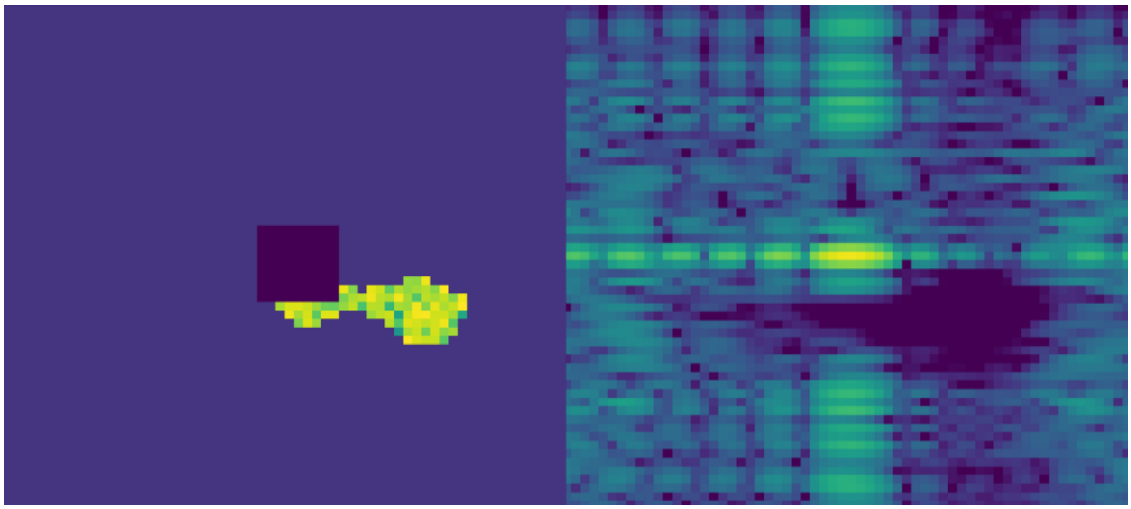


Machine Learning for Diverse and Adaptive Waveform Generation

Development of Neural Networks for Coherent MIMO Radar Waveform Generation for Clutter Suppression

Master's thesis in Complex Adaptive Systems

PER-OLA GRADIN

GABRIEL MELIN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Machine Learning for Diverse and Adaptive Waveform Generation

Development of Neural Networks for Coherent MIMO Radar Waveform Generation for Clutter Suppression

Per-Ola Gradin
Gabriel Melin

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2026

Machine Learning for Diverse and Adaptive Waveform Generation

Development of Neural Networks for Coherent MIMO Radar Waveform Generation for Clutter Suppression

PER-OLA GRADIN
GABRIEL MELIN

Supervisor: Patrik Dammert, Saab AB
Examiner: Tomas Mackelvey, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Side-by-side view of a clutter map and an ambiguity function suppressing returns from a strong clutter region.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Machine Learning for Diverse and Adaptive Waveform Generation
Development of Neural Networks for Coherent MIMO Radar Waveform Generation for Clutter Suppression
PER-OLA GRADIN
GABRIEL MELIN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Multiple-input multiple-output (MIMO) radar systems offer high flexibility in the design of transmit waveforms, something that can be leveraged for several different radar objectives. This thesis focuses on using neural-network based approaches for designing waveforms that suppress unwanted signal returns (clutter), while maintaining strong target returns. The waveform design problem is formulated around shaping the range-angle ambiguity function, where the objective is to maximize signal-to-clutter-noise ratio (SCNR) under a soft constraint on the bandwidth of the phase-coded waveforms. Unlike conventional optimization-based methods of waveform design, the proposed approach aims to train neural networks capable of generalizing to arbitrary, previously unseen clutter environments at test-time.

Several neural network architectures are explored throughout this thesis, namely Convolutional Autoencoders, Vision Transformers, Residual Networks, and Diffusion-based models. The models are trained and evaluated on synthetically generated clutter maps, with varying waveform dimensionalities and clutter distributions. The performance is evaluated using held-out test sets of clutter maps, with corresponding optimization-based waveforms as a benchmark. Additionally, methods for generating diverse sets of waveforms for the same clutter scenario are explored, motivated by the non-convexity of the waveform design objective as well as the potential mitigation against adversarial identification and counter-measure techniques.

The results demonstrate that neural networks can achieve performance comparable to the provided optimization-based benchmark on previously unseen clutter scenarios, particularly for clutter environments with fewer and larger regions of clutter. Models trained directly using differentiable waveform performance objectives are shown to significantly outperform a supervised training approach. Moreover, the results show that diverse waveform generation is achievable, although there is an indication of a tradeoff between achieved diversity and average waveform performance, and the effective diversity of the generated waveforms is shown to be significantly dependent on the method of generating multiple outputs.

Overall, the findings indicate that machine learning and neural networks offers a promising strategy for adaptive MIMO radar waveform synthesis, while also highlighting some important limitations and challenges, with room for future research.

Keywords: Machine Learning, MIMO Radar, Waveform Design, Clutter.

Acknowledgements

We would like to give a special thanks to our supervisor at Saab AB, Patrik Dammert, for all the guidance, encouragement and support throughout this thesis work. A big thank you also goes to Stefan Eilertsen, who has helped us with valuable insights and guidance throughout our work. Moreover we would like to express our gratitude to Saab AB in general, we have been very well received and are grateful for the opportunity to work in a welcoming and inspiring environment. We also want to thank our examiner Tomas McKelvey at Chalmers University of Technology for taking on this project, allowing us to finish our master's program with an interesting and challenging problem.

As this project brings our studies at Chalmers to a conclusion, we would also like to show our appreciation for all the friends we made along the way and all the people we've met who helped us out through this journey. Finally, a huge thanks to our families and loved ones, your support has been invaluable.

Per-Ola Gradin and Gabriel Melin, Gothenburg, May 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis:

MIMO	Multiple-input multiple-output
SCNR	Signal-to-clutter-noise ratio
SINR	Signal-to-interference-plus-noise ratio
ISL	Integrated sidelobe levels
AF	Ambiguity function
CM	Clutter map
WISL	Weighted integrated sidelobe levels
AESA	Active electronically scanned array
PRI	Pulse repetition interval
RCS	Radar cross section
SNR	Signal-to-noise ratio
FFT	Fast Fourier Transform
ML	Machine Learning
IML	Informed Machine Learning
PIML	Physics-Informed Machine Learning
MLP	Multi-layer perceptron
MSA	Multi-head self-attention
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SSL	Self-Supervised Learning
CAE	Convolutional Autoencoder
ViT	Vision Transformer
ResNet	Residual Network
MSE	Mean Squared Error

Nomenclature

Below is the nomenclature of common parameters and variables that have been used throughout this thesis.

Parameters

M	Number of antenna elements
N_b	Number of fast-time sample points
N_u	Number of discrete angle bins
N_r	Number of discrete fast-time (range) bins
K	Number of model-generated waveforms

Variables

$\mathbf{\Phi}$	Waveform phase-coding matrix
$\mathbf{S}$	Waveform matrix
θ	Generic look-angle
θ_0	Target look-angle
u	Parametrized look-angle ($u = \sin(\theta)$)
u_0	Parametrized target look-angle ($u_0 = \sin(\theta_0)$)
$\mathbf{a}(\theta)$	Transmit steering vector at look-angle θ
$\mathbf{b}(\theta)$	Receive steering vector at look-angle θ
$\mathbf{S}_\tau$	Time-shifted waveform matrix
$\mathbf{CM}$	Clutter map matrix
$\mathbf{AF}$	Discrete ambiguity function matrix

Contents

List of Figures

List of Tables

1
Introduction

MIMO radar systems offer many advantages in the area of signal design and processing. A commonly studied area in this regard involves the transmission of orthogonal waveforms for the different antenna elements, which allows for a clean separation of the different transmitted signals on the receiving end [1]. However, there are other ways to leverage the degrees of freedom in a MIMO radar that does not always require orthogonality. Particularly, MIMO radar systems also allow for flexible waveform design tailored to suppress clutter [2], i.e. unwanted signal dependent returns from the environment and other reflective sources [3]. In this regard, various research focuses on waveform design and additional radar design parameters developed to suppress clutter, often using optimization-based approaches to specific design problems.

Furthermore, radar systems often have practical constraints based on hardware platforms with limited computational resources, while adaptivity to rapidly changing environments is of interest. Computations for waveform design can be demanding off-line, but on-line in the radar system there is demand for close to real-time execution and often with restricted hardware. Therefore, waveform design algorithms often focus on low computational complexity and convergence speed [4]. Artificial neural networks (ANNs) can allow for low computational complexity at test time, with adaptive properties, by leveraging heavy computations and large datasets during training. Additionally, artificial neural networks can efficiently utilize GPU hardware and extensive machine learning libraries, potentially giving an advantage on future radar platforms with integrated GPUs. This makes neural network-based approaches interesting for the case of adaptive waveform design in clutter environments.

1.1 Background

Multiple-input multiple-output (MIMO) radar exploits the spatial degrees of freedom offered by an array of transmit and receive antenna elements to improve resolution, target detection and parameter estimation, and flexibility of transmit beampattern designs [5]. Unlike standard phased-array antennas that transmit scaled versions of the same waveform on each antenna element, MIMO radars can transmit different waveforms on each antenna element, giving a much larger degree of freedom in the waveform design problem [5]. The use of these design freedoms to optimize

transmit-receive patterns have been subject to considerable research, including research on interference- and clutter-suppression [1].

In related works, the joint design of transmit waveforms and receive filters for improved detection of multiple targets in the presence of signal dependent clutter and independent noise has been studied using iterative optimization algorithms in [6]. In [7], the joint design of transmit waveform, receiving filters and antenna configurations is studied, with the aim of enhanced target detection in clutter environments. Although a large focus on research in waveform design is on the computational complexity of implemented algorithms, as well as solving the non-convexity of certain design objectives, the utilization of deep learning to solve these problems is less explored. Deep learning can theoretically map to non-convex optimization functions by the artificial neural networks forming non-linear systems, as noted in [8] where deep learning is applied to a waveform design objective of high SINR and low integrated sidelobe levels (ISL). The focus of [8] is however in regards to training a neural network on a single scenario, where the network learns to generate one optimal waveform for a fixed set of design objectives. While it shows promising results for waveform design in a non-convex optimization landscape, it does not address the training of neural networks for real-time generation of optimal waveforms for changing environments. Further, generative adversarial networks (GANs) have been studied in [9] with the purpose of waveform synthesis with desired orthogonality properties. In current research however there seems to be a gap in regards to research into neural network-based approaches to waveform synthesis with the purpose of suppressing range-angle distributed clutter, especially with the purpose of generalized neural networks able to handle arbitrary clutter scenarios at test-time.

Since waveform design objectives often are formalized with non-convex optimization functions, there can in some cases exist an arbitrary amount of separate solutions with satisfactory performance. It can therefore be of interest to generate a diverse set of solutions for the same scenario, which in turn could produce more robust performance by evaluating multiple solutions and using the best one. A further practical reason for encouraging sets of diverse solutions is the increased robustness against adversarial electronic-counter-measure (ECM) techniques, specifically using digital radio frequency memory (DRFM) repeaters which can replicate an intercepted signal and retransmit it to produce false targets. Additionally, a diversity in transmitted waveforms can mitigate an adversary's attempts to identify intercepted signals and categorize the transmitting systems characteristics [10], also known as electronic support measures (ESM). A commonly researched approach for combating this relates to pulse diversity [11], which in the context of this thesis relates to obtaining multiple waveform solutions for the same clutter environment scenario.

The background outlined thus far motivates the research of using deep learning and neural networks for the purpose of generating waveforms for arbitrary clutter environments, and additionally the research on the possibility of generating multiple solutions for any given clutter environment.

1.2 Problem formulation

This section presents a problem formulation that is based on the waveform design problem at hand. Here, the problem is introduced and motivated, briefly covering the necessary signal modeling and processing. For a more detailed and thorough view of the underlying theory, see Section 2.1.

We investigate transmit waveform design for MIMO radar systems by shaping the ambiguity function (AF), which describes the radar systems response for a given transmit waveform and antenna configuration after matched filtering of signal returns, to enhance target detection performance in clutter environments. The positions of clutter regions and targets, as well as the ambiguity function, are defined in a 2D range-angle map. Given an arbitrary clutter environment, or clutter map (CM), the primary objective is to synthesize a waveform that maximizes signal-to-clutter-noise ratio (SCNR), i.e. minimize received signals from clutter regions while maximizing received signal from the target, with a secondary objective of maintaining the bandwidth of the transmitted waveform within a desired region. The SCNR metric is closely related to the actual detection performance in clutter environments [3], and can in this case be motivated as a suitable proxy for waveform performance. The waveforms used here are constrained to unit modulus, meaning all transmitted waveforms have amplitude 1, which is a common design constraint in related works such as [12], [6] and [7].

Throughout this thesis, it should be noted that the SCNR and receive processing is mainly formulated with a single receiving channel, instead of combining multiple receive channels. This can also be related to the special case of having a multiple-input single-output (MISO) radar system, with multiple transmit antennas and a single receive antenna. Consequently, the waveform design objective does not include the direction-dependent gain of a receive array. This isolates the waveform design objective to being centered around the pure waveform properties, and the receive-array beamforming is considered out of scope for this thesis.

We consider a coherent, monostatic radar with M antenna elements, each transmitting a unimodular phase-coded pulse of N_b fast-time sample points, i.e. a phase matrix $\mathbf{\Phi} \in \mathbb{R}^{M \times N_b}$ defining a waveform $\mathbf{S} = e^{j\mathbf{\Phi}} \in \mathbb{C}^{M \times N_b}$. The phase matrix $\mathbf{\Phi}$ constitutes the parameters to be optimized, or in this case generated as output from a neural network, to obtain a high SCNR. The ambiguity function is an important tool for characterizing the response of a radar system, and many radar design problems can be formulated in terms of obtaining a desired ambiguity function adapted to various scenarios and design objectives [13]. In particular, for the problem of designing waveforms for clutter suppression, the ambiguity function provides a unified perspective where the design objective can be formulated as shaping the ambiguity function to adapt to the clutter environment.

For a given transmit waveform $\mathbf{S} = e^{j\mathbf{\Phi}}$, the transmitted signal in a direction θ_0 is given by

$$\mathbf{s_E}(\theta_0) = \mathbf{a}(\theta_0)^H \mathbf{S}, \qquad \mathbf{s_E}(\theta) \in \mathbb{C}^{1\times N_b} \tag{1.1}$$

where $\mathbf{a}(\theta_0)$ denotes the steering vector, given by

$$\mathbf{a}(\theta) = [1, e^{j\pi \sin(\theta)}, ..., e^{j\pi m \sin(\theta)}, ..., e^{j\pi M \sin(\theta)}]^T \tag{1.2}$$

assuming a half-wavelength spacing between antenna elements. The symbol $(\cdot)^H$ denotes the Hermitian transpose. Under the considered single receive-channel model, the corresponding ambiguity function is given by

$$AF(\tau, \theta, \theta_0) = \mathbf{a}^H(\theta)\mathbf{S}_\tau \mathbf{S}^H \mathbf{a}(\theta_0) \tag{1.3}$$

where $\mathbf{S}_\tau$ is a time-shifted version of the transmit waveform matrix $\mathbf{S}$, representing the received signal. More specifically, we have

$$(S_\tau)_{m,n} = \begin{cases} S_{m,n}, & \text{if } \ 0 \le n + \tau \le N_b - 1, \\ 0, & \text{otherwise,} \end{cases} \tag{1.4}$$

i.e. $\mathbf{S}_\tau$ is zero-padded as to account for a sampling of the returned signal that begins before the arrival of the target echo. For a target located in θ_0, we can then define the matrix $\mathbf{AF} \in \mathbb{C}^{N_u \times N_r}$ as a matrix of samples of the ambiguity function, where N_u denotes the angle bins and N_r denotes the range bins. We set $N_r = 2N_b - 1$ to cover the whole range-delay interval that can be resolved after matched filtering of a signal with N_b fast-time sample points. Since the ambiguity function defines the signal returns after matched filtering, the exact target response is given by $AF(\tau = 0, \theta = \theta_0, \theta_0)$ when, without loss of generality, the target is assumed located at delay $\tau = 0$. The target return power is then given by

$$G(\theta_0, \mathbf{\Phi}) = |AF(\tau = 0, \theta = \theta_0, \theta_0)|^2 = |\mathbf{a}^H(\theta_0)\mathbf{S}\mathbf{S}^H\mathbf{a}(\theta_0)|^2. \tag{1.5}$$

Now, to model clutter environment, we define a clutter map with the same dimensions $\mathbf{CM} \in \mathbb{R}^{N_u \times N_r}$. The entries of $\mathbf{CM}$ encode the strength of the different components that appear in the range-angle plane:

- *Clutter regions*: large positive values $CM_{i,j} \gg 1$ indicating strong unwanted reflections.
- *Noise floor*: low values close to zero $CM_{i,j} \sim 0.1$ modeling a constant background noise.
- *Target region*: A value of zero $CM_{i,j} = 0$ in the exact target position and a small neighborhood of surrounding bins.

