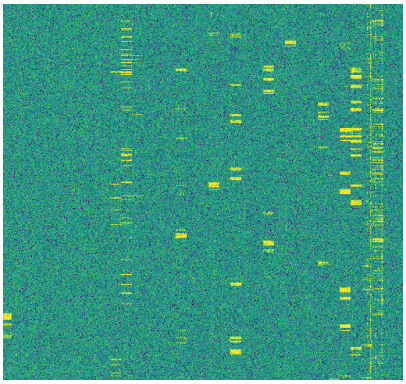

Figure 4: An example spectrogram.[3]

### 2.1.2 AOFlagger

AOFlagger is the algorithm that was introduced in 2010/2012 to flag RFI in the LOFAR pipeline. It consists of two separate algorithms, Sum Threshold [15] for detecting the RFI and Scale-Invariant Rank Operator (SIR) [16] for complementing the parts Sum Threshold missed.

**Sum Threshold** Sum Threshold works by looking at a certain window in the absolute magnitude. The absolute magnitude has a large dynamic range, so before it can be input into the function it has to be converted into a logarithmic scale. Essentially converting it into decibels (dB), which makes the absolute magnitude similar to a volume in sound. The algorithm starts with very small windows and computes the sum of the absolute magnitude in that window. If it is above the set threshold it is flagged as RFI. Then the algorithm runs again on progressively larger windows, except the threshold gets lower. The threshold is also adjusted for flagged samples in the window: if out of ten samples two were flagged in the previous round, the threshold is divided by eight instead of ten which would be used when nothing is flagged in the previous round.

[3] Spectrogram is from the test set of the dataset from the paper by Mesarcik et al. [13]

**Scale Invariance** Finally, Scale Invariance looks at all samples again. It uses an aggressiveness parameter and a point system. Each flagged sample gets a positive score, and unflagged samples get a negative score; the aggressiveness parameter is added to all values. Then all peaks and valleys in points are calculated. If a peak minus the previous valley is greater than or equal to zero all values between said valley and peak are flagged as RFI.

## 2.2 Deep Learning for Image Segmentation

In the following section we will explain the different models and parts of the model training.

**Loss Functions** Deep Learning works by adjusting model weights such that predictions created by those weights become more accurate. To calculate how wrong the model is we use loss functions. A loss value is a non-negative value, with 0 being perfectly accurate.

**Binary Cross Entropy Loss** Binary Cross Entropy (BCE) Loss works in a simple way. It takes on the shape of the logarithmic curve when the probability approaches uncertainty (0.5). This means that the loss increases steeply when the model is not confident or wrong. To get the loss value closer to zero, the model must be confident and correct.

**Dice-Sørensen Coefficient** The Dice-Sørensen Coefficient [3] compares the similarity between two samples. Mathematically it is the same as the F1-Score.

$$\text{DSC} = \frac{2|P \cap G|}{|P| + |G|}$$

The coefficient is intuitive for spectrograms with binary masks. All the matching pixels between the ground truth and the prediction are summed, and then divided by the sum of positive pixels in both masks to get a value between zero and one. This is also why the numerator is multiplied by two; it balances out doubly counting each pixel. Because G is a binary mask, the numerator can be written as $\sum_{i}^{N} P_i \cdot G_i$.

**Dice Loss** Dice loss [14] works based on the Dice-Sørensen coefficient (DSC). If we rewrite the formula to work with the masks, we get the following:

$$\text{DSC} = \frac{2 \cdot \sum_{i}^{N} P_i \cdot G_i}{\sum_{i}^{N} P_i + \sum_{i}^{N} G_i}$$

$$\text{Dice Loss} = 1 - \text{DSC}$$

DSC works inversely to normal loss values, as the maximum score is one and the minimum score is zero. Thus, to get a normal loss value we simply invert the value.

Milletari et al. [14] deviated from the normal Dice-Sørensen coefficient by squaring the probabilities and ground truth in the denominator. They do not explicitly motivate their choice to square it, but both the linear and squared variants are widely used in practice.

**RFI Segmentation** When using deep learning for image segmentation the entire image is processed at once. In spectrograms there are two options at every pixel: there is RFI or there is not. Thus, this is called binary image segmentation. The deep learning model takes in the entire image and then outputs a mask of binary values, true or false, in the same size of the image. This mask can then be used to process the data in flagged areas differently.

**Convolutional Neural Networks** A deep learning technique for image processing is a Convolutional Neural Network (CNN). CNNs mostly consist of four types of layers: Convolutional layers, Pooling layers, Activation layers, and the fully connected layer. These convolutional layers work by sliding a kernel, basically a filter, over the image. It looks at patches of pixels at a time and extracts features. A CNN can have many different convolutional layers and at each layer many different kernels which all create their own feature map. A kernel is like a filter which samples an area of an image in order to create a feature map which preserves spatial information. Pooling layers down-sample the image; they turn a patch of values into one single value. For example by looking at the max value or the average. The activation layer consists of an activation function. Activation functions are non-linear functions, which allow the model to detect non-linear relations in the input data. The fully connected layer finally connects the last layer to the outputs.

**U-Net** U-Net [22] is a fully convolutional neural network, which means all feature extraction is done by convolutions. The U-Net follows the shape of the letter U. The left side of the U consists of many convolutional layers, each followed by a ReLU activation layer and a max pool layer. This left side down-samples the image while preserving the most important features. Then the right side of the U uses this down-sampled image to up-sample back to the original image's input size, except now it is in the form of a mask with the different segments.

**Residual neural network (ResNet)** As the model gets more layers, and thus gets deeper, training the model becomes more difficult. To combat this, ResNet [7] introduced skip connections. A skip connection allows a model to bypass a layer during training by allowing a feature map to just pass through to the next layer; this is done by adding the input directly to the output.

**MobileNetV2** MobileNetV2 [23] builds upon ResNet but mainly focuses on being for low-powered edge devices. MobileNetV2 also has skip connections but where ResNet has them between compressing into a lower dimension and then up-sampling back to the original dimension, MobileNetV2 expands the dimension before compressing back to the original dimension, essentially inverting ResNet's approach. The researchers also found that ReLU discarded some valuable data, so they removed the ReLU layer at the end of the bottleneck block. Consequently, this optimization makes this model architecture highly suitable for low-powered edge devices, and it is fast and power efficient.

**Vision Transformer** The transformer [27] has shown to be a versatile deep learning technique in many fields. In computer vision it is also a very strong technique. Where the CNN used a kernel to look at a local moving window, a Vision Transformer (ViT) [4] looks at the entire image at once by dividing it into patches. Where a CNN has difficulties with correlating between pixels that are on the opposite ends, a ViT can relate them directly. A ViT does this by the self-attention function. This function takes all patches and then calculates relationships between all pairs of patches. In this way a ViT can learn to correlate different features with the wanted output. CNNs also use the same kernels on every image, whereas a ViT calculates dynamic weights based on input data and can thus process different images in different ways. A downside of this method is that it needs a large amount of data. It has to learn that pixels that are close together are correlated, which a CNN is inherently biased towards, whereas a ViT needs a massive amount of data to understand it.