With this, the objective of clutter suppression can be formulated as minimizing the weighted integrated sidelobe level (WISL), where the clutter map is set as the weights. This can be written as the expression

$$\text{WISL}(\mathbf{\Phi}, \mathbf{CM}) = \langle \mathbf{CM}, |\mathbf{AF}|^2 \rangle_F = \sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j} \, |AF_{i,j}|^2 \, . \tag{1.6}$$

where we take the Frobenius inner product, denoted $\langle \cdot, \cdot \rangle_F$, between the power of the ambiguity function and the clutter map. The WISL represents the power of all signal returns excluding the target, and the interpretation of the CM as the reflective power gain of spatially distributed clutter holds consistent. Together with the target power in Eq. (1.5), the SCNR is given by

$$\text{SCNR} = \frac{|\mathbf{a}(\theta_\mathbf{0})^H \mathbf{S}\mathbf{S}^H \mathbf{a}(\theta_\mathbf{0})|^2}{\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j} \, |AF_{i,j}|^2}. \tag{1.7}$$

To incorporate potential hardware constraints in a real-world radar system, and limit spectral leakage, a soft constraint on the instantaneous bandwidth (or phase-increments) is implemented. For each antenna element m and every pair of adjacent fast-time samples n and $n+1$ we define a wrapped phase increment

$$\Delta\Phi_{m,n} = \text{wrap}\Big(\Phi_{m,n+1} - \Phi_{m,n}\Big) \in [-\pi, \pi), \tag{1.8}$$

where $\text{wrap}(\cdot)$ adds or subtracts 2π so that the result lies in the principal interval. A tolerance $D_{max} > 0$ is set, and the deviation from the tolerance is measured by

$$D_{m,n} = \Big[|\Delta\Phi_{m,n}| - D_{max} \Big]_+, \tag{1.9}$$

where we have $[\,\cdot\,]_+ = \max\{0, x\}$. Using this measure of deviation, we can formulate the bandwidth design objective, or penalty, as

$$J_{\text{bw}}(\mathbf{\Phi}) = \frac{1}{M(N_b - 1)} \sum_{m=1}^{M} \sum_{n=1}^{N_b - 1} \Big[1 + D_{m,n} \Big]^p, \qquad J_{\text{bw}}(\mathbf{\Phi}) \geq 1 \tag{1.10}$$

with the exponent p as a design parameter determining the severity of J_{bw}, in this case set to $p = 8$. This bandwidth-regularizing term is combined with the SCNR metric, to obtain a waveform design objective F:

$$F = \frac{|\mathbf{a}(\theta_\mathbf{0})^H \mathbf{S}\mathbf{S}^H \mathbf{a}(\theta_\mathbf{0})|^2}{\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j} \, |AF_{i,j}|^2} \frac{1}{J_{\text{bw}}(\mathbf{\Phi})}. \tag{1.11}$$

The F-value is used as the primary performance metric used to evaluate waveforms throughout the thesis, with a main design objective centered around maximizing the F-value.

In addition to the waveform design problem for a single given clutter environment, this thesis is focused on the development of neural networks that are trained on datasets of multiple different clutter environments in order to generalize and, at test time, generate suitable waveforms for arbitrary clutter environments.

In regards to the training of neural networks for waveform design, part of this thesis focuses on developing neural networks that can generate more than one suitable waveform matrix $\mathbf{\Phi}$ for the same clutter map $\mathbf{CM}$, as opposed to only generating a single waveform per clutter map. For this, additional metrics for evaluation are formulated and the loss function is adjusted to suit this aim, presented further in Section 2.1 and Section 3.2.1.

1.3 Aim

The aim of this Master's thesis is to research the feasibility of using neural network-based approaches developed to generate suitable waveforms for arbitrary clutter environments, where the designed waveforms suppress the returns from clutter regions while maintaining sufficient target illumination thereby obtaining a good signal-to-clutter-noise ratio (SCNR). By training neural networks on datasets of varying clutter scenarios, the aim is to achieve good performance on previously unseen clutter maps at test-time, as opposed to only optimizing a single waveform for a single scenario.

To achieve this, several different neural network architectures are explored, as well as different dimensions and characteristics of clutter scenarios to investigate the feasibility of a neural network-based approach in different settings of degrees of freedom (DoF) and difficulty of clutter environment. Further, part of this thesis explores the additional aim of achieving diverse sets of solutions for the same clutter environment, given that the design objective is non-convex and thus can contain a multitude of feasible solutions and can also provide increased robustness against adversarial electronic-counter-measure techniques.

1.4 Delimitations

In order to maintain a feasible and focused scope, certain delimitations have been set. The clutter maps are defined as two-dimensional range-angle maps, omitting any 3D volumetric clutter and any Doppler processing or processing between different waveform pulses. The problem is formulated based on a monostatic, coherent radar system where only a single colocated transmit/receive antenna array is considered. Extending to bistatic or multistatic geometries would introduce additional phase-modeling complexity. The antenna configuration is based on half-wavelength spacing between antenna elements, otherwise known as a uniform linear array (ULA). We also implement a unit modulus constrain on the transmit waveforms, meaning only the phase $\mathbf{\Phi} \in \mathbb{R}^{M \times N_b}$ is a freely set design parameter in the transmit waveform $\mathbf{S} = e^{j\mathbf{\Phi}}$. The design objective is limited to varying only the transmit waveform and evaluating the matched-filter output in a single receive channel, before any multi-channel receive processing. Receive beamforming and joint optimization of the transmit waveform and a linear receive combiner are thereby outside the scope of this thesis.

In addition to the delimitations on antenna configuration, this thesis is limited to a single target set to be located in $\tau = 0,\ \theta_0 = 0$. This limitation is set since the relevant information is the relative positions between clutter and target, and a shift of target position in terms of τ is equivalent to keeping the target fixed and implementing the same translation on all clutter regions instead. A shift of the targets angular direction θ_0 is instead equivalent to an applied steering vector. On

the subject of multiple targets, it would be a natural extension to this thesis. The clutter maps used for training and evaluation are all synthetically generated, allowing for large and controllable datasets without the need for field-measured data collection. The clutter maps are also given as input to the neural networks without any perturbations corresponding to measurement inaccuracies, and noise is modeled by a constant, low level noise floor. This isolates the performance of the neural network to be evaluated based on how accurate it is in regards to the given clutter environment, although in real applications measurement inaccuracies can affect the overall performance.

The problem formulation, with the design of waveforms aimed to suit given clutter environments and enhance target detection performance, can in and of itself be said to be a delimitation. This focus lies in a specific radar scenario, where a priori information about a potential target and the clutter environment is used as an advantage to adaptively design suitable waveforms. This is often referred to as knowledge-aided, or cognitive radar, and has been an increasing area of interest as hardware improvements and radar technology in general has advanced [14]. This constitutes a specific mode of radar operation, and other modes such as a "search mode" where the radar scans its entire search volume in search of targets, is not considered. It would however be a natural extension for further research.

Together, these delimitations ensure that the thesis remains tractable and focused on the core research question on wether neural network-based waveform generation can reliably produce high-performance, diverse solutions for arbitrary clutter environments within the defined modeling. Among these restrictions there are several potential avenues for future extensions and further research, such as multi-target environments and extending to Doppler-processing for potential high velocity targets.

2
Theory

This chapter provides the theoretical foundations on MIMO radar systems and signal modeling as a background to the research context of this thesis. It also presents the relevant machine learning theory and concept behind the applied methods.

2.1 Radar and Signal Model

This thesis focuses on waveform code design for coherent MIMO radar, and as such this section presents the necessary radar and signal model relevant to this area of research. Active sensing systems such as radars have the goal of determining useful properties of the surrounding environment and potential targets. For example, a radar can determine the range of a target by measuring the round-trip time delay between transmitting a signal and receiving an echo, and the speed of said target can be estimated by measuring the Doppler frequency shift when processing the received signal [15]. The operation of a radar system can be described briefly as: the radar transmits electromagnetic energy from an antenna, some of which is re-radiated by reflecting objects in many directions with parts of the re-radiated energy directed back at and returned to the radar antenna, which can then amplify and process the returned signal [2].

2.1.1 Radar Introduction

In Figure 2.1, a block diagram of a basic, elementary radar system can be seen, showing an example of how a radar and its subsystems can be organized. A transmitter consisting of a waveform generator and a power amplifier generates a waveform that is transmitted by the antenna. The power of transmitted signal can vary greatly between different radar systems and different application, with average power being an important factor in the capability and performance of a radar. This configuration has a shared antenna between transmitter and receiver, which is enabled by using short pulse waveforms, something that is commonly used in radars [2]. The duplexer shifts between the receiver and the transmitter, and protects the sensitive receiving side from the high energy of the transmitter. The receiver has an amplifier to strengthen the weak signal returns, and processes the signal to enable detection and extract information about the target. The signal processing includes filtering, commonly using a matched filter (i.e. the filter is a replica of the transmit waveform), to maximize signal-to-interference-noise ratio (SINR) to improve the ability

of separating target echoes from noise and other unwanted echoes, known as clutter [2].

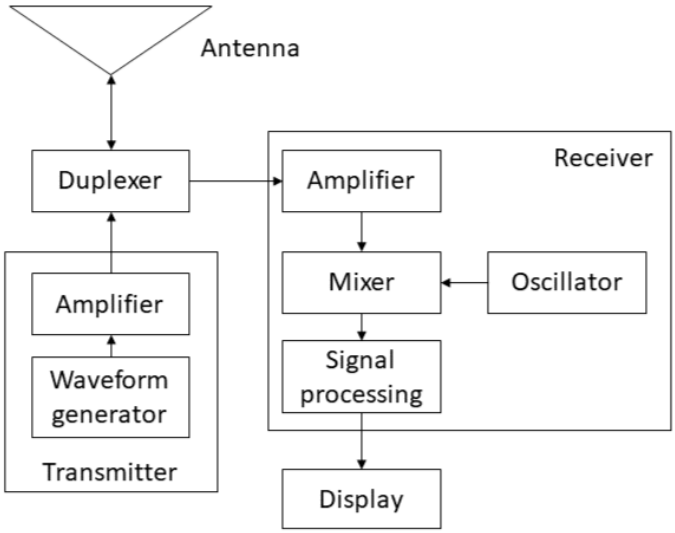


Figure 2.1: Block diagram of a simple radar system with transmitter and receiver sharing a single antenna via a duplexer.

There are many different types of radars that are used for a wide array of applications, ranging from weather monitoring and automotive collision controls to military surveillance. One fundamental distinction is between monostatic radars, where the transmitter and receiver are colocated, and bistatic or multistatic radars where the spacing between transmit and receive arrays is very large in comparison to the signal wavelength [16]. Another important distinction is mechanically scanned radars, which use the physical rotation of antennas to steer transmit beams, and electronically scanned radars which steer beams electronically using phase-shifts between different antenna elements [2].

Traditional radars typically use a single waveform, for example a baseband modulated carrier wave transmitted by a single antenna element. This results in reflected signals being scaled versions of the same waveform, with the scaling depending on the radiation pattern of the antenna. However, with active electronically scanned array (AESA) architectures and digital radar arrays, the radar hardware allows for much more flexibility in terms of the generated waveforms [17]. Multi-input multi-output (MIMO) radar is a technique that uses space-time diversity in waveforms, typically involving independent waveforms on each antenna elements, leading to transmit waveform that can vary greatly between different directions not only by scaling but varying by the waveform envelope itself. This is due to the fact that the far-field waveform is the coherent sum of all the input waveforms [18].

Similar to other radar systems, MIMO radars can be divided into the categories of multistatic MIMO, also known as statistical MIMO, with wide separation of transmit and receive arrays, and coherent MIMO which has colocated arrays. An overview of the MIMO concept is shown in Figure 2.2, where the transmit antenna elements transmit independent waveforms $s_m(t)$, with the returning echoes being combinations of all signals $\{s_m(t)\}$. The coherent MIMO system can share the same antenna arrays between transmitters and receivers, or separate colocated arrays. On the receiving side of the MIMO radar, each receiver can separate the radar returns from each individual transmit element (if the waveforms are separable), allowing for offline scanning of transmit patterns after receival unlike traditional single aperture radars [18].

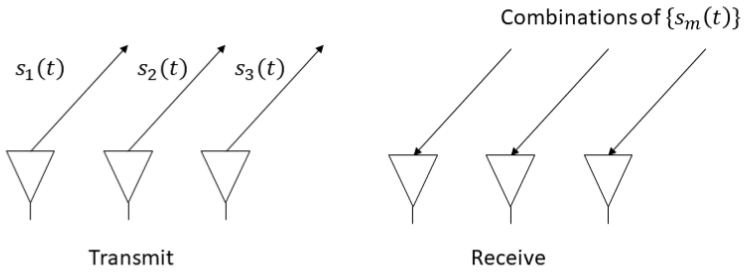


Figure 2.2: A basic overview of a coherent MIMO radar system, with transmit and receive antenna arrays.

In the case of a traditional phased-array radar, the transmit signals $s_m(t)$ would be scaled versions of the same waveform $s_m(t) = w_m s(t)$, limiting the flexibility in transmit beampattern design [19].

When it comes to waveform design, there are several different approaches. The types of signal used by a radar can be divided into two main categories: continuous wave (CW) where the radar continuously transmits a signal, and pulsed signals where the radar transmits signals in sequences of pulses [20]. Both categories can be further divided into different types of waveform design approaches. This thesis focuses on waveform design of pulsed transmission signals, with a waveform design in the code domain, specifically phase-coded waveform design.

In a pulsed radar the transmitted burst is repeated with some pulse-repetition interval (PRI), which creates two distinct time scales. The time between different pulses is often referred to as slow-time, with some of the typical slow time operations being integrating several pulses for enhanced signal strength and Doppler processing for velocity estimation among others [21]. The time within a transmitted pulse is commonly referred to as fast-time, where typical operations are signal formation such as the phase-coding considered here, matched filtering and beamforming among others [21].

2.1.2 Signal model

We consider a coherent, monostatic MIMO radar system with M transmit antenna elements, each transmitting a unimodular phase-code pulse of N_b fast-time sample points. Let $s_m(n)$ denote the discrete-time signal transmitted by antenna element m, and θ_0, τ_0 denote the azimuth angle and range-delay from the antenna array to a generic target. Under the assumption that the signals are narrowband, i.e. they occupy a narrow frequency spectrum, and that the signal propagation is non-dispersive, the far-field of the signal at the target location can be described by [19]:

$$s_E(\theta_0, \tau_0, n) = \sum_{m=1}^{M} e^{-j2\pi f_0 \tau_m^{(\text{prop})}(\theta_0,\tau_0)} s_m(n) = \mathbf{a}^H(\theta_0, \tau_0)\mathbf{s}(n) \tag{2.1}$$

where f_0 denotes the carrier frequency of the signals, $\tau_m^{(\text{prop})}(\theta_0, \tau_0)$ denotes the propagation time delay for the signal sent from antenna element m to reach the target, and $(\cdot)^H$ is the Hermitian transpose (or conjugate transpose). Taken over all fast-time samples, the far-field signal can be written as

$$\mathbf{s}_E(\theta_0, \tau_0) = \mathbf{a}^H(\theta_0, \tau_0)\mathbf{S} \tag{2.2}$$

$$\mathbf{S} = [\mathbf{s}(0), \mathbf{s}(1), \ldots, \mathbf{s}(N_b - 1)] \tag{2.3}$$

where $\mathbf{S}$, given the unimodular constrain, can be described as $\mathbf{S} = e^{j\mathbf{\Phi}} \in \mathbb{C}^{M\times N_b}$ where the matrix $\mathbf{\Phi}$ describes the phase-coding of the pulses sent by the transmit antennas. The steering vector $\mathbf{a}(\theta_0, \tau_0)$ is given by

$$\mathbf{a}(\theta_0, \tau_0) = [e^{j2\pi f_0, \tau_1^{(\text{prop})}(\theta_0,\tau_0)}, e^{j2\pi f_0 \tau_2^{(\text{prop})}(\theta_0,\tau_0)}, \ldots, e^{j2\pi f_0 \tau_M^{(\text{prop})}(\theta_0,\tau_0)}]^T \tag{2.4}$$

with $(\cdot)^T$ denoting the transpose. Without loss of generality, we use a fixed target location of $\tau_0 = 0$, using the notation $a(\theta, \tau = 0) = a(\theta)$. Given this, we have the time-delay

$$\tau_m^{(\text{prop})}(\theta) = \frac{d\, m \sin(\theta)}{c} \tag{2.5}$$

where d is the spacing between antenna elements and c is the propagation velocity of the electromagnetic wave. Angles $\theta \in [-\pi/2, \pi/2]$ are taken from the normal vector of the antenna array, as illustrated in Figure 2.3.
Assuming half wavelength spacing $d = \lambda/2$ with lambda denoting wavelength, and using the fact that the carrier frequency is given by $f_0 = c/\lambda$, we have:

$$\begin{aligned} a_m(\theta) &= \exp(j2\pi f_0 \tau_m^{(\text{prop})}(\theta)) = \\ &= \exp(j2\pi \frac{c}{\lambda} \frac{d\, m \sin(\theta)}{c}) \\ &= \exp(j2\pi \frac{c}{\lambda} \frac{\lambda}{2} \frac{m \sin(\theta)}{c}) \\ &= \exp(j\pi m \sin(\theta)) \end{aligned} \tag{2.6}$$

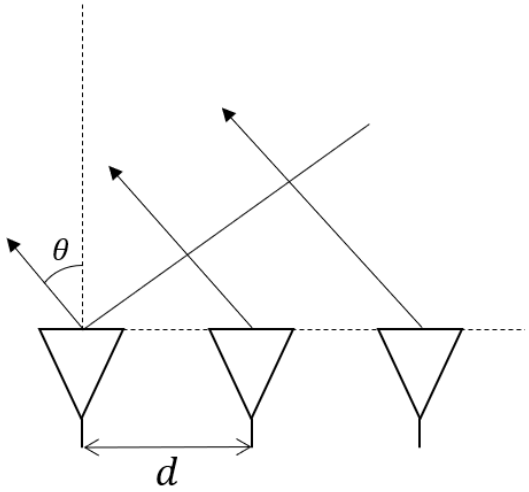


Figure 2.3: Overview of an antenna array showing the angle θ defined from the arrays normal vector, with inter-element spacing d.

where $a_m(\theta)$ denotes element m of the steering vector. The expression of the steering vector in Eq. (2.4) can now be written as

$$\mathbf{a}(\theta) = [1, e^{j\pi \sin(\theta)}, e^{j2\pi \sin(\theta)}, \ldots, e^{j(M-1)\pi \sin(\theta)}]^T. \tag{2.7}$$

As per [19], under the simplified model of far-field point reflectors, the received signal can be described by the equation

$$\mathbf{X} = \sum_{k=1}^{K} \beta_k \overline{\mathbf{b}(\theta_k)} \mathbf{a}^H(\theta_k)\mathbf{S} + \epsilon \tag{2.8}$$

where K denotes the number of signal reflecting targets, $\{\beta_k\}$ are amplitudes proportional to the reflectors radar cross sections (RCSs), θ_k are their angular locations, ϵ covers noise plus interference and $\overline{(\cdot)}$ is the complex conjugate. Here, $\mathbf{b}(\theta_k)$ denotes the receiving ends steering vector, which is defined in the same way as the transmit steering vector $\mathbf{a}(\theta)$ in Eq. (2.7).

2.1.2.1 Clutter model

In radar and signal processing, clutter refers to unwanted signal returns which can come from different scatterers such as the earth's surface, man-made structures such as buildings, rain clouds and other weather echoes [21]. In our case, the focus is spatially distributed clutter, defined in a range-angle map referred to as a clutter map (CM).

The clutter map is a matrix $\mathbf{CM} \in \mathbb{R}^{N_u \times N_r}$ where N_u is the number of discrete angle-bins and N_r is the number of discrete range-bins considered in the processing

of received signals. In other words, the elements of the clutter map $\mathbf{CM}$ are the real-valued power scattering amplitudes of unwanted signal-dependent reflections, which can be related to the square of the target scattering coefficient β in Eq. (2.8). Large positive values in the clutter map correspond to strong unwanted reflections, while the target range-angle cell is set to zero. As previously presented in the problem formulation in Section 1.2, the clutter map encodes three distinct components:

- *Clutter regions*: large positive values $\mathbf{CM}_{i,j} \gg 1$ indicating strong unwanted reflections.
- *Noise floor*: low values close to zero $\mathbf{CM}_{i,j} \sim 0.1$ modeling a constant background noise.
- *Target region*: A value of zero $\mathbf{CM}_{i,j} = 0$ in the exact target position and a small neighborhood of surrounding bins.

This view leads directly to the use of weighted-integrated-sidelobe level as a metric defining the effects of signal dependent clutter and noise, introduced in Section 2.1.3.3. This modeling of clutter as point-like scatterers has been implemented with similar approaches in [7] which also uses range-angle distributed clutter, as well as in [14] where clutter and signals are considered in the range-Doppler domain instead. A more in-depth practical description of how the clutter maps are constructed can be seen in Section 3.1.1, describing method for synthetically generating clutter distributions.

2.1.3 Signal processing

This subsection presents the signal processing model and concepts that are used throughout this thesis. It is important to note that there are many different approaches to signal processing, with different suitable use cases and assumptions. The signal model and processing presented and used here is focused on fast-time processing of coherent MIMO radar waveforms with unimodular constraints. Another delimitation is that it excludes any Doppler processing, which is applicable under the assumption of negligible Doppler effects within the duration of the signal pulses and processing interval [22].

2.1.3.1 Transmit beampattern

For a colocated MIMO radar system, the transmitted waveform $\mathbf{S} \in \mathbb{C}^{M \times N_b}$ determines the spatial distribution of signal energy across different angular directions as, previously described in Section 2.1.2. The power of the transmitted signal in a generic direction θ is given by the transmit beampattern [5]:

$$P(\theta) = ||\mathbf{a}^H(\theta)\mathbf{S}||_2^2 = \mathbf{a}^H(\theta)\mathbf{S}\mathbf{S}^H\mathbf{a}(\theta). \tag{2.9}$$

The flexibility of probing signal design in MIMO radar has given rise to a wide array of research into design of transmit beampattern with different design objectives [23]. The transmit beampattern can be used to evaluate how well a designed waveform

focuses energy on target location, and is commonly used together with sidelobe metrics for waveform design objectives [24, 23].

2.1.3.2 Matched Filter and Ambiguity Function

A commonly used filter for processing received signals is the matched filter, which in the case of stochastic additive white noise maximizes the signal-to-noise ratio (SNR) [15]. A matched filter is defined as having a transfer function that, except for a possible amplitude and time-delay factor, is the complex conjugate spectrum of the signal to which it is matched [25]. Now, if we consider a target located in $(\theta_0, \tau_0 = 0)$, we construct a reference signal based on the expected returns from the target:

$$\mathbf{S}_R(\theta_0) = \overline{\mathbf{b}(\theta_0)}\mathbf{a}^H(\theta_0)\mathbf{S}, \tag{2.10}$$

that is used for matched filtering of received signals. Given an incoming reflected signal from angle θ with delay τ, the response of the matched filter is given by

$$\mathbf{Y}(\theta, \tau, \theta_0) = \mathbf{S}_R(\theta, \tau)\mathbf{S}_R^H(\theta_0) = \overline{\mathbf{b}(\theta)}\mathbf{a}^H(\theta)\mathbf{S}_\tau\mathbf{S}^H\mathbf{a}(\theta_0)\mathbf{b}^T(\theta_0) \tag{2.11}$$

where $\mathbf{S}_\tau$ denotes a time-shifted, zero-appended version of the waveform matrix $\mathbf{S}$ that incorporates the delay τ, more specifically defined as

$$(\mathbf{S}_\tau)_{m,n} = \begin{cases} S_{m,n}, & \text{if } \ 0 \leq n + \tau \leq N_b - 1, \\ 0, & \text{otherwise.} \end{cases} \tag{2.12}$$

The matched filter response $\mathbf{Y}(\theta, \tau, \theta_0)$ is matrix-valued, where the (p,q)-th element corresponds to the matched-filter response between the p-th receive channel for an incoming signal from direction θ and the q-th receive channel associated with the reference direction θ_0. Thus, the matrix contains all pairwise receive-channel responses. In Eq. (2.11), we can note that the term $\mathbf{a}^H(\theta)\mathbf{S}_\tau\mathbf{S}^H\mathbf{a}(\theta_0)$ collapses to a scalar, as $\mathbf{a}(\theta) \in \mathbb{C}^{M\times 1}$ and $\mathbf{S} \in \mathbb{C}^{M\times N_b}$. We can rewrite it as

$$\mathbf{Y}(\theta, \tau, \theta_0) = \left(\overline{\mathbf{b}(\theta)}\mathbf{b}^T(\theta_0)\right)\left(\mathbf{a}^H(\theta)\mathbf{S}_\tau\mathbf{S}^H\mathbf{a}(\theta_0)\right). \tag{2.13}$$

where it becomes clear that the receive-array factor is decoupled from the waveform-dependent factor, in the sense that it's not dependent on the transmit waveform, as is also shown in [22]. This decoupling allows the waveform-dependent factor to be considered independently of the receive-array factor. Since this thesis is focused on transmit waveform design, we restrict the considered formulation to the case of a single receive channel, where the receive-array steering vector reduces to unity, i.e. $\mathbf{b}(\theta) = [1]$. Under this restriction of a single receive-channel, the waveform design objective can thereby be isolated to only the waveform-dependent factor. In a multi-channel receive array, however, the receive-array factor is directionally dependent and would generally affect the target and clutter returns.

The response of a matched filter is also commonly referred to as the ambiguity function (AF), and generally has an important role in radar systems, as many radar design problems can be formulated as objectives to achieve desired properties of the

ambiguity function to adapt to various applications and environments [13]. The ambiguity function is often presented as the range-Doppler response of a matched filter, as in [13], but it has been shown that the MIMO ambiguity function presents a range-angle coupling except for perfectly orthogonal waveforms, which are unrealistic in practice [22]. In this case we instead study the range-angle ambiguity function, as the waveform design problem relates to suppressing spatially distributed clutter for heightened detection performance, with the assumption of negligible Doppler effects within the pulse duration and processing interval.

Under the previously introduced restriction of a single receive-channel, we define the discrete range-angle ambiguity function

$$AF(u, \tau, u_0) = \mathbf{a}^H(u)\mathbf{S}_\tau\mathbf{S}^H\mathbf{a}(u_0) \tag{2.14}$$

where we use the notation $u = \sin(\theta)$, similarly to [22], where the range-angle ambiguity function is described in similar fashion. The corresponding matched-filter output energy is given by

$$G(u, \tau, u_0) = |AF(u, \tau, u_0)|^2. \tag{2.15}$$

For a grid of N_u angle bins and $N_r = 2N_b - 1$ range-delay bins, the ambiguity function in Eq. (2.14) can be represented in discrete form as a matrix $\mathbf{AF} \in \mathbb{C}^{N_u \times N_\tau}$. This matrix describes the response of a matched filter for each range-angle cell in the spatial area covered by the signal processing. The number of angle bins N_u can be set freely and simply determines the angular resolution of the ambiguity function (and consequently the clutter map, as they need matching dimensions). The number of range bins is set to a fixed relation $N_r = 2N_b - 1$ as to cover the whole range-delay interval that can be resolved by the matched filter processing, also referred to as the sampling interval. The discrete ambiguity function can with benefit be calculated efficiently using Fast Fourier Transforms (FFTs), since it is in practice a convolutional operation between signals in the fast-time dimension.

2.1.3.3 Weighted Integrated Sidelobe Level

The weighted integrated sidelobe level (WISL) is a metric that can be used in order to shape the ambiguity function, both in the case of the range-Doppler domain as in [12] and for the case of range-angle domain as in [7]. As the name suggests, it is weighted version of the regular integrated sidelobe level (ISL), commonly used in different optimization algorithms to achieve good autocorrelation or cross-correlation properties [26] [27]. It can also be related the concept of cognitive radar, where prior information provided by the cognitive radar is used to define the weights in the WISL - which in turn can be used to design a suitable waveform [12].

In our case, we formulate a WISL in the range-angle domain, and use the previousy mentioned clutter map as the weighting factors. Given that the clutter map is used to define the reflective power of clutter, this results in a WISL that corresponds to the integrated energy received from all clutter echoes after transmission and matched

filtering of a given waveform. This is later used to define the signal-to-clutter-noise ratio (SCNR), described in Section 2.1.3.4. The WISL is defined as

$$\text{WISL} = \langle \mathbf{CM}, |\mathbf{AF}|^2 \rangle_F = \sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j} \, |AF_{i,j}|^2 \tag{2.16}$$

where **CM** is the clutter map and **AF** is the discrete ambiguity function, and $\langle \cdot, \cdot \rangle_F$ denotes Frobenius inner product. The elements of the clutter map matrix **CM** are used as weighting parameters for each range-angle cell in the ambiguity function, with large positive values corresponding to strong unwanted reflections. A small value of the WISL indicates that the waveform produces low energy returns from clutter-heavy areas, while the target area is set to $\mathbf{CM}_{\text{target}} = 0$ as to not penalize target illumination. It is possible integrate the waveform design objective of signal strength on target into the WISL metric by having negative weighting at the target location $\mathbf{CM}_{\text{target}}$, but it is not necessary and as mentioned in [7] it can be very sensitive to the choice of weighting coefficient.

2.1.3.4 SCNR

The signal-to-clutter-noise ratio (SCNR) is a commonly used waveform design metric in MIMO radar, as it is closely related to the detection performance when dealing with clutter environments [28] [3]. Here, it is used as a proxy for the detection performance, and thus used to measure waveform performances for given clutter environments.

If we consider the previously described received signal in Eq. (2.8), using a single target with reflective power $\beta = 1$ and setting the receive steering-vector $\mathbf{b}(\theta) = [1]$, we can rewrite it as

$$\mathbf{x} = \mathbf{a}^H(\theta_0)\mathbf{S} + \epsilon, \tag{2.17}$$

where $\mathbf{x}$ encompasses both the signal return from the target and from clutter. Again, setting $\mathbf{b}(\theta) = [1]$ corresponds to the single receive-channel restriction introduced in Section 2.1.3.2. Thus, the SCNR considered below is evaluated in a single receive channel, prior to any linear combination or beamforming across multiple receive channels. The noise plus interference term ϵ in our case is the signal dependent clutter and noise modeled via the clutter map. Inserting this into Eq. (2.17) we can write

$$\mathbf{x} = \mathbf{a}^H(u_0)\mathbf{S} + \sum_{i=1}^{N_u} \sum_{j=1}^{N_r} \sqrt{CM_{i,j}} \, \mathbf{a}^H(u_i)\mathbf{S}_{\tau_j} \tag{2.18}$$

where we use the notation $u = \sin(\theta)$, taking $N_u \times N_r$ discrete signal returns from the spatially distributed clutter environment around the target. Matched filter upon receive gives an output

$$\mathbf{y} = \left(\mathbf{a}^H(u_0)\mathbf{S} + \sum_{i=1}^{N_u} \sum_{j=1}^{N_r} \sqrt{CM_{i,j}}\, \mathbf{a}^H(u_i)\mathbf{S}_{\tau_j} \right) \mathbf{S}^H \mathbf{a}(u_0) \tag{2.19}$$

$$\mathbf{y} = \left(\mathbf{a}^H(u_0)\mathbf{S}\mathbf{S}^H\mathbf{a}(u_0) \right) + \left(\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} \sqrt{CM_{i,j}}\, \mathbf{a}^H(u_i)\mathbf{S}_{\tau_j}\mathbf{S}^H\mathbf{a}(u_0) \right) \tag{2.20}$$

where the first part is the filtered output from the target echo, and the second part is the filtered output from the noise and clutter modeled in the clutter map. Taking the energy received from target echo and clutter echoes, the SCNR is given by

$$\text{SCNR} = \frac{|\mathbf{a}^H(u_0)\mathbf{S}\mathbf{S}^H\mathbf{a}(u_0)|^2}{\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j}\, |AF_{i,j}|^2} \tag{2.21}$$

where we have inserted the ambiguity function matrix $AF_{i,j} = \mathbf{a}^H(u_i)\mathbf{S}_{\tau_j}\mathbf{S}^H\mathbf{a}(u_0)$. We can note that the integrated clutter power in the denominator in this formulation corresponds to a weighted integrated sidelobe level using the clutter map as weights, as previously presented in Section 2.1.3.3.

Equation (2.21) is the single receive-channel SCNR used throughout this thesis. If a multi-channel receive array is used instead and beamformed towards the target, an additional direction-dependent factor with the inner product of receive steering vectors would enter the target and clutter responses multiplicatively. This factor would generally reduce clutter returns arriving from directions different from the target direction. A more general formulation could also jointly optimize the transmitted waveform and the linear combination of the receive channels. These alternative formulations are not considered in this thesis.

It can be further noted that with the inclusion of additive gaussian noise on the receiving end $w \sim \mathcal{N}(\mathbf{0}, \sigma_n^2\mathbf{I})$, the corresponding formula would be given by

$$\text{SCNR} = \frac{|\mathbf{a}^H(u_0)\mathbf{S}\mathbf{S}^H\mathbf{a}(u_0)|^2}{\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j}\, |AF_{i,j}|^2 + \sigma_n^2 ||\mathbf{a}^H(u_0)\mathbf{S}||_2^2}, \tag{2.22}$$

which can be related to the signal-to-interference-noise ratio (SINR) formulation used in [13].