**Swin-UNETR** Swin-UNETR [6] aims to reduce the computational demand of the ViT. It does this by modifying the self-attention to be more like CNNs. Where ViTs apply self-attention on all pairs of patches, Swin (Shifted Window) [10] uses a moving window of patches. So instead of comparing all pairs of patches in an image, it compares all pairs of patches inside a window. This local window approach works well with RFI mitigation. RFI usually occurs in a patch, and RFI in different patches is mostly uncorrelated. Consequently, using global self-attention would be unnecessary yet expensive computationally. Then, in order to keep correlation between windows, it shifts the window on the next layer. The computational complexity of Swin thus is $O(n)$, instead of $O(n^2)$, but all pairs in a window are still compared to each other, which means the complexity in a window remains $O(n^2)$. With window size $M$ this would be $O(n \cdot M^2) \rightarrow O(n)$. The model also follows the U-Net architecture, with the described mechanism being the encoder (down-sampler); the decoder (up-sampler) is a CNN, as in the traditional U-Net.

**Swin-UMamba** Instead of self-attention Swin-UMamba [8] utilizes a State Space Model (SSM). Rather than calculating self-attention for every subsequent layer, the SSM is a model that keeps track of the state. Because of this it can be updated in one linear pass over the input image, achieving true $O(n)$. Due to Mamba [5] being inherently 1D, there is a catch. If the 2D image is squeezed into one sequential dimension the relationship between pixels is distorted. This is why Mamba does four scans in parallel instead. It scans from all possible different directions and these directions have their own state. This creates a complexity of $O(4 \cdot n) \rightarrow O(n)$. The model also follows the U-Net architecture, with the described mechanism being the encoder (down-sampler); the decoder (up-sampler) is a CNN, as in the traditional U-Net.

**Open Neural Network Exchange (ONNX)** ONNX is an open-source standard format for neural networks. It consists of the model weights and the operations of the model as a standardized set of instructions. Thus, ONNX is often used as a format to share neural networks, or to run them on specialized hardware. Deep learning frameworks like PyTorch and TensorFlow contain functions to export models in the ONNX format. Whether a model export is supported depends on the instructions the model uses. If its implementation is highly specialized on a certain framework like CUDA, it might not be possible to export the model to the ONNX format.

**Quantization-Aware Training** If the model is to be quantized at runtime, it can be helpful to introduce the model to the noise of the quantized values during training. One way to do Quantization-Aware Training (QAT) is using fake quantization. With fake quantization, the input and output are quantized, to replicate how it would be done when the model is deployed. Thus, values are clipped; then, instead of using the actual quantized value, the value is converted back to full precision, except that there is now a precision reduction due to the value being fit to a discrete step size. Because the values are still in full-precision, the model can be trained using standard backpropagation and floating point gradients. Training with these values allows the model to adapt to the values in the discrete steps that would be used when quantized, and thus be more resistant to noise.

## 2.3 Edge Accelerators

Neural Processing Units or NPUs are special processors made to run neural networks. They achieve a huge speedup compared to CPUs, because a CPU is made to be versatile. There are hundreds to thousands of instructions a CPU can execute, while specialized NPUs only have tens of instructions in total. In this way they can optimize for executing those instructions only, which allows them to achieve better throughput. They can also utilize different ways of addressing memory which is more suitable for the type of data they process. All of these NPUs are basically chips optimized for matrix operations, as that is what neural networks and some other AI models use. Where CPUs usually only have to perform the same calculation one or a couple of times in a row, NPUs and other matrix optimized chips perform the same operation on entire matrices at once. This makes NPUs highly efficient at specific computations, thus allowing them to reach high performance-per-watt ratios. Some examples of NPUs are Google's Tensor Processing Units (TPUs), Apple's Neural Engine and the Axelera Metis.

## 2.4 Evaluation Metrics

Before analyzing the performance of RFI flagging models, some evaluation metrics for binary classifiers need to be covered.

### 2.4.1 True Positive Rate and False Positive Rate

The True Positive Rate (TPR) and False Positive Rate (FPR) measure the rate of correct and incorrect positive predictions respectively. The TPR shows whether a model is able to predict the positive class. Alone it does not give enough information, as flagging everything as positive would yield a perfect TPR of 1.0, even if the model made a substantial number of mistakes on the negative class. FPR shows this because if a model would predict everything as positive, including when it is actually the negative class, the FPR would be 1.0. When a dataset contains a heavy class imbalance towards the negative class, the FPR can be forced near-zero due to the massive amount of true negative samples being in the denominator.

### 2.4.2 AUROC

AUROC or Area Under the ROC Curve measures both these rates, and at every different threshold. Threshold here refers to when the model decides if a sample belongs to the positive or negative

class. It plots the TPR and FPR on a graph and calculates the area under the curve. If a model is great at separating the classes AUROC will be high because the model has a high confidence for both classes. When the model has low confidence, but the model by chance achieves a high accuracy at a specific threshold, the AUROC would show this by being low. If a model were to guess randomly, and thus have 0.5 TPR and 0.5 FPR the AUROC would be 0.5. Similarly, a model that perfectly separates the two classes would have an AUROC of 1.0.

#### 2.4.3 Precision

Whereas the TPR measures how many actual positive samples were caught, precision measures the rate of correct positive predictions over the total number of predicted positives. This becomes especially important when the classes are imbalanced.

#### 2.4.4 AUPRC

AUPRC utilizes another metric: recall. Recall is mathematically equivalent to the TPR described earlier. The Area Under the Precision-Recall Curve (AUPRC) works similarly to the AUROC, except that it plots precision with recall instead of FPR with the TPR for every threshold the model could use. The practical difference is that AUROC is easily misleading when both classes are heavily imbalanced; AUPRC utilizes precision instead of the FPR and thus is still accurate when the classes are imbalanced.

#### 2.4.5 F1-Score

F1-Score is the harmonic mean of precision and recall. It is similar to AUPRC in its accuracy when the classes are imbalanced, but it only measures the model's final prediction, thus using a fixed threshold, where AUPRC plots for every possible threshold.

# 3 Related Work

## 3.1 Success with NLN

In 2022 Mesarcik et al. used the Nearest Latent Neighbors (NLN) [13] deep learning technique for RFI detection. They realized that in order to train a neural network, an accurately labeled dataset of RFI is not necessary [11]. A weakly labeled dataset of recordings with RFI would work too if the model just learns what normal signals without interference look like. This means that flagging from a less accurate tool, such as AOFlagger, would be enough for the training dataset.

However, for the test dataset manually labeled data is necessary to verify whether the model has accurately learned flagging normal signals, and thus by inverse the RFI. Due to the training data being created with weak labels, using the same weak labels for the test set would result in biased results, hence why the human-labeled samples are necessary. The dataset they created used recordings from LOFAR with the RFI flagged by AOFlagger. It has 7500 samples of $512 \times 512$ spectrograms in the training set, and 109 human-labeled samples in the test set.

Their results using NLN were a considerable improvement in accuracy compared to AOFlagger. But the neural network required a large amount of computational overhead, which makes it

impractical in the real world. They achieved an AUROC of 0.8622, AUPRC of 0.6216 and F1-Score of 0.5114.

### 3.2 Success with ViTs

In 2024 subsequent work used ViTs, more specifically Swin-UNETR, for semantic RFI segmentation on the spectrograms of the LOFAR telescope [17]. The dataset they used for training was exactly the same as in the NLN work. They used three different feature sizes of the Swin model, resulting in three different numbers of parameters, 6M, 102M and 408M. These progressively larger models also required progressively larger amounts of compute, 1.1, 12.3 and 67.0 TFLOPs respectively. Their accuracy was noticeably better compared to AOFlagger, achieving a 0.9771 AUROC and 0.6401 F1-Score compared to AOFlagger's 0.7883 and 0.5698 respectively.

Notably, scaling up the model from 6M to 408M parameters had diminishing returns: the performance of the 408M model was only slightly better than the 6M model and the 102M model had a lower F1 score than the 6M model. Ultimately, while these accuracy improvements are substantial, the increased computational overhead makes it impractical for live flagging and expensive for offline flagging.

### 3.3 Success with Autoencoders

van Zyl et al. [26] demonstrated with their RFDL method that convolutional neural networks are also capable of RFI flagging with a high accuracy. They first trained an autoencoder on corrupted images. These corrupted images were created from images without RFI, then they added artificial RFI. The goal of the autoencoder was to restore the original image. They then used this autoencoder as a preprocessor for a convolutional neural network. Three experiments were tried, training on the weakly labeled AOFlagger training set, training on 20 expertly labeled samples, and training on the weakly labeled training dataset and then fine-tuning using 20 expertly labeled samples. They found that those 20 expertly labeled samples during training drastically improved accuracy. Their best results are an AUROC of 0.989, AUPRC of 0.748 and F1-Score of 0.675. However, due to these results using a different test set they are not directly comparable to the results of the other described models.

### 3.4 Success with Spiking Neural Networks

Spiking Neural Networks (SNNs) are specialized neural networks which are designed for inference on Spiking Neural Processors (SNPs). These SNPs have an advantage over traditional von Neumann-based architectures as they only respond to inputs above a certain threshold, which allows them to run at much lower clock frequencies without a reduction in computational power.

In 2024, Pritchard et al. demonstrated SNNs are capable enough to be competitive with Artificial Neural Networks [21]. Following up on their previous work, Pritchard et al. refined their approach and achieved a peak F1-Score of 0.474 [18]. Finally, they introduced an end-to-end SNN pipeline for detecting RFI [19]. They achieved competitive performance on a simulated dataset while utilizing less than 100 mW, thus enabling ultra-low power RFI detection.

Pritchard et al. investigated the existing energy usage and proposed a neuromorphic replacement to the deployed hardware [20]. They estimate that their proposed processing pipeline for LOFAR

would reduce power consumption by 88% compared to the current system. This thesis investigates whether utilizing Digital In-Memory Compute (D-IMC) based NPUs can yield energy reductions on a similar order of magnitude.

# 4 Approach

In this section, we present our approach to developing, optimizing and evaluating deep learning models for real-time RFI mitigation on resource-constrained edge hardware. The goal is to find an architecture capable of matching the accuracy of the large state-of-the-art models while keeping the required computational power low enough such that it can run efficiently in real-time on specialized edge-hardware.

To achieve this, we have to go through four steps. First we explain how we load and modify the dataset before training such that it is suitable for the deep-learning models. Second, we explore the different architectures used, namely Swin-UMamba and U-Net with a MobileNetV2 encoder. Third, we outline the exporting and compiling of our models using the ONNX format and Axelera compiler. Finally, we explain how we deployed our models to the Axelera Metis hardware, and how we evaluate the throughput, latency and power consumption alongside the accuracy metrics.

## 4.1 Dataset

For all the models developed, we adopted the approach from Mesarcik et al. [13], and we used their LOFAR dataset [12]. Thus, all performance metrics are directly comparable to the results from Mesarcik et al. For the hyperparameter tuning of the CNN we used half of the test set, so we also include performance metrics of our own test split. The approach from Mesarcik et al. is that the model does not need a perfectly labeled dataset, as long as it learns what clean signals look like it can detect RFI in that way. They split the dataset into a training set and a test set. The training set has 7500 samples flagged by AOFlagger, thus these are weak labels. In contrast, the test set contains 109 samples, but these are manually labeled by experts.

Because RFI is only a very small part of the entire spectrogram, we have to take into account the class imbalance. If we do not take appropriate measures, the model might learn a trivial solution, failing to classify RFI. We partially mitigate this by applying a larger weight to the positive class (the RFI). This makes sure that missing RFI is penalized more than marking normal signals as RFI. Since RFI is sparse, this results in a better balance between false negatives and false positives.

Another technique to mitigate the heavy class imbalance is to use a different loss function. Dice loss compares the flagged area directly with the flagged area in the ground truth. The problem with Dice loss is when the area flagged by the model has no overlap with the ground truth. This will give the maximum loss value of one, which does not provide proper guidance to the model on how to improve, in contrast to Binary Cross Entropy (BCE) loss, which will be lower if a completely wrong prediction is slightly better than the previous prediction. That is why we combine Dice loss with BCE loss. Together it gives more constructive feedback, while still optimizing the Dice coefficient. These losses are combined in a combo loss function [24], which is simply calculated as the weighted average of both losses.

For Optuna [2] hyperparameter optimization we use a validation set. The training set is weakly labeled, so taking a part from there might result in the model optimizing for the weaknesses from

AOFlagger. Thus, we had to split the test set into a validation set and a test set. Due to the small size of the test set we decided to split it 50/50. We used a random state such that the split remained consistent among all our experiments, preventing data leakage.

## 4.2 Swin-UMamba

Building upon the high accuracy with Swin-UNETR from Ouyang et al. [17], Swin-UMamba is an evolution of Swin-UNETR. Since the results from Swin-UNETR already proved transformers are capable of detecting RFI, we explored the efficacy of Swin-UMamba. We expect that the linear scaling of Mamba also makes it highly suitable for real-time RFI detection. There was some uncertainty whether Swin-UMamba would achieve the same accuracy, as the SSM system is fundamentally different from traditional self-attention. Swin-UMamba is also less compute intensive during training than Swin-UNETR, but training from scratch was outside the computational budget, therefore we used ImageNet pretrained weights. The pretrained weights are already capable of extracting features, just not specifically on these LOFAR spectrograms. The hypothesis was that if Swin-UMamba is capable of detecting RFI, pretrained weights would show this; we could always train from scratch later to see if it can achieve better performance.

## 4.3 CNN Approach

Due to compatibility mismatches between Mamba and the Axelera hardware, we had to use a different model architecture. The U-Net architecture has proven to be exceptional at segmentation accuracy. The original U-Net has been tried before. However, accuracy was not competitive [1]. Therefore, we chose to use a U-Net with a ResNet50 encoder, and later with a MobileNetV2 encoder.