2.1.3.5 Bandwidth

The signal model used in this thesis is based on all antenna elements emitting signals with the same carrier wave frequency, however the phase-coded waveform design approach gives rise to a bandwidth dependent on the instantaneous phase jumps. Given the phase matrix $\mathbf{\Phi}$, we can define a wrapped phase increment

$$\Delta\Phi_{m,n} = \text{wrap}\left(\Phi_{m,n+1} - \Phi_{m,n} \right) \in [-\pi, \pi), \tag{2.23}$$

where m denotes the antenna element and n denotes the fast-time sample. The wrap function is defined as

$$\text{wrap}(x) = ((x + \pi) \pmod{2\pi}) - \pi. \tag{2.24}$$

The instantaneous frequency can then be described as [29]

$$f_m(t) = \frac{1}{2\pi} \frac{d\Phi_m(t)}{dt}, \tag{2.25}$$

which in the discrete phase-coded waveforms can be approximated as

$$\begin{aligned} \frac{d\Phi_m(t)}{dt} &\approx \frac{\Delta\Phi_{m,n}}{T_c} \Rightarrow \\ f_m(n) &\approx \frac{\Delta\Phi_{m,n}}{2\pi T_c} \end{aligned} \tag{2.26}$$

where T_c denotes the chip duration, i.e. the time interval between each discrete fast-time sample in the waveform. The abrupt phase transitions of phase-coded waveforms can lead to spectral leakage and waveform distortion from the radar system electronics [30]. Thus, limiting the phase increments to some maximum D_{max} can lead to better spectral compactness and containing the effective bandwidth. Limiting the phase increments can also be necessary to accommodate certain hardware restrictions, where there might be limitations on the hardware's ability to increment phases.

2.1.3.6 Diversity between multiple waveforms

When it comes to MIMO radar waveforms, diversity can often refer to the diversity between the waveforms sent by each antenna element, which is an essential advantage in comparison with traditional phased array radars [19]. This waveform diversity between a group of waveforms sent simultaneously over the antenna elements, which can be referred to as intragroup diversity, is not a focus of this thesis since the objective of maximizing signal strength on a known target location warrants a strong correlation within a sent waveform group. On the other hand, the diversity between different groups of waveforms, or intergroup diversity, is of interest both for the purpose of increased robustness and performance as well as it's potential robustness against jamming methods [31].

In related works, [31] implements a design algorithm for developing group orthogonal waveforms using the peak cross-correlation (PCL) as a metric to minimize in order to achieve intergroup diversity. By minimizing the correlation between the adjacent pulses transmitted by the radar, DRFM-based jamming becomes more difficult, since although they can quickly and accurately intercept waveforms, they must delay at least to the subsequent pulse before generating jamming signals [31].

Given a set of K phase matrices $\{\mathbf{\Phi}^{(k)}\}_{k=1}^{K}$ with corresponding set of waveform matrices $\{\mathbf{S}^{(k)}\}_{k=1}^{K}$, the cross-correlation between two of the waveforms is defined as

$$\mathbf{R}_{i,j}(\tau) = \mathbf{S}_{\tau}^{(i)}(\mathbf{S}^{(j)})^H \tag{2.27}$$

where i and j denotes the indices of the waveforms and $\mathbf{S}_\tau$ is again defined as the time-delayed waveform matrix in Eq. (2.12). Following, [31] the PCL is given as

$$\begin{aligned} \text{PCL} &= \frac{1}{N_b^2} \max_{i,j,\tau \in \mathcal{G}} \|\mathbf{R}_{i,j}(\tau)\|_{\max}^2 , \\ \mathcal{G} &= \{i, j, \tau \mid i, j \in \{1, \ldots, K\},\ i \neq j,\ \tau \in \{-N_b + 1, \ldots, N_b - 1\}\} , \end{aligned} \tag{2.28}$$

where $\|\cdot\|_{\max}$ denotes the entry-wise maximum absolute value, i.e.

$$\|\mathbf{R}_{i,j}(\tau)\|_{\max} = \max_{m,n} \left| [\mathbf{R}_{i,j}(\tau)]_{m,n} \right| . \tag{2.29}$$

The PCL therefore represents the worst-case squared intergroup cross-correlation peak over all waveform pairs, matrix entries, and time delays.

Group orthogonality and metrics as the PCL above are in the waveforms code-domain, but what a jammer actually observes is the far-field of the transmitted waveforms, as given in Eq. (2.1) [32]. This comes naturally from the fact that an observer in the far-field of a transmitted MIMO radar waveform observes a coherent sum of each antenna elements transmitted signal [18]. Therefore it can be argued that diversity metrics developed to mitigate DRFM jamming can benefit from being formulated in terms of the angular spectrum of the far-field signal.
Based on this, we formulate a far-field similarity metric $\text{sim}[i, j]$ between two waveforms. First, we take the far-field representation produced by the k-th waveform

$$\mathbf{s}_E(u)^{(k)} = \mathbf{a}^H(u)\mathbf{S}^{(k)} \in \mathbb{C}^{1 \times N_b} \tag{2.30}$$

where we use the notation $u = \sin(\theta)$ as previously. For any pair (i, j) we compare the two far-field signals represented by matrices $\mathbf{S}_E^{(i)}, \mathbf{S}_E^{(j)} \in \mathbb{C}^{N_u \times N_b}$ whose N_u rows correspond to discrete angle bins. The far-field matrices are normalized according to

$$\tilde{\mathbf{s}}_{\mathbf{E}}^{(k)}[u, :] = \frac{{\mathbf{s}_{\mathbf{E}}}^{(k)}[u, :]}{||{\mathbf{s}_{\mathbf{E}}}^{(k)}[u, :]||_2}. \tag{2.31}$$

The similarity between the two far-field matrices is then defined as the mean absolute inner product of the corresponding rows

$$\text{sim}[i, j] = \frac{1}{N_u} \sum_{u=1}^{N_u} \left| \frac{1}{N_b} \sum_{n=1}^{N_b} \tilde{\mathbf{s}}_{\mathbf{E}}^{(i)}[u, n] \tilde{\mathbf{s}}_{\mathbf{E}}^{(j)H}[u, n] \right| \in [0, 1] \tag{2.32}$$

.
When $\text{sim}[i, j] = 1$ the two far-fields are identical (up to a global phase), whereas $\text{sim}[i, j] = 0$ indicates complete orthogonality in every angle bin. It should be noted that this can be extended to cover different time lags τ between the compared far-field angular spectrums:

$$\text{sim}[i, j, \tau] = \frac{1}{N_u} \sum_{u=1}^{N_u} \left| \frac{1}{N_b} \sum_{n=1}^{N_b} \tilde{\mathbf{s}}_{\mathbf{E}}^{(i)}[u, n + \tau] \tilde{\mathbf{s}}_{\mathbf{E}}^{(j)H}[u, n] \right|. \tag{2.33}$$

The extension to cover time lag τ could then be used to formulate a worst-case measure, by extracting the maximum value across the different lags. This is however not something that is implemented throughout this report, instead the zero lag version in Eq. (2.32) is used.

2.2 Machine Learning

Machine Learning (ML) is a subfield within Artificial Intelligence (AI) that focuses on how statistical models can learn from data to identify patterns that generalize to unseen data [33].

2.2.1 General machine learning

This section describes some of the fundamental principles of machine learning relevant to the implementations throughout this thesis. For more implementation-specific details and descriptions, see Chapter 3.

2.2.1.1 Input-Output Mapping

The general goal for Machine Learning (ML) is to approximate a mapping function from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. Formally, this can be expressed as learning an approximate function $\hat{f}$ that maps the input variables $\mathbf{x} \in \mathcal{X}$ to the output variables $\mathbf{y} \in \mathcal{Y}$ according to:

$$\hat{\mathbf{y}} = \hat{f}(\mathbf{x}). \tag{2.34}$$

The input variables $\mathbf{x}$ consist of all input features that the model should consider when predicting the output variables $\hat{\mathbf{y}}$. Both the model's output $\hat{\mathbf{y}}$ and any desired output variables $\mathbf{y}$ belong to the output space $\mathcal{Y}$, where desired output variables $\mathbf{y}$ are often called targets, ground truth or labels in different datasets [33].

For most real-world problems, the exact mapping function f between the input and output space is unknown. Machine learning is therefore used to learn an approximate mapping, $\hat{f}$, such that its predictions $\hat{\mathbf{y}}$ perform well according to a defined objective function $\mathcal{L}$.

2.2.1.2 Parametrization of $\hat{f}$

To find a suitable mapping function $\hat{f}$, a machine learning model's architecture and size are chosen to parametrize $\hat{f}$ which constrains the space of possible mapping functions. The structure and size of the model is chosen with regards to the data and problem at hand. The model is defined by a set of parameters $\boldsymbol{\theta}$ which controls how the mapping function behaves. The learning process of a machine learning model involves optimizing the parameters $\boldsymbol{\theta}$ with respect to the objective function in order to adjust $\hat{f}$ to capture relevant structure in the training data.

2.2.1.3 Maximum Likelihood Estimation

Maximum likelihood estimation is a statistical framework used to optimize model parameters, where the parameters $\boldsymbol{\theta}$ are updated to maximize the likelihood that the observed data were generated by the model. This corresponds to adjusting the approximated mapping function $\hat{f}$ such that the model predictions align more closely with the underlying data distribution and the training objective given the input.

Under the assumption that the dataset consists of N independently and identically distributed (IID) drawn samples, the likelihood of the dataset can be expressed as the product of the likelihood of individual samples indexed by i.

$$\mathcal{L}_{\text{MLE}}(\boldsymbol{\theta}) = \prod_{i=0}^{N-1} p_{\boldsymbol{\theta}}(\mathbf{y}_i|\mathbf{x}_i). \tag{2.35}$$

To make the MLE objective suitable for a wide range of optimization techniques, this maximization problem is rewritten as a minimization problem. Using that the logarithm is a monotonically increasing function one often implements a logarithmic version of $\mathcal{L}_{\text{MLE}}(\boldsymbol{\theta})$ which improves the numerical stability by transforming a product to a sum. This results in the *negative log-likelihood (NLL)* described in Equation (2.36) [33].

$$\mathcal{L}_{NLL}(\boldsymbol{\theta}) = -\sum_{i=0}^{N-1} \log p_{\boldsymbol{\theta}}(\mathbf{y}_i|\mathbf{x}_i), \tag{2.36}$$

2.2.1.4 Empirical Risk and loss function

Real-world datasets often contain an infinite number of possible data points where only a fraction can be attained to train a model, this makes a true risk assessment of the underlying mapping f unattainable. Therefore, it is often only possible to minimize the discrepancy between the approximate and the underlying mapping function for a subset of the possible dataset, a process known as empirical risk minimization, where a number of data points, defined as the *batch size*, is considered to empirically estimate the discrepancy between $\hat{f}$ and f. In most cases, the goal of a model is to learn the underlying mapping function for a large portion of the input space. Since the model is adjusted towards a subset of the input space, the distribution of the available training and evaluation data becomes crucial in order to generalize across the input space.

The loss function, denoted as $\mathcal{L}$, is introduced as a practical proxy for quantifying the empirical risk. $\mathcal{L}$ is a function of the model outputs $\hat{y}$ and other relevant variables that together defines the optimization objective. In several machine learning applications, probabilistic assumptions about the output space are used to derive specific loss functions from the negative log-likelihood in Equation (2.36), providing a differentiable metric that describes the discrepancy between the underlying and approximate mappings from the input space to the output space. This way of deriving the loss function can be seen as the practical implementation of the MLE principle described in Section 2.2.1.3 [33].

How the loss function is designed and which terms it includes influence the optimization landscape and determines which features that the machine learning model should learn to prioritize. In many cases, standard likelihood formulations under assumptions of the dataset cannot fully capture all relevant characteristics of the data. For these cases, a loss function can be custom made as long as it is differentiable, which is the criteria to make backpropagation and weight optimization work.

2.2.1.5 Backpropagation, optimization and Stochastic gradient descent (SGD)

The objective of training a machine learning model is to minimize the discrepancy between the approximate mapping $\hat{f}$ and the underlying mapping f between the input and output space. This is achieved through updates of the model parameters $\boldsymbol{\theta}$ to minimize the empirical risk, an iterative process guided by the gradient of the loss function.

Backpropagation is the method that drives this process. It uses the chain rule to compute the partial derivatives of the loss function $\mathcal{L}$ with respect to each of the trainable model parameters θ_i through the model output $\hat{\mathbf{y}}$. These partial derivatives are used to propagate the loss signal from the output of the model back to each individual parameter that came before, summarized in a gradient with respect to the parameters $\boldsymbol{\theta}$ according to:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \theta_1}, \frac{\partial \mathcal{L}}{\partial \theta_2}, \ldots, \frac{\partial \mathcal{L}}{\partial \theta_n} \right]^\top . \tag{2.37}$$

The gradients carry information about how a change in each parameter affects the loss value. This is utilized in the optimization of the model parameters by updating each parameter in the reverse direction of the gradient to move towards a parameter set $\boldsymbol{\theta}$ that corresponds to a local minimum of the loss function according to the following update schema:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \nabla_{\boldsymbol{\theta}_t} \mathcal{L}, \tag{2.38}$$

where the parameter η is the *learning rate* which decides step length and thereby how large the parameter updates will be in the gradient direction.

The computation of gradients and updating the parameters for all data points in a dataset at the same time is often computationally infeasible, therefore one could compute the gradients and update the parameters of the model for a random sampled batch of the dataset. This is often called Stochastic gradient descent or mini-batch Stochastic gradient descent [34].

2.2.1.6 Generalization

Generalization is measured in how well the model's learned approximate mapping $\hat{f}$ performs for data that the model was not trained on and shows the discrepancy

between $\hat{f}$ and f. The ability to generalize to unseen data describes if the model has learned to memorize the noise of the training data or if it has extracted a general underlying pattern from the data. The performance difference for training data compared to unseen data can be analyzed to understand the model's generalization capabilities.

The model's generalization ability is governed by the model architecture and model size, which controls the complexity of the learned mapping $\hat{f}$ and thereby how sensitive the trained model can become to variations in the input data. To measure the models ability to generalize, one can split the available data into three different dataset [35].

The *training dataset* is used for computing gradients and updating the model parameters, while the *validation dataset* consists of unseen data points that the model does not use to adjust its parameters, but rather to evaluate the model's performance on an unseen set of data points. This dataset is also used to determine how to adjust the model's architecture and training process to improve performance and generalization.

The third part, the *test dataset*, contains unseen data used to judge the model's true performance and ability to generalize. Since this dataset has been withheld from the model during both training and architecture improvements, it provides an unbiased estimate of the model's generalization performance.

2.2.1.7 Regularization

Regularization is a technique used to navigate the optimization of the model parameters, by constraining the model's learning process. Regularizing the learning process of a machine learning model can be done in various ways [34].

One example is by using direct additions of constraints and penalties to the optimization objective. For example, additional terms in the loss function.

Another aspect of regularization involves methods that indirectly steer the learned mapping function $\hat{f}$ to become less noise sensitive, which can be done through the use of early stopping or data normalization.

By implementing regularization, models with higher complexity can be utilized, enabling them to learn more complex mapping functions while reducing the risk of learning the noise in the training data. Regularization achieves this by limiting the model's ability to adapt to noise present in the training data. Consequently, the model becomes more robust and is more likely to generalize to unseen data [34].

2.2.1.8 Deep Machine Learning

Deep machine learning is a subfield within machine learning that aims to approximate $\hat{f}$ using multiple layers of differentiable transformations. The structure of the

models within this category uses multiple nested functions to construct $\hat{f}$ and a simple version of these models can be expressed through:

$$\hat{f}(x) = f_{L-1}(f_{L-2}(\dots f_l(\dots f_1(f_0(x))))) \tag{2.39}$$

where f_l represents the transformation made in layer l of a model with L layers.

A primary advantage of using deep learning is its ability to approximate highly non-linear relationships in the data. This is made possible by the addition of non-linear *activation functions* between each layer, l. Without the activation functions, the combined transformations made in all layers would mathematically collapse into a single effective linear transform regardless of the model's depth. The use of multiple linear transformations and non-linear activation functions makes it possible for the model to extract different levels of abstraction from the data, where earlier layers typically focus on low-level features while later layers can learn high level contexts [34].

2.2.2 Learning paradigms

There are multiple paradigms one can use to optimize $\hat{f}$ that are more or less suitable for different problem domains and data availability. This subsection describes the paradigms used during this work.

2.2.2.1 Supervised Learning

The basis of supervised learning is that the optimization of $\hat{f}$ is done by using pairs of input, $\mathbf{x}_i$ and output, $\mathbf{y}_i$ samples, using sample index i. Since the goal of the trained model is to adjust its approximation $\hat{f}$ to be as close as possible to the underlying mapping f for all data pairs $\mathbf{x}_i, \mathbf{y}_i$, this paradigm uses a loss function that compares the model's predictions $\hat{\mathbf{y}}_i = \hat{f}(\mathbf{x}_i)$ to the ground truth output data $\mathbf{y}_i = f(\mathbf{x}_i)$ which is used to guide model weight updates and minimize a supervised loss function [35]

$$\mathcal{L}_{supervised} = \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i). \tag{2.40}$$

This type of learning is primarily data driven, as the model attempts to learn the underlying mapping f by interpolating between known input-output pairs, typically without incorporating prior domain-specific information and rules that the data distribution lives under.

2.2.2.2 Self-supervised Learning

Self-supervised learning is another learning paradigm where $\hat{f}$ is optimized by only using input data examples $\mathbf{x}_i$ without any target examples of output data $\mathbf{y}_i$. Instead, the learning process involves using the model's own outputs $\hat{\mathbf{y}}_i = \hat{f}(\mathbf{x}_i)$ to directly evaluate the loss $\mathcal{L}(\hat{\mathbf{y}}_i)$ and extract loss signals to optimize the model parameters, without the need for true output values $\mathbf{y}_i$ [36].

2.2.2.3 Informed Machine Learning (IML)

Deep Machine learning has been and is being used to solve various problems in a range of domains with great success. Traditional approaches focus on a purely data driven training process of the models which for several objectives require massive amounts of data to result in generalizable and reliable models. Having access to large amounts of data can however still lead to unsatisfactory results for applications where the predictions needs to follow natural laws or other requirements to be useful. *Informed Machine learning* is defined, according to [37], as learning from a hybrid information source that consists of data and prior knowledge. The prior knowledge is given by formal representations and integrated into the machine learning pipeline. This integration can be done in various ways, transforming the data into specific formats or versions, adjusting model architectures towards the objective at hand, integrating regularizing terms or modifying the loss function of the training procedure.

Using IML and incorporating prior knowledge of the objective results in a constrained optimization landscape. Instead of letting the model search for parameter solutions in the full parameter space, the embedded knowledge aims to limit the parameter space to only valid model configurations, thereby simplifying the model's learning process. By explicitly guiding the model gradients towards valid sub-regions of the solution space, the model can be trained to directly solve the objective rather than solving a proxy for it. This can be crucial in situations where a good proxy for the main objective is difficult to formalize or configure to be sensitive and accurate enough for the task at hand. Integrating IML can lead to faster convergence, a more data-efficient training process and robust models that are consistent with the underlying domain rules.

Physics-Informed Machine Learning (PIML) can be seen as a subfield of IML where the prior knowledge consists of physical constraints and mathematical models from scientific and engineering domains, which is integrated into the construction of the machine learning pipeline [38].

2.2.2.4 Combining Informed Machine Learning with Self-supervised Learning

Combining these two ways for how a machine learning model can learn the underlying mapping function f, one can gain several advantages. Training a model in a self-supervised fashion eliminates the need for true output examples. In cases where few input-output pairs exist but there are large amounts of input data available, the amount of data does not hinder a model's performance limit when self-supervised learning is used. When introducing differentiable physical constraints and mathematical models for evaluating the model output, it both guides the model by constraining the output space into a smaller subspace of valid outputs and it makes the model outputs adhere to physical constraints which can be crucial for how the model is used.

2.2.3 Deterministic versus Generative models

Deterministic models These models learns a direct mapping from the input to the output space. Given the same input $\mathbf{x}_i$ the model always produces the same output $\hat{\mathbf{y}}_i = f(\mathbf{x}_i)$.

Generative models These are models that treat the output generation as an iterative refinement process, where a random start sample is drawn and the model is used to iteratively refine or denoise the sample conditioned on the input variable $\mathbf{x}_i$ to generate the final output $\hat{\mathbf{y}}_i$. This approach enables sampling which can produce more diverse outputs $\hat{\mathbf{y}}_i = f(\mathbf{y}_{sample,i}|\mathbf{x}_i)$.

2.2.4 Architectural foundations

The following sections will go through architectural theory behind the models that are considered in this report. More details on exact model implementations can be found in the Sections under 3.3.

2.2.5 Architectural building blocks

This section describes the fundamental neural network layers used to build the considered models in this work.

2.2.5.1 Multi-layer perceptron (MLP)

A Multi-layer perceptron (MLP) consists of a stack of fully connected layers, where each neuron in the preceding layer is connected to every neuron in the subsequent layer through a learnable weight matrix, $\mathbf{W}$. Each layer that maps from N input dimensions to M output dimensions computes:

$$\mathbf{a} = \sigma(\underbrace{\mathbf{W}\mathbf{x} + \mathbf{b}}_{\mathbf{z}}), \tag{2.41}$$

where $\mathbf{x} \in \mathbb{R}^N$ is the input vector, $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^M$ is the bias vector. The resulting vector $\mathbf{z}$ is the intermediate output of the layer, which passes through a non-linear activation function σ to produce the activation vector for that layer denoted $\mathbf{a}$. This vector then serves as the input of the next layer [34].

The computation described in Equation (2.41) is repeated for all layers in the MLP stack to produce the final output vector of the MLP.

2.2.5.2 Convolutional layers

A convolutional layer uses N learnable kernels or filters, consisting of trainable weights, that operate on the spatial dimensions of the input tensor $\mathbf{X}$ to extract local spatial information. Each filter is applied by sliding it over the input data according to the *stride* parameter, which describes the step size between consecutive filter positions. At each position, a dot product is computed between the filter

weights and the input data with the corresponding overlapping region. The spatial size of the convolutional layer's output can be controlled by *padding*, which involves adding padding elements to the boundaries of the input data.

Each dot product is stored as a point in the feature map $\mathbf{O}$ for the specific filter, which builds the output of a convolutional layer once the filter has traversed all possible positions of the input data. This is done for all filters resulting in an output that consists of N feature maps. When using two dimensional input data, a two dimensional kernel $\mathbf{K}$ is defined with the spatial shape $(k_h \times k_w)$. If the input data contains multiple channels, e.g RGB colors, the kernel is extended to a three dimensional kernel in order to match the number of channels in the input data. When there are multiple input channels, the filter sums the dot products across all channels to produce a single value for each point in the output feature map $\mathbf{O}$. The convolution computation for a single channel input with the kernel $\mathbf{K}$ can be expressed through Equation (2.42) [34].

$$\mathbf{O}_{i,j} = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \mathbf{X}_{i+m,j+n} \mathbf{K}_{m,n}. \tag{2.42}$$

2.2.5.3 Transposed convolutional layers

A transposed convolutional layer is used to increase the spatial resolution and operates by multiplying each element of the input data $\mathbf{X}$ by the entire kernel $\mathbf{K}$ to generate intermediate tensors $\mathbf{H}^{(i,j)}$. Similarly to the regular convolutional layers, *stride* and *padding* are used to control output size of the layer by affecting how the kernel is applied and slid across the input data. These intermediate tensors $\mathbf{H}^{(i,j)}$ are placed in a larger output matrix based on the input element's position (i, j) that was used during multiplication with the kernel. This set of multiple intermediate tensors is thereafter summed in regions where the intermediate tensors overlap to create the final output tensor $\mathbf{O}$ [39]. The computation can be conceptually described as:

$$\mathbf{O} = \sum_i \sum_j \mathbf{X}_{i,j} \cdot \mathbf{P}_{i,j}(\mathbf{K}), \tag{2.43}$$

where $\mathbf{P}_{i,j}(\mathbf{K})$ denotes the kernel $\mathbf{K}$ placed in the output tensor at the spatial location corresponding to the input element $\mathbf{X}_{i,j}$.

2.2.6 Deterministic Single-pass architectures

The following sections describe the logic behind the two *deterministic models* used in this work.

2.2.6.1 Convolutional Autoencoder (CAE)

The concept of autoencoders originated from the application of neural networks for dimensionality reduction and feature extraction. An autoencoder's fundamental principle is to learn a low dimensional representation of a dataset by focusing on the most prominent features required to establish a robust mapping $\hat{f}$ [34]. The

architecture consists of an encoder $E_{\boldsymbol{\varphi}}$ that transforms the input $\mathbf{x}_i$, where i denotes the sample index, to a latent representation $\mathbf{z}_i \in \mathbb{R}^{d_z}$, and a decoder $D_{\boldsymbol{\psi}}$ that transforms the latent representation $\mathbf{z}_i$ to the output $\mathbf{y}_i$, which can be described through:

$$\mathbf{z}_i = E_{\boldsymbol{\varphi}}(\mathbf{x}_i), \quad \mathbf{y}_i = D_{\boldsymbol{\psi}}(\mathbf{z}_i) \quad \Longrightarrow \quad \mathbf{y}_i = D_{\boldsymbol{\psi}}(E_{\boldsymbol{\varphi}}(\mathbf{x}_i)), \tag{2.44}$$

where the encoder mapping $E_{\boldsymbol{\varphi}}$ is parametrized by the set of weights $\boldsymbol{\varphi}$ and the decoder mapping $D_{\boldsymbol{\psi}}$ by the parameter set $\boldsymbol{\psi}$.

For Convolutional Autoencoders, the architecture of the encoder $E_{\boldsymbol{\varphi}}$ leverage convolutional layers to take advantage of spatial structures within tensor data. Each layer applies learnable filters to detect localized features in the layers input data and reduce the spatial dimensions. Therefore, the use of multiple convolutional layers leads to an encoder where the subsequent layers gradually extracts features of increasing complexity and abstraction.

In the final stage of the encoder, the resulting abstract features, $\mathbf{H}_{enc} \in \mathbb{R}^{C \times H \times W}$ are compressed into a latent representation $\mathbf{z}_i$ through a flattening operation followed by a MLP bottleneck, $\mathcal{F}_{\varphi}$, through:

$$\mathbf{h}_{flat} = \text{flat}(\mathbf{H}_{enc}), \quad \mathbf{z}_i = \mathcal{F}_{\varphi}(\mathbf{h}_{flat}), \quad \mathbf{h}_{flat} \in \mathbb{R}^D, D = C \times H \times W. \tag{2.45}$$

The decoder D_{ψ} is used to perform this process in reverse. Similarly, the decoder uses an MLP module, $\mathcal{G}_{\psi}$, followed by a reshape to project the latent representation $\mathbf{z}_i$ into a spatially structured tensor. This tensor version of the latent representation is used as the input to sequential transposed convolution layers. Unlike regular convolutional layers, which typically downsamples and extracts features by aggregating spatial information, transposed convolutional layers increase spatial dimensions through learnable filters, stride and appropriate padding [34].

2.2.6.2 Vision Transformer

Vision Transformers (ViTs) adapt the standard transformer architecture to handle image data by dividing images into two-dimensional patches, which are projected into embedding vectors and passed as sequential tokens together with positional encodings, similarly to how words in text are processed [40]. In this section, spatially structured tensor data is referred to as an image for simplicity.

The input image is represented as $\mathbf{X}_i \in \mathbb{R}^{C \times H \times W}$, where i denotes the data sample index and C is the number of input channels, while H and W are the height and width of the input image. This image is split into T non-overlapping vectorized patches $\mathbf{x}_{i,p} \in \mathbb{R}^{P^2 \cdot C}$, where $p \in \{0, \ldots, T-1\}$ indexes patches and P is the patch size. When H or W is not divisible by P, appropriate padding is applied, yielding $T = \left\lceil \frac{H}{P} \right\rceil \cdot \left\lceil \frac{W}{P} \right\rceil$.

The embedding projection, $\mathbf{E} \in \mathbb{R}^{P^2 \cdot C \times d_{model}}$, is a learnable mapping that projects each patch vector $\mathbf{x}_{i,p}$ into the d_{model}-dimensional embedding space. These embedded

vectors are combined with positional encodings $\mathbf{E}_{pos} \in \mathbb{R}^{(T+1)\times d_{model}}$ to form the input to the initial transformer block $\mathbf{Z}_{i,0}$ through:

$$\begin{aligned}\mathbf{Z}_{i,0} = [\mathbf{x}_{class}, \mathbf{x}_{i,0}\mathbf{E}, \mathbf{x}_{i,1}\mathbf{E}, \ldots, \mathbf{x}_{i,T-2}\mathbf{E}, \mathbf{x}_{i,T-1}\mathbf{E}] + \\ [\mathbf{E}_{pos,0}, \mathbf{E}_{pos,1}, \ldots, \mathbf{E}_{pos,T-1}, \mathbf{E}_{pos,T}],\end{aligned} \tag{2.46}$$

where the second index of $\mathbf{Z}$ denotes the transformer block index. The vector $\mathbf{x}_{class} \in \mathbb{R}^{d_{model}}$ is a learnable embedding vector that is prepended to the sequence of patch embeddings. Its state at the output of the transformer encoder serves as the global contextualized representation of the image and is used for downstream tasks.

The resulting sequence of token embeddings, $\mathbf{Z}_{i,0}$, is fed into a series of transformer encoder blocks. Each encoder block contains multi-head self-attention (MSA) and an MLP connected by pre-normalization layers (LN) and residual connections. The self-attention mechanism enables the model to attend to different parts of the input sequence, thereby allowing it to focus on different patches of the input image. However, the self-attention computation does not inherently account for the order of the input tokens, which is why the positional encodings are needed for the attention computation [40]. By extending the attention mechanism to multi-head self-attention, the model uses several attention heads that each have the ability to focus on different aspects or relationships between patches and capture a more nuanced view of the input.

The attention for each head h, given the input $\mathbf{Z}_{i,b-1}$, is computed using Equations (2.47), (2.48) and (2.49) following [41].

$$\mathbf{Q}_h = \mathbf{Z}_{i,b-1}\mathbf{W}_{\mathbf{Q}_h} \quad , \quad \mathbf{K}_h = \mathbf{Z}_{i,b-1}\mathbf{W}_{\mathbf{K}_h} \quad , \quad \mathbf{V}_h = \mathbf{Z}_{i,b-1}\mathbf{W}_{\mathbf{V}_h}, \tag{2.47}$$

where the input is projected into query ($\mathbf{Q}_h$), key ($\mathbf{K}_h$) and value ($\mathbf{V}_h$) representations. These representations are then used to compute attention weights from the dot products between the query and key vectors. The attention weights are applied to the value representation of the input sequence through:

$$\text{head}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h) = \text{softmax}\left(\frac{\mathbf{Q}_h\mathbf{K}_h^T}{\sqrt{d_k}}\right)\mathbf{V}_h, \tag{2.48}$$

where d_k is the dimension of each query and key vector contained in the matrices $\mathbf{Q}_h$ and $\mathbf{K}_h$, which each contain one vector for every token in the input sequence. The final representation that forms the output sequence of the multi-head attention block is computed by concatenating the results for each head_h and projecting back to the original dimension d_{model} of the input embeddings according to:

$$\text{MSA}(\mathbf{Z}_{i,b-1}) = [\text{head}_1; \text{head}_2; \ldots; \text{head}_k]\mathbf{W}_O. \tag{2.49}$$

In each block b, the embedded input sequence $\mathbf{Z}_{i,b-1}$ gets transformed into the contextualized output sequence $\mathbf{Z}_{i,b}$ through MSA according to:

$$\mathbf{Z}'_{i,b-1} = \text{MSA}(\text{LN}(\mathbf{Z}_{i,b-1})) + \mathbf{Z}_{i,b-1}, \tag{2.50}$$

and through the MLP layers by:

$$\mathbf{Z}_{i,b} = \mathrm{MLP}(\mathrm{LN}(\mathbf{Z}'_{i,b-1})) + \mathbf{Z}'_{i,b-1}, \tag{2.51}$$

where the MLP consists of sequential fully connected layers and non-linear activation functions.

By stacking L encoder blocks, the ViT progressively builds representations of increasing complexity and abstraction. Self-attention makes it possible to learn global dependencies between tokens, while residual connections help preserve information across layers and stabilize optimization. The output of the ViT is extracted from the class-token position of the final representation $\mathbf{Z}_{i,L}$, which is a latent representation of the input image that can be used for various downstream tasks.

2.2.7 Iterative refinement architectures

In this work iterative refinement architectures refers to model structures that start from a sample and iteratively adjusts the initial sample in order to improve the generated output.

2.2.7.1 Residual Networks

Residual networks consist of multiple blocks that are connected through residual connections. Each residual block learns a residual mapping, $F(\mathbf{x})$, which is added element-wise to the input of the block, $\mathbf{x}$, through a skip connection. The output of the block can therefore be written as:

$$H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}. \tag{2.52}$$

This creates an identity shortcut over the layers inside the residual block, allowing the network to learn the residual function:

$$F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}, \tag{2.53}$$

instead of the full underlying mapping $H(\mathbf{x})$. This enables the model to refine features in multiple steps, since the layers do not need to rediscover the entire identity mapping to preserve previously extracted information.

Residual connections also allow gradients to flow more easily, since the skip connection provides a direct path for gradients to propagate backward through the network, mitigating the vanishing gradient problem that can occur in deep multilayer networks [42].

Residual connections can be incorporated into many neural network architectures, which makes it possible to add residual connections to parts of models that can benefit from an increased depth or complexity.

Residual connections between multiple network blocks can therefore be used to iteratively refine an initial random sample. This results in a generative model that has the ability to focus on finer details in the refining process of the starting sample.

2.2.7.2 Score-based generative models for self-supervised learning

Score-based models provide a framework for evolving randomly sampled data into meaningful samples by following a learned gradient field. The score in this score-based model refers to the gradient field, $\nabla_x \log p(\mathbf{x}_t)$, which represents the gradient of the log-probability density of the data distribution $p(\mathbf{x}_t)$ at a specific noise level or time step t. This gradient points in the direction of the steepest increase in probability density, which can be used to move the sample $\mathbf{x}_t$ toward regions that are more likely under the data distribution $p(\mathbf{x}_t)$. The gradient field can be used over several time steps to adjust the sampled data in a direction that moves the samples towards a desired denoised distribution.