## 4.4 Axelera Metis

The Axelera Metis is an edge accelerator for AI. This is a system with a chip that is highly specialized, and thus is efficient at running AI models. It does this by utilizing Digital In-Memory Compute (D-IMC). This means that the model weights are stored directly in the computational unit and the computation takes place directly in memory, reducing the need for transferring weights. Transferring data in and out of the computational unit is one of the largest bottlenecks for AI compute in other architectures like GPUs.

To run a model on the Axelera Metis we use their Voyager SDK. This SDK comes with a compiler that can compile the ONNX format into the format their hardware requires for running the model. This supports most ONNX files but not all ONNX instructions are supported.

## 4.5 Performance Metrics

We use AUROC, AUPRC and F1-Score to measure the classification accuracy. We measure efficiency across the edge-accelerator and an NVIDIA RTX A4000 GPU. To measure efficiency, we use samples per second to measure the total throughput rate. One sample is one $512 \times 512$ spectrogram. We measure power usage in two different ways. For the edge-accelerator we measure the total board power at the outlet. We measure the average idle usage which we then subtract from the

power usage under load. This method of power monitoring for the edge-accelerator will be slightly inaccurate, because CPU power and the power supply inefficiencies are included. However, since the CPU is an ultra-low-power ARM Cortex-A53, we expect this delta to be negligible. In fact, the thermal footprint of this CPU is so low that it relies on passive cooling, without a heat sink. For the GPU we use the power value reported by the NVIDIA driver through the NVIDIA-SMI tool.

# 5 Experiments

This section evaluates the performance and efficiency of Swin-UMamba and U-Net with a MobileNetV2 encoder.

## 5.1 Swin-UMamba

We will explain what experiments with Swin-UMamba we have tried. Then we will discuss the results, and finally we will explain why we have chosen to stop further development of this model.

### 5.1.1 Implementation

Our initial experiments utilized the Swin-UMamba [8] architecture, which was initially developed for medical image segmentation. We initialized the model with ImageNet pretrained weights from VMamba [9], and then we trained the model on the RFI dataset.

We used the AdamW optimizer with a learning rate of $10^{-5}$, and the previously described combo loss function. Due to the highly applicable features from the VMamba ImageNet weights, the model rapidly converged in nine epochs. The model has a high segmentation accuracy here, better than AOFlagger, but slightly worse than Swin-UNETR.

### 5.1.2 Hyperparameter Exploration

We manually explored multiple different hyperparameters configurations to improve the accuracy, but we made no significant improvements. The evaluated configurations include: a frozen encoder, different loss functions, different hyperparameters and label smoothing.

The hypothesis behind the frozen encoder was that because the encoder was already pretrained on ImageNet, it is already good at extracting features. We confirmed the hypothesis as the frozen model achieved similar accuracy as the unfrozen model, though it did not exceed it. We also tried leaving the encoder frozen for ten epochs before unfreezing it, such that the decoder can train while the encoder retains the ImageNet features it has learned. Then the encoder is unfrozen and can learn more appropriate features for the decoder, which is now already creating proper masks. This, however, also made a negligible difference in the flagging accuracy.

We did not try further optimization due to an architectural shift necessitated by constraints of the Axelera Metis, which is detailed in Section 5.1.5.

### 5.1.3 Computational Requirement

The main reason we chose Swin-UMamba was because of its similarity to Swin-UNETR but with a much smaller computational footprint. In total this model has 27M parameters which means it has

just over four times the number of parameters as the smallest model from Ouyang et al. [17], yet it requires thirty-five times less compute. The model configuration and required number of TFLOPs can be found in Table 1 along with the results from Ouyang et al.

The feature size is the number of channels an image patch gets converted into in the first stage. Hence, a feature size of 24 results in the network projecting the low-dimensional input into a 24-dimensional feature vector. The blocks per stage is the number of sequential transformer layers per stage of the network. Stage 3 contains the most blocks because here the remaining image resolution is low; thus, sequential blocks can be processed in a computationally efficient manner. Stacking more blocks in an earlier stage would drastically increase the required number of FLOPs.

Table 1: Model Configurations

| **Model** | **Parameters** | **Feature Size** | **Blocks per Stage** | **TFLOPs** |
|---|---|---|---|---|
| Swin-6M [17] | 6.5 M | 24 | (2, 2, 18, 2) | 1.105 |
| Swin-100M [17] | 102.1 M | 96 | (2, 2, 18, 2) | 12.334 |
| Swin-400M [17] | 408.1 M | 192 | (2, 2, 18, 2) | 67.025 |
| Swin-UMamba | 27.5 M | 96 | (2, 2, 9, 2) | 0.0317 |

#### 5.1.4 Results

Most of the different hyperparameters we tried resulted in about the same performance. Although the performance never surpassed the segmentation performance from Swin-UNETR, it is competitive with Swin-UNETR at only a fraction of Swin-UNETR's computational requirement. See Table 2 for the results of Swin-UMamba, along with the results of AOFlagger and Swin-UNETR. Swin-UMamba performance is from the combo loss with equal weights for Dice loss and BCE loss; we did not include other configurations for the Swin-UMamba model, as their performance was comparable. In Figure 5 the captured signal with RFI can be seen together with the model output masking the RFI.

Table 2: Performance Comparison Between Swin-UMamba and Different Swin-UNETR Model Configurations

| **Metric** | **AOFlagger [17]** | **Swin-6M [17]** | **Swin-100M [17]** | **Swin-400M [17]** | **Swin-UMamba** |
|---|---|---|---|---|---|
| AUROC | 0.7883 | 0.9711 | 0.9743 | 0.9771 | 0.9666 |
| AUPRC | 0.5716 | 0.6783 | 0.6831 | 0.6938 | 0.5893 |
| F1-Score | 0.5698 | 0.6302 | 0.6281 | 0.6401 | 0.6074 |

#### 5.1.5 Edge Hardware Deployment

Before optimizing performance, we tried to deploy the model on the Axelera Metis. For the Axelera compiler we need an ONNX file, which can be generated through PyTorch. Swin-UMamba replaces self-attention with the Mamba selective state-space algorithm. This algorithm mainly depends on the selective_scan function. The selective_scan function, however, is highly optimized for

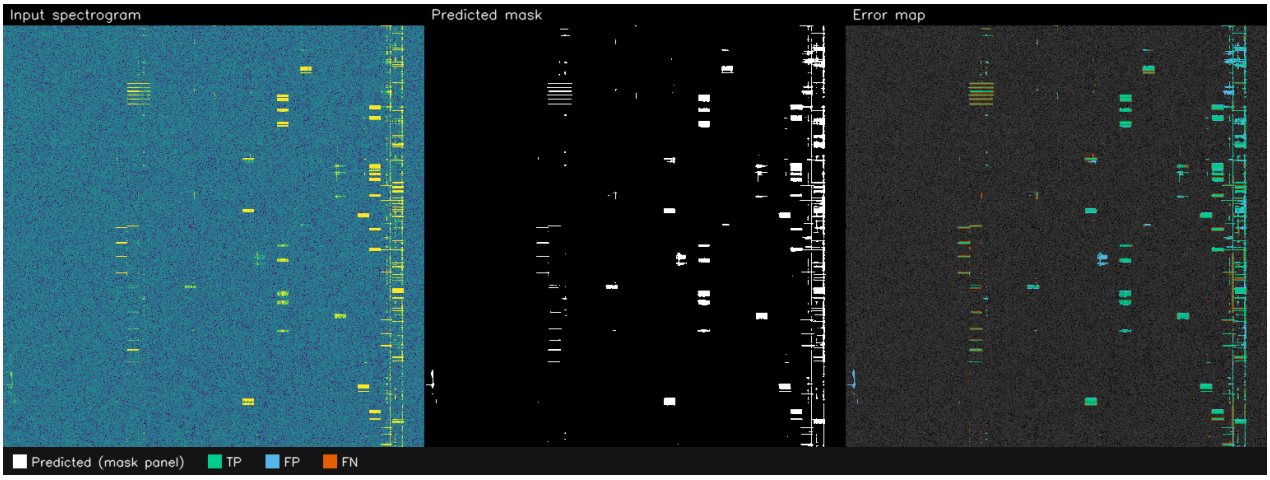


Figure 5: Left is the signal captured by the telescope, middle is the predicted RFI mask, and the right is the error map which highlights how the predicted mask compares to the expert labeled ground truth (Produced by Swin-UMamba).

CUDA. Unlike standard architectures which contain both CPU/GPU implementations, the Mamba selective_scan function is programmed in the highly specific CUDA language and takes advantage of the architecture of NVIDIA GPUs. Because of this, the PyTorch ONNX exporter cannot convert this function into the ONNX format, which causes the ONNX export to fail. There is a second selective_scan function in the library which they have provided as a reference implementation, though it is in pure Python and not optimized. ONNX export works when exporting with this reference function instead of the CUDA function.

During compilation with the Axelera compiler on the ONNX file created by the reference selective_scan function, the compiler failed due to ONNX instructions used which are not supported by the Axelera compiler.

Porting the selective_scan function from CUDA to a variant compatible with the Axelera compiler is the most direct solution for running Swin-UMamba on the Axelera Metis. Another repository, mamba.py [25], provides an ONNX compatible version of the selective_scan function, although implementing it in Swin-UMamba would require more than simply importing the function from mamba.py. Instead, we prioritized exploring other architectures to establish a performance baseline before committing to porting the function to be compatible with the Axelera Metis. Consequently, development shifted towards a U-Net architecture with a MobileNetV2 encoder. Given the strong performance and high compatibility with the Axelera Metis of the MobileNetV2 architecture, MobileNetV2 superseded further development of Swin-UMamba on Axelera Metis.

## 5.2 Convolutional Models

Given the Axelera Metis' heavy optimization for convolutional neural networks, we developed multiple U-Net models with convolutional encoders. We will describe the different model architectures we experimented with and explain the limitations we faced when executing the model on edge hardware.

### 5.2.1 Training

**Initial Training** The original U-Net performance on RFI detection was not great [1]. Consequently, we tried a ResNet50 encoder with the U-Net architecture. Since Swin-UMamba achieved good performance metrics we reused our training pipeline. Thus, the configuration was with the combo loss function combining BCE and Dice loss with an equal weighted average and a positive class weight of 9 to combat the class imbalance. Resulting performance metrics were similar to the original U-Net, except AUROC was much higher (0.95). Due to this result, we decided to try to further improve performance with convolutional neural networks, specifically MobileNetV2 due to size constraints.

To further optimize model performance, we split the test set into a validation and test set for hyperparameter tuning using Optuna [2]. The Optuna study requires a function to optimize. For this we chose a custom composite score. We defined composite score as follows:

$$0.2 \cdot \text{AUROC} + 0.4 \cdot \text{AUPRC} + 0.4 \cdot \text{F1-Score}$$

For the F1-Score we use a fixed threshold of 0.5, instead of the threshold that maximizes the F1-Score.

We chose this score instead of the loss because we also tried different weights for the combo loss. This makes the loss an unreliable metric to compare different trials. We gave the AUROC a lower weight because it was already high, and it is the easiest metric to achieve a high score, as it measures false positives and the positive class is sparse. If AUROC were to decrease a bit while AUPRC and F1-Score improved that would be a valuable trade-off for RFI mitigation. Training is stopped once the composite score on the validation set has peaked and not improved in the following five epochs; this epoch is the model we then use.

**Quantization-Aware Training** The quantized model on the Axelera hardware does run at a reduced accuracy. The input values in the spectrogram are in dB, which utilizes a logarithmic scale. Due to the signal never being below one, unless it is zero which suggests it is dropped or removed earlier in the pipeline, the input values are strictly positive. The hardware utilizes signed 8-bit integers, and the compiler cannot quantize the strictly positive values into negative values. This means that in the best case only half of the input range is utilized, which means it is effectively quantized to 7 bits. In 32-bit floats there are enough positive numbers such that negatives are not necessary to get a big dynamic range. However, 8-bit integers only allow for 128 positive values.

To mitigate this we normalize the input during training and convert it to a range of -1.0 to 1.0. The minimal signal strength was zero, though this means the signal is dropped. We chose a range of [43, 90] dB in order to capture as much dynamic range as possible. If we exclude zero values, this would capture 99.97% of all signals in the training dataset; including missed zero values, it captures 99.34% of all signals. During training and inference we convert all values to the range [43, 90] by clipping. Then we normalize the values using min-max normalization. Because the values do remain strictly positive, we multiply the normalized values by two and then subtract one, such that all values are between [-1.0, 1.0].

Due to reduced performance after quantizing, we trained the model again but with fake quantization on the model input and output using the exact quantization parameters used by the Axelera compiler. This increased the performance, which is now competitive with the full precision model.

### 5.2.2 Deployment

The Axelera compiler could not compile the ResNet50 model due to size constraints. Therefore, we evaluated smaller architectures, including ResNet34, ResNet18, and MobileNetV2. We selected MobileNetV2 for its smaller size, yet it performed better than ResNet50. We also successfully compiled MobileNetV2 using the Axelera compiler.

Due to limitations on the hardware side the input and outputs have to be padded to a shape that is appropriate for the hardware. The Axelera compiler calculates the required padding automatically. For our MobileNetV2 U-Net however, the output channels are padded to 64, from one. This means the output of the model is 64 times larger than necessary. The total output tensor size, which contains the spectrogram mask, is 16 MiB.

Actual real-world PCIe read speeds of the exact System on a Chip (SoC) in the Arduino Portenta x8 we use are not recorded. However, NXP confirmed the expected PCIe read speed of a similar processor to be around 230 MB per second, due to overhead. In NXP Application Note AN13164, NXP describes PCIe bandwidth limitations for different processors, but there is a note in section 6 that says that the same limitation is present in the i.MX 8M Mini, which is the processor in the Arduino Portenta. Furthermore, in a community post, NXP TechSupport shares read and write speeds for two of their processors supporting up to PCIe gen 2 (Note that in this post write and read speed is from the perspective of the PCIe controller to the CPU, and thus inverted to what we would normally describe as PCIe write and read bandwidth).