Traditionally, score-based models are trained using a denoising objective where samples from the target distribution $p(\mathbf{x})$ are available. This makes it possible to define a forward diffusion process that gradually perturbs the target data by adding noise, producing noisy samples $\mathbf{x}_t$ that correspond to different time steps t. Therefore it is possible to train a score-based model to approximate the gradient field for multiple time steps of the noising process. This model of the score function, $f(\mathbf{x}_t, t) \approx \nabla_x \log p(\mathbf{x}_t)$, can thereafter be used to evolve a random sample into an output similar to the target data that it was trained on by a reverse diffusion process, effectively sampling from the target data distribution $p(\mathbf{x})$ [43].

One way of performing the reverse diffusion process using the score function and sample from the data distribution $p(\mathbf{x})$ is to use Langevin dynamics [44]. Starting from an initial random sampled data point $\mathbf{x}_0$, with a step size $\epsilon_t > 0$ the Langevin method recursively denoises the sample through:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \frac{\epsilon_t}{2} \underbrace{\nabla_x \log p(\mathbf{x}_{t-1})}_{f(\mathbf{x}_{t-1}, t)} + \sqrt{\epsilon_t}\mathbf{z}_t, \tag{2.54}$$

where $\mathbf{z}_t \sim \mathcal{N}(0, I)$. The distribution of $\mathbf{x}_T$ converges to the target distribution $p(\mathbf{x})$ when $\epsilon_t \to 0$ and the total number of diffusion steps $T \to \infty$, in which case the data point $\mathbf{x}_T$ becomes a sample from $p(\mathbf{x})$. The denoising steps and trade-off between the gradient-based updates and the added noise throughout the process is controlled by the step size ϵ_t [45].

A trained model that approximates the score function, $f(\mathbf{x}_t, t)$, can also be conditioned on other features to guide the reverse diffusion process toward different regions of $p(\mathbf{x})$ based on the specified conditions. The model can therefore be expressed as $f(\mathbf{x}_t, t, \mathbf{c}_i) \approx \nabla_x \log p(\mathbf{x}_t|\mathbf{c}_i)$, where $\mathbf{c}_i$ represents the additional condition.

In the case of self-supervised learning, samples from the target distribution are not needed. Instead, the score function $\nabla_x \log p(\mathbf{x}_t|\mathbf{c}_i)$ that the model tries to mimic is replaced by the negative gradient of the conditional loss function $-\nabla_x \mathcal{L}(\mathbf{x}_t, \mathbf{c}_i)$. This transforms the backward diffusion process to instead of adjusting $\mathbf{x}_t$ toward regions that are likely under $p(\mathbf{x})$, the process moves $\mathbf{x}_t$ towards regions that minimize the loss function. In this setting, the target distribution is defined by the loss

function, where maximizing the likelihood of $\mathbf{x}$ is equivalent to minimizing the energy landscape defined by $\mathcal{L}(\mathbf{x}_t, \mathbf{c}_i)$.

3
Methods

This chapter presents the methods implemented in order to investigate neural-network based waveform generation for MIMO radar in the presence of clutter. First, we detail the methodology for generating clutter maps as training data for the neural networks, as well as the different datasets that were used and their corresponding characteristics. Further, the benchmark datasets of clutter-maps and corresponding optimized waveforms obtained from Saab AB are presented briefly. Next, the design and implementation of the neural-network architectures used throughout the thesis are detailed, with their structure presented and descriptions of the hyperparameters that vary throughout different experiments and results. Additionally, the specific loss functions used to train the neural networks are presented, together with the different evaluation metrics used to measure performance and characteristics of the neural-network generated waveforms. Finally, the experimental setups used to investigate different properties of neural-network based waveform generation is presented, including the training setups, model sizes and hyperparameters.

3.1 Data

This section presents the methods used to obtain synthetically generated datasets of clutter maps used to train the neural networks, the pre-processing of data before fed as input to neural networks and the datasets of clutter maps with corresponding optimized waveforms provided by Saab AB. The training of neural networks are done primarily in a self-supervised training regime, meaning that pairs of clutter map - optimized waveform are not primarily used for training. One experiment does however compare supervised and self-supervised learning, thereby using pairs of labeled data. For the other experiments and results, the labeled datasets are primarily used as a benchmark to compare the neural-network approach to an optimization based approach.

3.1.1 Data generation

The experiments and training uses synthetically generated clutter-maps, which are matrices $\mathbf{CM} \in \mathbb{R}^{(N_u \times N_r)}$ determining the weight of unwanted signal returns from different range-angle bins (excluding the target location in the center of the map). The generation process starts from a uniform random field

$$\mathbf{W}^{(0)} \sim \mathcal{U}(0, 1) \tag{3.1}$$

initializing a raw texture with no spatial correlation. A triangular kernel is used for low-pass filtering of $\mathbf{W}^{(0)}$ smoothing the random field to obtain spatially correlated distribution. First, a one-dimensional triangular window $\mathbf{h} \in \mathbb{R}^{(1\times N_h)}$ in the range (r) and angle (u) directions is defined with elements

$$h[n] = 1 - \frac{|2n - N_h - 1|}{N_h}, \qquad n = 1, 2, \ldots, N_h. \tag{3.2}$$

The filter size N_h is an integer defined parametrically depending on N_u, with $N_h = \lfloor K_f \cdot N_u \rfloor$ where $K_f > 0$ and $\lfloor \cdot \rfloor$ denotes the floor function. The two-dimensional kernel is then defined as

$$\mathbf{H} = \mathbf{h}^T \mathbf{h}, \tag{3.3}$$

where we have $\mathbf{H} \in \mathbb{R}^{(N_h \times N_h)}$. In Figure 3.1, an example of a kernel $\mathbf{H}$ can be seen, where the filter size is set to $N_h = 15$.

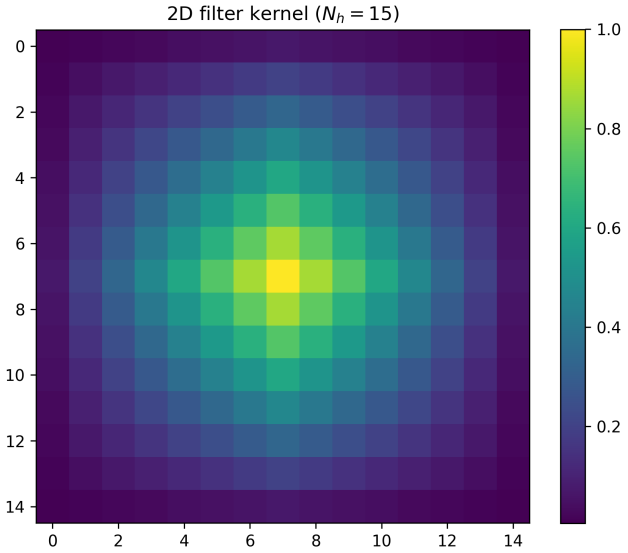


Figure 3.1: An example of a kernel $\mathbf{H}$ used to smoothen the initial distribution of uniformly random sampled points, which later become synthetically generated clutter maps. The example shown has a dimension of $N_h = 15$, making the filter an (15×15) matrix.

Using the kernel $\mathbf{H}$, the field $\mathbf{W}^{(0)}$ is filtered via a convolution

$$\mathbf{W}^{(1)} = \left(\left[\mathbf{W}^{(0)} * \mathbf{H}\right]_{same}\right) \oslash \left(\left(\left[\mathbf{J}_{N_u \times N_r} * \mathbf{H}\right]_{same}\right)^{\circ\alpha}\right) \qquad \in \mathbb{R}^{N_u \times N_r}, \tag{3.4}$$

with zero-padding around the field $\mathbf{W}^{(0)}$ ensuring that the dimension of $\mathbf{W}^{(1)}$ remains $(N_u \times N_r)$. Here $*$ denotes the convolution operator, and $\left[\mathbf{A} * \mathbf{B}\right]_{same}$ denotes a convolutional operation where $\mathbf{A}$ is padded such that the result has the same dimensionality as $\mathbf{B}$. The $\oslash$ denotes Hadamard division, otherwise known as element-wise division, and $(\cdot)^{\circ\alpha}$ denotes Hadamard (element-wise) exponentiation. The matrix $\mathbf{J}_{N_u \times N_r}$ is a matrix of all ones with dimensions $(N_u \times N_r)$. The parameter $0 < \alpha \leq 1$

is introduced to vary the spatial distribution more to the middle (smaller α) or more spread out to the sides (larger α).

To obtain a sparse set of clutter regions, i.e. separate clusters of clutter points, the field is binarized using an empirical quantile. More specifically, for a chosen threshold value $0 < W_{thr} < 1$, we take the empirical quantile

$$T = \mathrm{Q}\Big(\mathbf{W}^{(1)}, W_{thr}\Big) \tag{3.5}$$

which is the value below which a proportion W_{thr} of the entries of $\mathbf{W}^{(1)}$ lie. The binary mask is then given by

$$\mathbf{B} = \mathbb{I}\Big(\mathbf{W}^{(1)} \geq T\Big) = \begin{cases} 1 & \text{if } \mathbf{W}^{(1)}i, j \geq T \\ 0 & \text{if } \mathbf{W}^{(1)}i, j < T \end{cases} \tag{3.6}$$

with $\mathbb{I}$ as the indicator function.

The binary mask $\mathbf{B}$ now defines the spatial distribution of clutter for a given clutter map, where the ones denote positions of clutter. The amplitudes of the clutter is modeled with a Weibull distribution. A set of uniform samples $\mathbf{U} \sim \mathcal{U}(0, 1)$ is transformed as

$$\mathbf{C}^{(0)} = \Big(-\ln\left(\mathbf{U}\right)\Big)^{\beta}, \tag{3.7}$$

where $\beta > 0$ is the Weibull shape parameter. This sample is then scaled according to

$$\mathbf{C}^{(1)} \leftarrow C_{lev}\left(\frac{\mathbf{C}^{(0)}}{\mu(\mathbf{C}^{(0)})}\right), \tag{3.8}$$

where $\mu(\mathbf{C}^{(0)})$ denotes the mean, resulting in a Weibull distribution with a mean value set by the parameter C_{lev}.

The Weibull-distributed amplitudes are then applied to the binary mask, and a noise-floor is added

$$\mathbf{W}^{(2)} = \Big(\mathbf{B} \odot \mathbf{C}^{(1)}\Big) + n_{\text{lev}}\mathbf{I}_{(N_u \times N_r)}, \tag{3.9}$$

where $n_{\text{lev}} > 0$ denotes a scaling parameter representing a general uniform noise-floor in the clutter map, and $\mathbf{I}_{(N_u \times N_r)}$ is the identity matrix. The $\odot$ denotes the Hadamard product, also known as element-wise product.

A rectangular guard region centered around the target cell at reference angle $u_0 = \sin(\theta_0)$ is introduced, in this case always set with $u_0 = 0$, within which all elements of the clutter map are set to zero. Let

$$u_k = \frac{2k}{N_u} - 1, \qquad k = 0, 1, \ldots, N_u - 1 \tag{3.10}$$

be the normalized angular coordinate. The column index closest to the desired reference angle is

$$c_0 = \arg\min_k \left| u_k - u_0 \right| \tag{3.11}$$

where arg min denotes the argument of the minimum value. The half-width of the guard region in range and angle is set as

$$g_r = \left\lceil K_g N_r \right\rceil, \qquad g_u = \left\lceil K_g N_u \right\rceil, \tag{3.12}$$

where the parameter $K_g \in (0, 1)$ controls the relative size of the guard region in proportion to the size of the whole clutter map, and $\lceil \cdot \rceil$ denotes the ceiling function. Finally the guard region is zeroed:

$$\mathbf{W}^{(2)}\left[r_0 - g_r : r_0 + g_r, c_0 - g_u : c_0 + g_u \right] = 0, \tag{3.13}$$

where $r_c = \lfloor N_r/2 \rfloor$, centering the guard region in the range-dimension of the clutter map. This leads to a final synthetic clutter map

$$\boxed{\mathbf{CM} = \mathbf{W}^{(2)}} \in \mathbb{R}^{N_u \times N_r}. \tag{3.14}$$

In table 3.1, a summary of all parameters that determine the clutter map generation can be seen, together with brief descriptions and their typical values or value ranges. Some examples of clutter maps generated using this method can be seen in Figure 3.2.

Table 3.1: Parameters of the clutter map generation pipeline, their typical values/ranges and brief descriptions.

Parameter	Typical value/range	Description
N_u	$10 \sim 62$	Number of angular bins (columns) of the clutter map.
N_r	$2N_b - 1$	Number of range bins (rows) of the clutter map. Always set in relation to N_b, number of fast-time samples of the given waveform.
K_f	$0.2 \sim 3$	Determines the size of the 2D triangular filter used to smoothen initial spatial distribution of clutter.
α	$0.9 \sim 1.0$	Exponential parameter applied to filter, determining spatial distribution of clutter. Smaller values concentrate energy toward the center of the map.
W_{thr}	~ 0.97	Quantile determining the proportion of a clutter map that is covered by clutter.
β	~ 1.0	Shape parameter of the Weibull distribution that determines the spread of amplitudes.
C_{lev}	$\sim 10^4$	Linear scaling factor that sets the mean of the Weibull-distributed clutter amplitudes.
n_{lev}	~ 0.1	Constant noise floor added to every pixel after masking.
K_g	~ 0.05	Relative size of the rectangular guard-band around the target cell.
u_0	0 (default)	Reference angle where the target is placed. The guard-band is centred at the column whose angular coordinate is closest to u_0.

3.1.2 Overview of Generated Datasets

A varied collection of different synthetic datasets have been generated for training and evaluation in different settings. This subsection introduces said datasets, the parameter values used to generate them and brief descriptions. The datasets span over three different categories of waveform dimensions, $(M = 2, N_b = 5)$, $(M = 4, N_b = 15)$ and $(M = 12, N_b = 30)$ and the datasets will be presented according to their dimensions. For a statistical overview of the clutter map datasets, see Appendix A.

$\mathbf{M2 - N_b5}$

This is the smallest dimension of waveform used, and the datasets in $(M = 2, N_b = 5)$ are mainly used in experiments and analyses investigating the performance over varying dimensionality, and in a comparison between a supervised and self-supervised

training regime. All datasets in this category share a set of parameter settings, which can be seen in Table 3.2.

Table 3.2: Shared parameters for all generated datasets with waveform dimension ($M = 2, N_b = 5$). The parameters α and K_f that vary between different dataset are denoted with a "*" instead of parameter value.

Parameter	Value
N_u	10
N_r	9
K_f	*
α	*
W_{thr}	0.97
β	1.0
C_{lev}	10^4 (40 dB)
n_{lev}	0.1 (-10 dB)
K_g	0.05
u_0	0.0

The specific datasets are detailed in Table 3.3, where they are named together with specification of dataset size and parameter values for α and K_f.

Table 3.3: Clutter map datasets for the waveform dimension ($M = 2, N_b = 5$) presented with name, total size of dataset and specific parameters (excluding the set of fixed parameters for all datasets of this dimension).

Dataset name	Total size	Parameter settings
`65K-DS-M2xNb5`	65 000	$\alpha = 1$, $K_f = 0.85$ (placeholder)
`1CM-DS-M2xNb5`	400 000	$\alpha = 0.915$, $K_f = 1.7$

M4 – N_b15

This is an intermediate dataset regarding the dimensionality of waveform used, and the dataset in ($M = 4, N_b = 15$) is mainly used in experiments and analyses investigating the performance over varying dimensionality. There is a single dataset in this category called `1CM-DS-M4xNb15` that is created by the set of parameter settings seen in Table 3.4.

Table 3.4: Parameters for the generated dataset with waveform dimension ($M = 4, N_b = 15$).

Parameter	Value
N_u	31
N_r	29
K_f	3
α	0.98
W_{thr}	0.97
β	1.0
C_{lev}	10^4 (40 dB)
n_{lev}	0.1 (-10 dB)
K_g	0.05
u_0	0.0

M12 – N_b30

This is the largest dimension of waveform used, and the datasets in ($M = 12, N_b = 30$) are mainly used in experiments and analyses investigating the performance over varying dimensionality, and in a clutter map difficulty and generalization comparison between the type of data that the model has been trained on versus testing on other datasets with increasing or decreasing difficulty. The difficulty of the dataset is defined by the median number of clutter regions m_{CM} in the clutter map, the median number (m_{CM}) of clutter regions for each dataset is displayed first in the dataset name in the format "m_{CM}`CM-DS-M12xNb30`". Here it should be noted that the median number of clutter regions does not determine the total area of the clutter map covered with clutter, only the distribution of the clutter in terms of cohesive regions. All datasets in this category share a set of parameter settings, which can be seen in Table 3.5.

The specific datasets are detailed in Table 3.6, where they are named together with specification of dataset size and parameter values for α and K_f.
Using the stated parameter settings for clutter map generation, the process generated the following examples of clutter maps, which were extracted from the six datasets described in Table 3.6 and cover different values of m_{CM}.

Table 3.5: Shared parameters for all generated datasets with waveform dimension ($M = 12, N_b = 30$). The parameters α and K_f that vary between different dataset are denoted with a "*" instead of parameter value.

Parameter	Value
N_u	62
N_r	59
K_f	*
α	*
W_{thr}	0.97
β	1.0
C_{lev}	10^4 (40 dB)
n_{lev}	0.1 (-10 dB)
K_g	0.05
u_0	0.0

Table 3.6: Clutter map datasets for the waveform dimension ($M = 12, N_b = 30$) presented with name, total size of dataset and specific parameters (excluding the set of fixed parameters for all datasets of this dimension).

Dataset name	Total size	Parameter settings
`1CM-DS-M12xNb30`	400 000	$\alpha = 0.99$, $K_f = 3$
`2CM-DS-M12xNb30`	400 000	$\alpha = 0.96$, $K_f = 0.46$
`3CM-DS-M12xNb30`	400 000	$\alpha = 0.95$, $K_f = 0.35$
`4CM-DS-M12xNb30`	400 000	$\alpha = 0.93$, $K_f = 0.28$
`5CM-DS-M12xNb30`	400 000	$\alpha = 0.91$, $K_f = 0.23$
`6CM-DS-M12xNb30`	400 000	$\alpha = 0.91$, $K_f = 0.21$

3.1.3 Optimized benchmark data

In addition to the synthetically generated data described above, Saab AB supplied a number of datasets that contain pairs of clutter maps and their corresponding optimized waveforms. All clutter maps were generated using the same algorithm as described in Subsection 3.1.1.

The dataset labeled `CM-DS-M2xNb5-1` is the only one that includes the paired optimized waveforms for training purposes. It is therefore used in the experiment where we compare a supervised learning regime (which requires labeled clutter map - waveform pairs) with the self-supervised regime employed for the majority of the

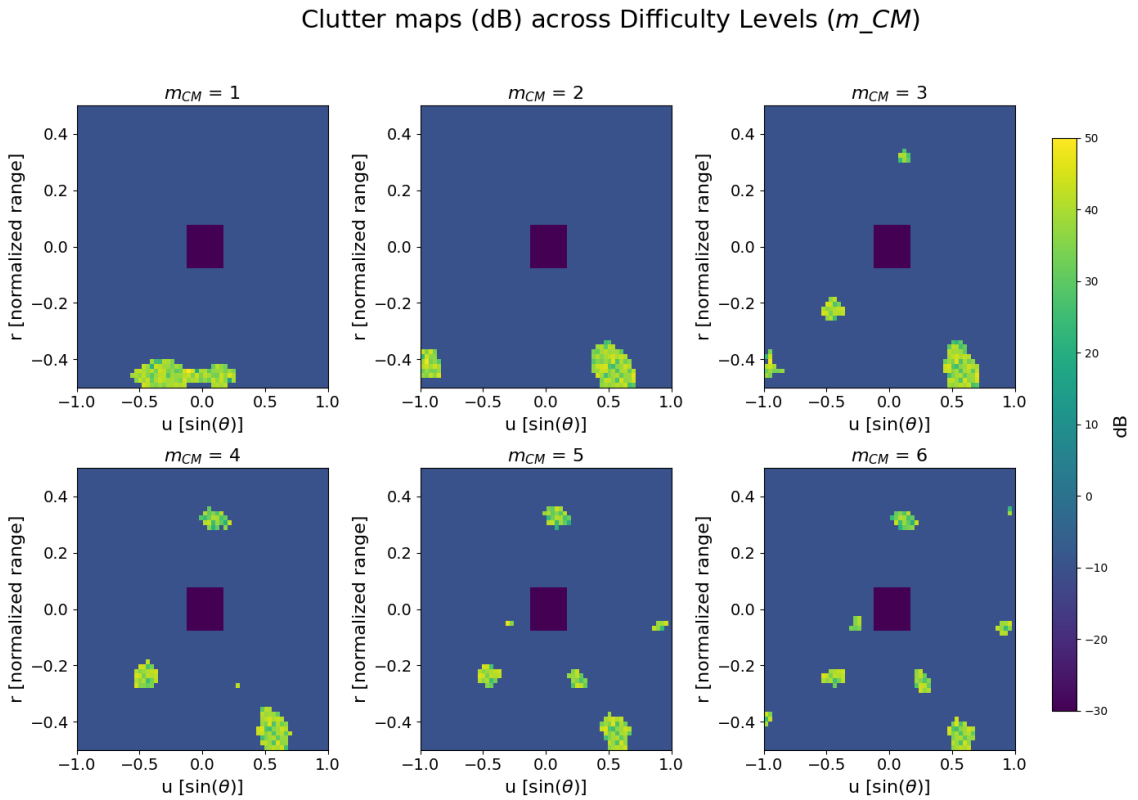


Figure 3.2: Clutter maps (in dB) across six difficulty levels ($m_{CM} = 1, \ldots, 6$). The difficulty level increases from left to right, top to bottom. The y-axis shows normalized range (r) and the x-axis shows normalized angular unit (u).

work. This experiment is detailed in 3.4.1.

All remaining synthetic datasets listed in Subsection 3.1.2 were generated without any associated optimized waveforms. For these, Saab AB provided additional test sets that consist of 1000 samples of clutter map - optimized waveform pairs. These test sets are used exclusively for evaluation and for benchmarking the neural-network approaches against the optimization-based baseline. The optimization used the same waveform design objective, formalized by the F-value described in Eq. (3.21), which is a bandwidth-regularized version of the signal-to-clutter-noise ratio (SCNR).

3.1.4 Data pre-processing

This method of generating clutter maps results in a clutter amplitude distribution with a long tail of high amplitude values, which can be seen in Table A.1. These raw clutter maps were used directly to evaluate the waveform matrices without any pre-processing. The evaluation of waveform matrices is explained in more detail below.

However, a large range and spread of possible input values is something that can cause issues when training machine learning models. Both exploding gradients and

difficulties to generalize across large shifts in the input. In light of this, the clutter maps were normalized by taking their corresponding $\log_{10}$-value as input to the models. Since the guard area is designed to be 0.0, the values of the guard area were replaced by 0.001 prior to the logarithm transform. This results in normalized clutter maps with clutter amplitudes ranging between $\approx [-3, 5.2]$.

3.2 Loss function design and evaluation metrics

This section presents the design of loss function(s) used for the training of neural networks and presents the specific evaluation metrics used to analyze and compare model performances. Both the loss function design and evaluation metrics build upon the signal model and theory presented in Chapter 2.

3.2.1 Loss function

The loss function determines what the neural network is optimized to learn during training, and as such the loss function design is very important in order to achieve the sought behavior. Due to the different experimental setups, three different loss functions are used, which are presented in the following subsections.

3.2.1.1 Bandwidth-regularizing term

As to control the effective bandwidth and the spectral contents of the transmitted waveforms, we introduce a soft constraint limiting the instantaneous phase-increments of the waveforms $\mathbf{\Phi}$. Using the wrapped phase increment introduced in Section 2.1.3.5 and Equation (2.23), we can formulate a function to penalize excessive phase increments. Setting a tolerance $D_{max} = 2.0\,\text{rad}$, the deviation from the tolerance is measured by

$$d_{m,n} = \Big[|\Delta\Phi_{m,n}| - D_{max} \Big]_+, \tag{3.15}$$

where we have $[\,\cdot\,]_+ \triangleq \max\{0, x\}$. It should be noted that the value of $D_{max} = 2.0\,\text{rad}$ is set somewhat arbitrarily and other values could be used, but no variations of D_{max} are explored throughout this report. Using this measure of deviation, we formulate the bandwidth-regularizing function

$$J_{\text{bw}}(\mathbf{\Phi}) = \frac{1}{M(N_b - 1)} \sum_{m=1}^{M} \sum_{n=1}^{N_b - 1} \Big[1 + d_{m,n} \Big]^p \tag{3.16}$$

where the added scalar 1 is included to ensure $J_{\text{bw}}(\mathbf{\Phi}) \geq 1$, making it suitable to incorporate as a multiplicative factor in the SCNR-based loss function presented in Section 3.2.1.3. The exponent p is used as a design parameter determining the severity of J_{bw}, which throughout this thesis is fixed to $p = 8$.

3.2.1.2 Mean Squared Error (MSE)

The Mean Squared Error (MSE) is one of the most commonly used loss functions in deep learning [46]. It is defined by taking the average of the squared differences

between a models predictions and target values, or true values. It relies on having labeled dataset of input and target output pairs, falling into the category of supervised learning. Let $\hat{\mathbf{\Phi}}^{(i)} \in \mathbb{R}^{M \times N_b}$ denote the models predicted output for the i-th sample and $\mathbf{\Phi}^{(i)} \in \mathbb{R}^{M \times N_b}$ denote the corresponding target output. Comparing the phase values directly leads to a loss that doesn't account for their circular nature. To address this, the MSE is calculated using their corresponding waveform matrices $\mathbf{S}_i = e^{\mathbf{j}\mathbf{\Phi}^{(i)}}$ and $\hat{\mathbf{S}}^{(i)} = e^{\mathbf{j}\hat{\mathbf{\Phi}}^{(i)}}$, more specifically

$$\text{MSE} = \frac{1}{M\,N_b} \left\| \mathbf{S}^{(i)} - \hat{\mathbf{S}}^{(i)} \right\|_F^2 = \frac{1}{M\,N_b} \sum_{m=1}^{M} \sum_{n=1}^{N_b} \left| \mathbf{S}^{(i)}_{(m,n)} - \hat{\mathbf{S}}^{(i)}_{(m,n)} \right|^2 \tag{3.17}$$

for a single sample i. To enforce the bandwidth constraint, the MSE-based loss function includes an addition of the previously formulated bandwidth-regularizing function $J_{\text{bw}}(\mathbf{\Phi})$:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{M\,N_b} \left\| \mathbf{S}^{(i)} - \hat{\mathbf{S}}^{(i)} \right\|_F^2 + J_{\text{bw}}(\mathbf{\Phi}) \tag{3.18}$$

When the loss function is evaluated over a batch of samples, it is taken as the mean loss value over the batch. The MSE loss is differentiable both with respect to the output predictions and the model parameters [46].

3.2.1.3 SCNR-based loss function

In the context of informed self-supervised learning, the loss function is defined without need for labeled data, i.e. pairs of input and target outputs. Instead, the loss function can be defined solely using the predicted output of the model, in this case in combination with the model input.

Since the waveform design objective is to maximize signal-to-clutter-noise ratio (SCNR), the loss function can be formulated based on this. We recall the formulation of SCNR from Section 2.1.3.4

$$\text{SCNR} = \frac{|\mathbf{a}(\theta_\mathbf{0})^H \mathbf{S}\mathbf{S}^H \mathbf{a}(\theta_\mathbf{0})|^2}{\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} \mathbf{CM}_{i,j} \, |\mathbf{AF}_{i,j}|^2}. \tag{3.19}$$

where we exclude additive gaussian noise on the receiving end and instead only include the signal dependent noise and clutter modeled by the clutter map $\mathbf{CM}$. Since this function encompasses the primary waveform design objective, and is a function directly evaluating the models output waveforms $\mathbf{\Phi}$, it is well-suited to include in a loss function.

Including the bandwidth-regularizing function J_{bw}, we can now formulate an objective function, first in linear units, as

$$F_{\text{lin}} = \frac{|\mathbf{a}(\theta_\mathbf{0})^H \mathbf{S}\mathbf{S}^H \mathbf{a}(\theta_\mathbf{0})|^2}{\sum_{i=1}^{N_u} \sum_{j=1}^{N_r} \mathbf{CM}_{i,j} \, |\mathbf{AF}_{i,j}|^2} \frac{1}{J_{\text{bw}}(\Phi)}. \tag{3.20}$$

Because performance metrics within radar and signal processing research is more commonly defined in terms of decibels, we also define the dB-scaled objective function

$$F = F_{\text{dB}} = 10\log_{10}(F_{\text{lin}}), \tag{3.21}$$

and for the remainder of the thesis the symbol F will denote the dB-scaled objective function. The F-value is an objective function to maximize, while loss functions are defined as functions to minimize. Thereby the loss function is formulated as

$$\mathcal{L}_{\text{SCNR}} = C_{\mathcal{L}} - F \tag{3.22}$$

where the constant $C_{\mathcal{L}}$ is added to ensure $\mathcal{L}_{\text{SCNR}} > 0$, in this case set to $C_{\mathcal{L}} = 100$. During training, evaluation of the loss function is done over a batch of samples, which is done by taking the average loss value for all samples within a batch. This loss function is now a bandwidth-regularized metric of waveform performance in terms of SCNR, and is used throughout the thesis for informed self-supervised learning where models are trained to generate a single waveform per clutter map. It falls under the category of informed machine learning since we use prior information about the clutter map and evaluate the waveforms using a verifiable metric based on signal processing theory.

3.2.1.4 SCNR-based loss function for diverse outputs

The loss function defined in the previous subsection is constructed for settings where models generate a single waveform for each clutter map. Consequently, it does not include any term that encourages the models to generate a diverse set of waveforms when they generate multiple outputs for the each individual clutter map.

In order to capture the properties of the actual transmitted waveforms, we base the diversity-promoting loss term based on the far-field signal. Let $\{\mathbf{\Phi}^{(k)}\}_{k=1}^{K}$ denote a set of K phase matrices with corresponding waveforms $\{\mathbf{S}^{(k)}\}_{k=1}^{K}$. For the k-th waveform, the far-field representation is denoted by $\tilde{\mathbf{s}}_{\mathbf{E}}^{(k)}$ as presented in Eq. (2.31), normalized over angle bins u to remove the influence of absolute signal power leaving only the directional shape of the angular spectrum.

From Section 2.1.3.6, we recall the similarity metric

$$\text{sim}[i,j] = \frac{1}{N_u}\sum_{u=1}^{N_u}\Big|\sum_{n=1}^{N_b}\tilde{\mathbf{s}}_{\mathbf{E}}^{(i)}[u,n]\tilde{\mathbf{s}}_{\mathbf{E}}^{(j)H}[u,n]\Big| \in [0,1] \tag{3.23}$$

to measure similarity of two waveforms i and j based on their angular transmission spectrum. To penalize similarity across all distinct pairs within the set $\{\mathbf{\Phi}^{(k)}\}_{k=1}^{K}$, we introduce the mean pair-wise similarity

$$J_{\text{div}}\Big(\{\mathbf{\Phi}^{(k)}\}_{k=1}^{K}\Big) = \frac{1}{K(K-1)}\sum_{\substack{k,\ell=1\\ k\neq\ell}}^{K}\text{sim}[k,\ell] \tag{3.24}$$

which is bounded by $0 \leq J_{\text{div}}\left(\{\mathbf{\Phi}^{(k)}\}_{k=1}^{K}\right) \leq 1$. Small values correspond to a set of waveforms producing a diverse set of far-field angular spectra, whereas large values indicate that the generated waveforms collapse to a largely similar spectra.

The final loss function combines the SCNR-based loss function in Eq. (3.22) with the diversity-promoting term. A weighting coefficient $\lambda_{\text{div}} > 0$ controls the relative importance of diversity, leading to the loss function:

$$\mathcal{L}_{\text{SCNR+div}} = \mathcal{L}_{\text{SCNR}} + \lambda_{\text{div}} J_{\text{div}}\left(\{\Phi^{(k)}\}_{k=1}^{K}\right). \tag{3.25}$$

For a single-output model ($K = 1$), the diversity term vanishes and the loss reduces to the original $\mathcal{L}_{\text{SCNR}}$. Again, when evaluating the loss function over a batch of samples, the loss is taken as the mean value over the batch.

3.2.2 Evaluation metrics

This subsection outlines which metrics and performance indicators are used to evaluate the model performance throughout the different experiment setups. The metrics that have already been defined in previous sections will be briefly referred to, while any additional metrics will be described more thoroughly.

3.2.2.1 General waveform performance metrics

The primary waveform design objective is to maximize SCNR, described in Eq. (2.21), and as such it is an important performance metric for evaluating waveforms. However, given the constraint on instantaneous bandwidth of the waveforms, we instead primarily evaluate waveforms using the F-value as defined in Eq. (3.21). The F-value includes a bandwidth regularizing term, and for the most part the difference between F-value and SCNR is relatively small after model training.

Although it doesn't measure performance in and of itself, the ambiguity function presents important insights about the signal returns, and in order to maximize the F-value (and SCNR) the ambiguity function needs to have low values in regions of clutter and a high value at the target location. We use the ambiguity function as defined in Eq. (2.14), but normalized in terms of it's theoretical maximum:

$$\widetilde{AF}(u, \tau, u_0) = \frac{a^H(u) S_\tau S^H a(u_0)}{M^2 N_b}. \tag{3.26}$$

In effort to obtain a more feasible representation, it's visualized in terms of dB, and absolute value, $20 \log_{10}(|\widetilde{AF}(u, \tau, u_0)|)$. Visualized next to the clutter map, the ambiguity function can provide visual information on wether the spatial returns matches the clutter distribution with lower returns from clutter regions and strong signal returns from the target.