Our transfer speeds are higher: 16.19 samples per second with 16 MiB output tensors gives $16.19 \cdot 16.0 = 259.0\text{MiB/s} = 271.6$ MB/s. We do not know why the PCIe bandwidth is substantially higher than the speed of the other NXP processors, but we still suspect PCIe bandwidth to be the bottleneck. When we run the Metis in PCIe gen 1, instead of gen 2, the throughput is reduced by 52.5%. This is almost the exact difference in throughput between these PCIe generations, which is 50%. Therefore, we conclude that transfer speed is the bottleneck. The Metis itself supports greater speeds, specifically PCIe gen 3 with four lanes. Our device, the Arduino Portenta x8, is limited to PCIe gen 2 on a single lane. See Figure 6 for a visual breakdown of the output tensor.

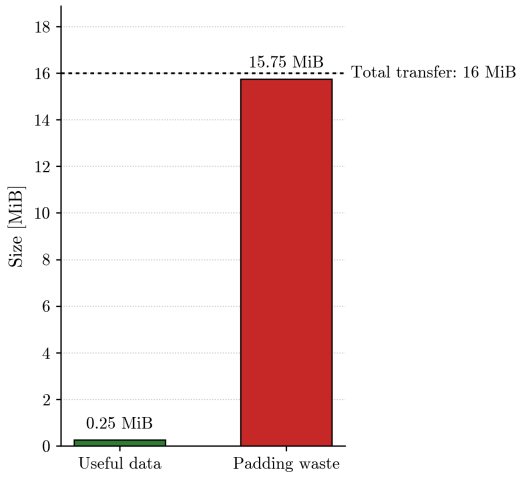


Figure 6: Useful payload versus padding overhead for a single transfer.

#### 5.2.3 Computational Requirements

The model has 6.6M parameters but due to it being a convolutional model it does not compare all input values using self-attention. Thus, it still requires considerably less compute than the 6M Swin-UNETR model. The number of TFLOPs required is 0.01365, eighty times less compute than Swin-UNETR 6M and around two times less than our Swin-UMamba model. Compared to the model with the closest performance, Swin-400M, the required compute for the CNN is around five thousand times lower.

#### 5.2.4 Results

The results of the model are on par with the performance of the Swin-400M model, but require a fraction of the computational power. In order to run the model on edge hardware, like the Axelera Metis, it often needs to be quantized to 8-bit integers (INT8), which introduces a small reduction in the accuracy. Therefore, the table below also includes the 8-bit integer accuracy; we calculated these metrics on the Metis. In Figure 8 the AUROC, AUPRC and F1-Score per epoch during training are plotted. Epoch 15 was the last epoch with an improvement to the composite score, which is why it stopped. Validation set performance is slightly higher than the test set, but we expected this because the hyperparameters are tuned to the validation set. See Table 3 for the performance metrics of the model.

The 16.19 samples the Axelera Metis can produce are $512 \times 512$ spectrograms; thus, the total throughput consists of these samples per second multiplied by 512, which results in $16.19 \cdot 512 = 8289$. This means the entire pipeline can process a sample rate of 8289 Hz, which is a resolution of $\frac{1}{8289} = 0.0001206$ s = 120.6 microseconds. In Figure 7 a flagged example can be seen.

In Table 4 the performance comparison between the GPU and the Metis is shown. The latency of the Metis includes the PCIe transfer time of the large output tensor. Technically the axmonitor tool is not officially supported on the Arduino Portenta x8, but it does work on our system. We have recorded the values during 30 minutes of inference and calculated the mean total inference runtime per core, which was 58.2 ms per second. We are using 4 instances of the model on the four different cores, which means they are executing in parallel. Since there are 16.19 samples per second on four separate cores, this means the mean total runtime is $58.2/(16.19/4) = 14.4$ ms per sample on a core. This indicates the high end-to-end latency is predominantly caused by the PCIe transfer speed, rather than the computation on the Metis itself. If we theoretically derive the throughput from this latency: $\frac{1000 \text{ ms}}{14.4 \text{ ms}} = 69.4$ samples per second per core. If we assume it scales to four cores the throughput would be $69.4 \cdot 4 = 278$ samples per second. Due to flagging 512 rows at once it means the sample rate would be $278 \cdot 512 = 142{,}336$ Hz, which is a time resolution of 7 microseconds. This comes close to the required 5 microseconds for imaging mode, but does not achieve it. Axelera also makes PCIe cards with four times as many cores, so if we extend the theoretical throughput to four times as many cores the resolution would further decrease to 1.76 microseconds. This is below the required 5 microseconds and even has a good margin to account for real-world performance, as real-world throughput will most likely be lower than this extrapolation.

#### 5.2.5 Energy Efficiency

The energy efficiency is hard to calculate as the Axelera Metis does not provide a direct power usage. As mentioned before, we calculate the Axelera Metis power as power under load − idle power. GPU

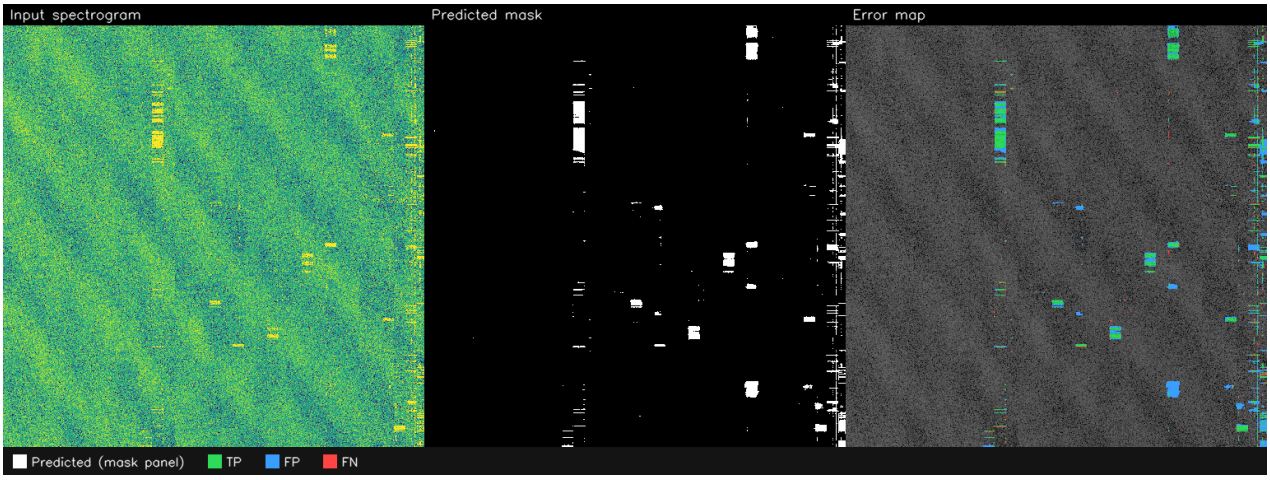


Figure 7: Left is the signal captured by the telescope, middle is the predicted RFI mask, and the right is the error map which highlights how the predicted mask compares to the expert labeled ground truth (Produced by the MobileNetV2 U-Net model on the Axelera Metis).

power is the value from the driver. We calculate the throughput as Samples per Second, the total number of spectrograms per second. Both devices are processing entire $512 \times 512$ frames at a time.

Table 3: Performance Comparison of Our MobileNetV2 CNN Approach Against State-of-the-Art Techniques, where bold is best

| **Metric** | **AOFlagger** [17] [a] | **Swin-400M** [17] | **RFDL** [26] [b] | **Swin-UMamba** | **MobileNetV2** [c] | **MobileNetV2 INT8** |
|---|---|---|---|---|---|---|
| AUROC | 0.7883 | 0.9771 | 0.975 | 0.9666 | **0.9858** | 0.9851 |
| AUPRC | 0.5716 | 0.6938 | 0.662 | 0.5893 | **0.7147** | 0.7104 |
| F1-Score | 0.5698 | 0.6401 | 0.634 | 0.6074 | **0.6545** | 0.6544 |

[a] Values in the table are from Ouyang et al. [17], AOFlagger is described by Offringa et al. [16].
[b] The values reported for RFDL [26] are from when they trained with the same methodology using a weakly labeled training set. When they used their own expert labeled dataset they achieved better metrics (0.989 / 0.748 / 0.675) on their test split.
[c] We used one half of the test set as a validation set for hyperparameter optimization. On the unseen test split the model achieved the following metrics: 0.9837 / 0.7033 / 0.6365.

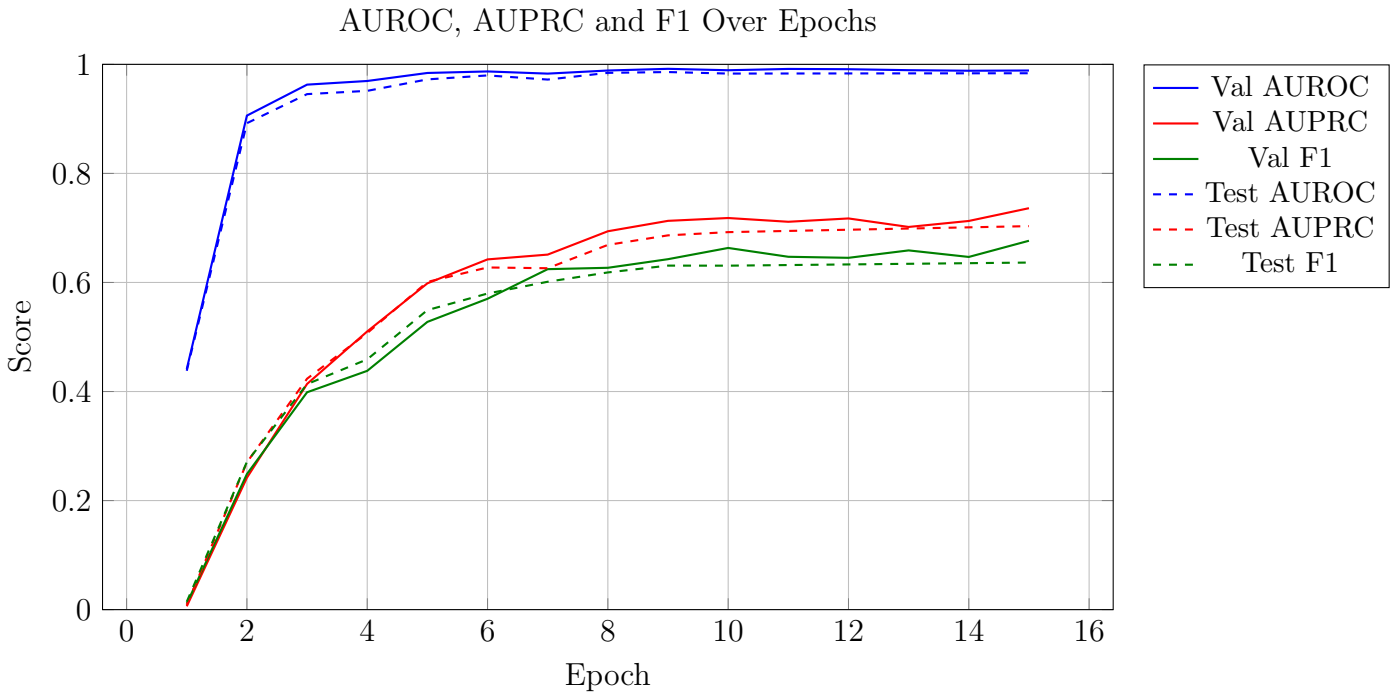


Figure 8: Validation and test AUROC, AUPRC and F1 across training epochs. Best validation composite score at epoch 15.

Table 4: Hardware Efficiency and Performance Comparison between Edge Hardware and GPU. Note that Axelera latency includes the transfer of the large output tensor. Presented metrics are mean values over at least thirty minutes of inference.

| **Metric** | **RTX A4000 (Baseline)** | **Axelera** |
|---|---|---|
| Energy per Sample (mJ) | 1056.1 | **160.6** |
| Throughput (Samples per Second) | **129.72** | 16.19 |
| Power Consumption (W) | 137 | **2.6** |
| Relative Efficiency | 1.00× | **6.58**× |
| Mean Latency (ms) | **6.65** | 234.11 |
| 99th percentile Latency (ms) | **6.76** | 236.05 |

# 6 Conclusions and Further Research

This thesis investigated whether the Axelera Metis is capable of running competitive models at ultra-low power and low-latency for RFI flagging on spectrograms of the LOFAR telescope.

## 6.1 Conclusions

Both the Swin-UMamba and MobileNetV2 U-Net model architectures are suitable for efficient RFI flagging while achieving comparable or higher accuracy than ViTs. The Swin-UMamba architecture is currently less suitable for Axelera Metis as Swin-UMamba models cannot be directly exported to ONNX. MobileNetV2 with the U-Net architecture demonstrates a high level of performance for RFI flagging by CNNs, namely an AUROC of 0.9858, AUPRC of 0.7147 and F1-Score of 0.6545. It is unclear whether it is possible to run the model on the Axelera Metis with sufficient throughput for real-time flagging as our system was not capable of achieving the speeds required. When we used the latency, as reported by the monitor, and extended it to a larger system with four times as many cores, the theoretical throughput was high enough. Further evaluation is needed to verify whether the throughput scales linearly as we assumed; thus, it is still uncertain whether the Axelera Metis can flag in real-time. However, it did achieve ultra-low energy usage at a negligible reduction in accuracy ($\Delta < 0.005$ on all metrics).

Our results on the Axelera Metis achieve an improvement in energy efficiency of a similar order of magnitude to the estimated 88% from Pritchard et al., as a 6.58× relative efficiency is roughly an 85% reduction. We recognize the estimate is for the entire system, whereas ours measures a single part of the pipeline relative to a GPU baseline, so the two are not directly comparable. Nonetheless, this indicates that utilizing CNN inference on an NPU, instead of a spiking neural processor, can achieve a similar energy reduction. This suggests that both spiking neural processors and D-IMC-based accelerators are viable solutions for ultra-low-power RFI mitigation.

## 6.2 Limitations

There are several limitations to this thesis' approach. The weak training labels might be the biggest limitation, proven by the results from RFDL [26]. The test set is also small and partially used for hyperparameter tuning. While this harms the results less than direct training on these samples, it is still a form of data leakage. Ideally a separate expert-labeled validation set would be created, or the other models should be evaluated on a subset of the test set such that there is a separate validation set.

Another limitation is the Arduino Portenta x8. The slow PCIe bus is most likely a limiting factor for the throughput and latency. A Metis PCIe card will likely yield considerably higher throughput and lower latency. It would also allow the theoretical extrapolation to be verified, and would establish whether real-time flagging on the Axelera Metis is feasible.

## 6.3 Further Research

Specific model architectures for RFI flagging remain an area of interest for future work. There might be CNN architectures capable of reaching higher accuracies, like RFDL [26]. In order to improve throughput on the Arduino Portenta x8 with Axelera Metis, transferring only the sliced

outputs, omitting padding which is ultimately discarded, is an area of interest. It would reduce the output tensor size and thus increase throughput. Another possible approach would be to modify the model size in order to achieve a better ratio between padded channels and the real output. For example, the input could be folded into multiple channels: instead of $512 \times 512$ on a single channel, $64 \times 64$ on 64 channels.

Testing the performance on a Metis PCIe card in a performant system would show the real capacity of the Metis as it will not be limited by second generation PCIe x1 speeds.

The code is available at: https://gitlab.com/nboom/bachelor-thesis/

# References


[1] J. Akeret, C. Chang, A. Lucchi, and A. Refregier. Radio frequency interference mitigation using deep convolutional neural networks. *Astronomy and Computing*, 18:35–39, 2017. ISSN 2213-1337. doi: 10.1016/j.ascom.2017.01.002. URL https://www.sciencedirect.com/science/article/pii/S2213133716301056.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[3] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3): 297–302, 1945. doi: 10.2307/1932409. URL https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

[5] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2024. URL https://arxiv.org/abs/2312.00752.

[6] Ali Hatamizadeh, Vishwesh Nath, Yucheng Tang, Dong Yang, Holger Roth, and Daguang Xu. Swin unetr: Swin transformers for semantic segmentation of brain tumors in mri images. *arXiv preprint arXiv:2201.01266*, 2022. URL https://arxiv.org/abs/2201.01266.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. doi: 10.1109/cvpr.2016.90. URL http://arxiv.org/abs/1512.03385.

[8] Jiarun Liu, Hao Yang, Hong-Yu Zhou, Yan Xi, Lequan Yu, Yizhou Yu, Yong Liang, Guangming Shi, Shaoting Zhang, Hairong Zheng, and Shanshan Wang. Swin-umamba: Mamba-based unet with imagenet-based pretraining. *arXiv preprint arXiv:2402.03302*, 2024. URL https://arxiv.org/abs/2402.03302.

[9] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024. URL https://arxiv.org/abs/2401.10166.

[10] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021. URL https://arxiv.org/abs/2103.14030.

[11] Michael Mesarcik, Albert-Jan Boonstra, Elena Ranguelova, and Rob V van Nieuwpoort. Learning to detect radio frequency interference in radio astronomy without seeing it. *Monthly Notices of the Royal Astronomical Society*, 516(4):5367–5378, 11 2022. ISSN 0035-8711. doi: 10.1093/mnras/stac2503. URL https://doi.org/10.1093/mnras/stac2503.

[12] Michael Mesarcik, Albert-Jan Boonstra, Rob van Nieuwpoort, and Elena Ranguelova. Learning to detect rfi in radio astronomy without seeing it, June 2022. URL https://doi.org/10.5281/zenodo.6724065.

[13] Michael Mesarcik, Elena Ranguelova, Albert-Jan Boonstra, and Rob V. van Nieuwpoort. Improving novelty detection using the reconstructions of nearest neighbours. *Array*, 14:100182, 2022. ISSN 2590-0056. doi: 10.1016/j.array.2022.100182. URL http://dx.doi.org/10.1016/j.array.2022.100182.

[14] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. *CoRR*, abs/1606.04797, 2016. URL http://arxiv.org/abs/1606.04797.

[15] A. R. Offringa, A. G. de Bruyn, S. Zaroubi, and M. Biehl. A lofar rfi detection pipeline and its first results, 2010. URL https://arxiv.org/abs/1007.2089.

[16] A. R. Offringa, J. J. van de Gronde, and J. B. T. M. Roerdink. A morphological algorithm for improving radio-frequency interference detection. *Astronomy & Astrophysics*, 539:A95, February 2012. ISSN 1432-0746. doi: 10.1051/0004-6361/201118497. URL http://dx.doi.org/10.1051/0004-6361/201118497.

[17] Xiaowei Ouyang, Henk Dreuning, Michael Mesarcik, and Rob V. van Nieuwpoort. Hierarchical vision transformers for rfi mitigation in radio astronomy. In *Proceedings of the 6th RFI Conference (RFI 2024)*, October 2024.

[18] Nicholas J Pritchard, Andreas Wicenec, Mohammed Bennamoun, and Richard Dodson. Spiking neural networks for radio frequency interference detection in radio astronomy. *Communications Physics*, 8:517, 2025. doi: 10.1038/s42005-025-02420-7.

[19] Nicholas J. Pritchard, Andreas Wicenec, Richard Dodson, Mohammed Bennamoun, and Dylan R. Muir. Neuromorphic astronomy: An end-to-end snn pipeline for rfi detection hardware, 2025. URL https://arxiv.org/abs/2511.16060.

[20] Nicholas J. Pritchard, Richard Dodson, and Andreas Wicenec. The potential impact of neuromorphic computing on radio telescope observatories, 2026. URL https://arxiv.org/abs/2601.07130.

[21] N.J. Pritchard, A. Wicenec, M. Bennamoun, and R. Dodson. Rfi detection with spiking neural networks. *Publications of the Astronomical Society of Australia*, 41:e028, 2024. doi: 10.1017/pasa.2024.27.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[23] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. doi: 10.1109/cvpr.2018.00474. URL http://arxiv.org/abs/1801.04381.

[24] Saeid Asgari Taghanaki, Yefeng Zheng, S. Kevin Zhou, Bogdan Georgescu, Puneet Sharma, Daguang Xu, Dorin Comaniciu, and Ghassan Hamarneh. Combo loss: Handling input and output imbalance in multi-organ segmentation, 2021. URL https://arxiv.org/abs/1805.02798.

[25] Alexandre Torres-Leguet. mamba.py: A simple, hackable and efficient mamba implementation in pure pytorch and mlx., 2024. URL https://github.com/alxndrTL/mamba.py.

[26] Daniel J van Zyl and Trienko L Grobler. Remove first detect later: a counter-intuitive approach for detecting radio frequency interference in radio sky imagery. *Monthly Notices of the Royal Astronomical Society*, 530(2):1907–1920, 05 2024. ISSN 0035-8711. doi: 10.1093/mnras/stae979. URL https://doi.org/10.1093/mnras/stae979.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.