Aside from just the ambiguity function itself, it is also of interest to evaluate waveform performance in terms of returned target energy and clutter energy, which are

the two components making up the SCNR-value. Taking the normalized ambiguity function in Eq. (3.26), we have target gain after matched filtering given by

$$\widetilde{G}(u_0) = |\widetilde{AF}(u = u_0, \tau = 0, u_0)|^2 = \left| \frac{a^H(u_0) S S^H a(u_0)}{M^2 N_b} \right|^2. \tag{3.27}$$

The corresponding clutter gain is given by the WISL described in Eq. (2.16), where we again use a normalized ambiguity function for consistency

$$\widetilde{\text{WISL}} = \langle \mathbf{CM}, |\widetilde{\mathbf{AF}}|^2 \rangle_F = \sum_{i=1}^{N_u} \sum_{j=1}^{N_r} CM_{i,j} \, |\widetilde{AF}_{i,j}|^2. \tag{3.28}$$

3.2.2.2 Waveform diversity metrics & Vendi Score

To evaluate diversity between sets of different waveforms, one of the main indicators is the similarity metric $\text{sim}[i, j]$ introduced in Eq. (2.32). This follows naturally, since it is the basis for the diversity-promoting term in the loss function that the neural networks are trained on. Since it measures pair-wise similarity, we can visualize and analyze a similarity-matrix for a given set of $K > 1$ waveforms, but it does not measure overall diversity by itself. One way to measure overall diversity is to simply evaluate the average pair-wise similarity $J_{\text{div}}\left(\{\Phi^{(k)}\}_{k=1}^{K}\right)$ defined in Eq. (3.24), or further averaging over a batch of N different sets of K waveforms

$$\overline{J}_{\text{div}} = \frac{1}{N} \sum_{i=1}^{N} J_{\text{div},i}. \tag{3.29}$$

Still, this metric is an average of pair-wise comparison, and can potentially miss important information about how diverse the set is as a whole, which motivates the need for further complimentary metrics such as the Vendi Score (VS).

Vendi Score: To capture the effective diversity of sets of waveforms we use the Vendi Score [47]. It is based on the concept of effective rank [48], which is a real-valued extension of the common matrix rank, giving an effective measure of dimensionality of a matrix. The Vendi Score is defined as the exponential of the Shannon entropy of the eigenvalue distribution of a normalized similarity matrix, for a given positive semidefinite similarity function with self-similarity equal to 1. Again using the similarity function $\text{sim}[i, j]$ in Eq. (2.32), we define the similarity matrix

$$(\mathbf{K})_{i,j} = \text{sim}[i, j], \qquad \widetilde{\mathbf{K}} = \frac{1}{K}\mathbf{K} \tag{3.30}$$

where K denotes the number of samples (making $\mathbf{K} \in \mathbb{R}^{K \times K}$) and $\widetilde{\mathbf{K}}$ denotes the normalized similarity matrix, such that the trace $\text{Tr}(\widetilde{\mathbf{K}}) = 1$. Let $\lambda_1, \ldots, \lambda_K$ denote the eigenvalues of $\widetilde{\mathbf{K}}$, the Vendi Score is then defined as

$$\text{VS}_{\text{sim}} = \exp\left(-\sum_{k=1}^{K} \lambda_k \log \lambda_k \right) \tag{3.31}$$

where the convention $0 \log 0 = 0$ is used [47]. This formulation is in practice the same as the effective rank of the similarity matrix $\mathbf{K}$, but will be referred to as the Vendi Score throughout the report. One can note that the Vendi Score can be calculated using any other similarity function that satisfy the constraints of being positive semidefinite with self-similarity equal to one, and other measures of similarity could be used to calculate the VS [47]. The VS can be calculated both for a set of waveforms generated for the same clutter map, and also over a set of waveforms corresponding to different clutter maps.

The Vendi Score can be understood as an effective number of dissimilar elements of a given sample set, providing a consistent measure of diversity that can be compared using ratios and percentages [47]. Given this, it can be interpreted such that a sample set with Vendi Score m is as diverse as a sample set consisting of m elements that are completely dissimilar according to the chosen similarity function, in this case corresponding to completely orthogonal far-field signals in every angle bin. As shown in [47], it accounts for correlation between features and can distinguish between sets that are uniformly diverse and sets that contain a few outliers and many nearly identical samples, making it a suitable measure of diversity for the application at hand.

3.3 Model Implementations

The following sections describe the thoughts behind and implementation details for the evaluated models. Throughout the model architecture tables below, B denotes the *batch size*.

3.3.1 Deterministic Single-pass Models

The deterministic single-pass models were designed to use a clutter map $\mathbf{CM}$ as the input and predict a fixed number, K, of waveform matrices $\hat{\mathbf{\Phi}}$ for each input clutter map, resulting in a model that approximates the mapping $\hat{f} : \mathbb{R}^{(N_u \times N_r)} \rightarrow \mathbb{R}^{K \times (M \times N_b)}$. These models are deterministic in the waveform generation process, meaning that given a clutter map, they always return the same fixed number, K, of waveform matrices.

3.3.1.1 Convolutional Autoencoder (CAE)

The convolutional autoencoder's encoder consists of three 2D convolutional layers, with number of filters/output channels according to $[64, 128, 512]$, interleaved with two dimensional batch normalization layers and $ReLU$ activation functions.

The encoder is followed by a 4 layer MLP block using $SiLU$ activation functions that forms a bottleneck, compressing the information extracted from the encoder down to a latent dimension $d_z = 256$ and thereafter expanding the latent representation before entering the decoder block.

The decoder mirrors the architecture of the encoder with interleaved batch normalization and activation functions. While being structurally similar, the decoder block instead consists of two 2D transpose convolutional layers, with number of filters/output channels according to $[128, 64]$, which are used to expand the representation. The final part of the decoder consists of a linear layer that transforms the decoded representation into the specified dimensions of the K generated waveforms. A more detailed view of the model architecture can be seen in Table 3.7.

The parameter settings described above are the ones used for this model architecture during the experiments unless other values are explicitly stated for each experiment.

No.	Type	Input Shape	Specifications
		Encoder	
1	Conv2d	$[B, K, N_u, N_r]$	out channels=64, stride=2
2	BatchNorm2d	$[B, 64, N_u//2, N_r//2]$	-
3	ReLU	$[B, 64, N_u//2, N_r//2]$	-
4	Conv2d	$[B, 64, N_u//2, N_r//2]$	out channels=128, stride=2
5	BatchNorm2d	$[B, 128, N_u//4, N_r//4]$	-
6	ReLU	$[B, 128, N_u//4, N_r//4]$	-
7	Conv2d	$[B, 128, N_u//4, N_r//4]$	out channels=512, stride=1
8	BatchNorm2d	$[B, 512, N_u//4, N_r//4]$	-
9	ReLU	$[B, 512, N_u//4, N_r//4]$	-
10	AdaptiveAvgPool2d	$[B, 512, N_u//4, N_r//4]$	out size=$[4, 4]$
11	Flatten	$[B, 512, 4, 4]$	-
		Bottleneck	
12	Linear	$[B, 8192]$	out size=512
13	SiLU	$[B, 512]$	-
14	Linear	$[B, 512]$	out size=256
15	Linear	$[B, 256]$	out size=512
16	SiLU	$[B, 512]$	-
17	Linear	$[B, 512]$	out size=8192
		Decoder	
18	Reshape	$[B, 8192]$	out shape=$[B, 512, 4, 4]$
19	ConvTranspose2d	$[B, 512, 4, 4]$	out channels=128, stride=2
20	BatchNorm2d	$[B, 128, 8, 8]$	-
21	ReLU	$[B, 128, 8, 8]$	-
22	ConvTranspose2d	$[B, 128, 8, 8]$	out channels=64, stride=2
23	BatchNorm2d	$[B, 64, 16, 16]$	-
24	ReLU	$[B, 64, 16, 16]$	-
25	Flatten	$[B, 64, 16, 16]$	-
26	Linear	$[B, 16384]$	out size=$K \times M \times N_b$
27	Output	$[B, K, M, N_b]$	-

Table 3.7: Architectural details of the convolutional autoencoder model.

3.3.1.2 Vision Transformer (ViT)

The ViT mainly consists of three parts, a the patch embedder, vision transformer blocks and the output heads. The patch embedder splits the input clutter map into $T = \lceil N_u/p \rceil \cdot \lceil N_r/p \rceil$ patches with patch size $p = 4$, embeds each patch to a $d_{model} = 256$ dimensional vector and generates a sequence of T embedded vectors which serve as the representation of the clutter map. The embedded clutter map is thereafter sent through $L = 14$ ViT blocks. Each ViT block uses MHA according to Section 2.2.6.2 configured with $H = 8$ attention heads and a per-head query-key dimension $d_k = \frac{d_{model}}{8} = 32$, together with a two layer MLP using a hidden dimension multiplier $k_{mlp} = 4$ and residual connections.

These blocks progressively adds contexts of increasing abstraction level and complexity before reaching the final latent representation. This representation is passed through K output heads which are built from two layer MLP blocks to form the model output with the shape $[B, K, M, N_b]$. A more detailed view of the ViT model architecture is found in Table 3.8.

The parameter settings described above are the ones used for this model architecture during the experiments unless other values are explicitly stated for each experiment.

No.	Type	Input Shape	Specifications
Patch Embedder			
1	Slice	$[B, K, N_u, N_r]$	out=$[B, N_u, N_r]$
2	Reshape	$[B, N_u, N_r]$	out=$[B, 1, N_r, N_u]$
3	Padding	$[B, 1, N_r, N_u]$	pad=$(N_r + p_h, N_u + p_w)$
4	Unfold	$[B, 1, N_r + pad_h, N_u + pad_w]$	out=$[B, p \times p, T]$
5	Transpose	$[B, p \times p, T]$	out=$[B, T, p \times p]$
6	Linear	$[B, T, p \times p]$	out=$[B, T, d_{model}]$
ViT Block			
1	LayerNorm	$[B, T, d_{model}]$	-
Attention			
2	Linear (QKV)	$[B, T, d_{model}]$	out=$[B, T, 3d_{model}]$
3	Reshape	$[B, T, 3d_{model}]$	out=$[B, T, 3, H, d_k]$
4	Permute	$[B, T, 3, H, d_k]$	out=$[3, B, H, T, d_k]$
5	Split	$[3, B, H, T, d_k]$	-
6	Dot-product (QK)	$2 \times [B, H, T, d_k]$	out=$[B, H, T, T]$
7	Softmax	$[B, H, T, T]$	-
8	Dropout	$[B, H, T, T]$	-
9	MatMul	$[B, H, T, T] \times [B, H, T, d_k]$	out=$[B, H, T, d_k]$
10	Permute	$[B, H, T, d_k]$	out=$[B, T, H, d_k]$
11	Reshape	$[B, T, H, d_k]$	out=$[B, T, d_{model}]$
12	Linear	$[B, T, d_{model}]$	out=$[B, T, d_{model}]$
13	Dropout	$[B, T, d_{model}]$	-
14	Add	$[B, T, d_{model}] + [B, T, d_{model}]$	-
MLP			
15	LayerNorm	$[B, T, d_{model}]$	-
16	Linear	$[B, T, d_{model}]$	out=$[B, T, d_{model} \times k_{mlp}]$
17	GELU	$[B, T, d_{model} \times k_{mlp}]$	-
18	Linear	$[B, T, d_{model} \times k_{mlp}]$	out=$[B, T, d_{model}]$
19	Dropout	$[B, T, d_{model}]$	-
20	Add (Residual)	$[B, T, d_{model}] + [B, T, d_{model}]$	-
Output Head			
1	LayerNorm	$[B, d_{model}]$	-
2	Linear	$[B, d_{model}]$	out=$[B, d_{model} \times k_{head}]$
3	GELU	$[B, d_{model} \times k_{head}]$	-
4	Linear	$[B, d_{model} \times k_{head}]$	out=$[B, K \times M \times N_b]$
5	Reshape	$[B, K \times M \times N_b]$	out=$[B, K, M, N_b]$

Table 3.8: Architectural details of the ViT Transformer.

3.3.2 Iterative refinement models

In contrast to the implementation of the deterministic single-pass models described in Section 3.3.1, the iterative refinement models were designed to start from an ini-

tial sampled waveform matrix $\mathbf{\Phi}_0$ and use the clutter map **CM** to condition each refinement step towards an improved waveform matrix $\mathbf{\Phi}_T$.

Sampled starting point: The random sampled starting point used for the iterative refinement models, denoted $\mathbf{\Phi}_0$, is generated by first computing a cumulative summation of N_b random steps, where each step is in the range $[-D_{max}, D_{max}]$, which forms a single waveform $\boldsymbol{\phi}_0$ vector with dimensionality $\mathbb{R}^{N_b}$. The waveform $\boldsymbol{\phi}_0$ is thereafter copied to all M antenna elements to create the randomly generated starting waveform matrix $\mathbf{\Phi}_0$.

Sensitive refinement: The two iterative refinement models focus on making smaller adjustments to $\mathbf{\Phi}_0$ in multiple steps. The refinement process is implemented deterministically in the residual-based model while the score-based diffusion model introduces stochastically influenced refinement updates.

For the residual-based model described in Section 3.3.2.1, finer adjustments are achieved by limiting each residual block's output using a *Tanh* activation function. This function forces the output to be bounded in $\in (-1, 1)$ which makes it possible for each block to learn smaller adjustments.

For the Score-based Diffusion Model described in Section 3.3.2.2, the finer updates are achieved by limiting the updates of $\mathbf{\Phi}_0$ through the use of an annealing step size which is a function of the iteration index t. The annealing step size ϵ_t is defined through a linear equation as a function of t according to:

$$\epsilon_t = (\beta_{min} + t \cdot (\beta_{max} - \beta_{min})), \quad \beta_{min} = 0.00001, \beta_{max} = 0.5, \quad t : 1 \rightarrow 0 \tag{3.32}$$

The original implementation of the step size according to the Langevin Equation (2.54) uses $\frac{\epsilon_t}{2}$ to scale the gradient direction proposed by the model while the standard deviation of the noise term is scaled by $\sqrt{\epsilon_t}$. This way of handling the trade-off between gradient-based and noisy updates makes the refinement process guided by the model's predicted gradient direction more for initial updates, when the gradients are large, to move the sample towards better regions of the loss landscape. Thereafter, the updates transitions towards smaller updates when the ϵ_t term becomes small. During the later iterations, the noise term aims to become dominant relative to the gradient updates, resulting in a small Brownian motion near a local optima of the loss landscape. This is a result of the linear scaling of the gradient factor compared to the square root scaling for the standard deviation of the noise.

Based on the loss landscape and the sensitivity of the loss function given a waveform matrix $\mathbf{\Phi}_t$, the original Langevin dynamics and gradient to noise tradeoff through ϵ_t were adjusted. In this work, the dynamics are switched from scaling the standard deviation of the noise by $\sqrt{\epsilon_t}$ to instead scale by ϵ_t^2. This means that the amplitude of the added noise becomes smaller relative to the gradient factor for later iterations, which minimizes the risk of disturbing high performing solutions in the later stages of the refinement process. The drawback of this rescaling is that the update rule no

longer preserves the standard Langevin dynamics interpretation, since the relation between the gradient term and noise term is changed. This also makes the updates less stochastic and weakens their ability to explore the solution space or escape local optima.

3.3.2.1 Residual Network (ResNet)

The Residual Network designed and evaluated in this report consists of 2 components, a convolution-based clutter map embedder and an MLP residual block designed to predict $\Delta\mathbf{\Phi}_t$ which describes how $\mathbf{\Phi}_{t-1}$, the waveform after $t-1$ refinement steps, should be further refined under the condition of a specific clutter map **CM**. The clutter map embedder mirrors the architecture for the encoder described in Table 3.7 but with modifications. The residual network's clutter map embedder utilizes two 2D convolutional layers, both with 256 filters. These layers, similarly to the autoencoder, are interleaved with batch normalization and $ReLU$ activation functions. The resulting clutter map embedding with dimensionality $d_{latent} = 256$ is thereafter used as a conditioning vector to the residual block. Each MLP residual block takes a flattened version of the previous block's output $\mathbf{\Phi}_{t-1}$ and concatenates it with the clutter map embedding vector to form its input. This block utilizes a three layer MLP with $GELU$ activation functions and $d_{mlp} = 512$ neurons per layer to predict $\Delta\mathbf{\Phi}_t$. The model starts with a waveform matrix $\mathbf{\Phi}_0$, randomly generated through a process described in Section 3.3.2, and refines $\mathbf{\Phi}_0$ during $L = 14$ steps by passing it through the model's L residual blocks. The skip connection between each block adds the previous block's output $\mathbf{\Phi}_{t-1}$ to the scaled predicted change $\boldsymbol{\alpha}_t \cdot \Delta\mathbf{\Phi}_t$, where each scaling factor at step t throughout all L steps is controlled by the scaling vector $\boldsymbol{\alpha} = [1, 1, \ldots, 1, 1] \in \mathbb{R}^L$. The updates can be described through:

$$\mathbf{\Phi}_t = \mathbf{\Phi}_{t-1} + \boldsymbol{\alpha}_t \cdot \Delta\mathbf{\Phi}_t. \tag{3.33}$$

Since this model structure only modifies the phase values contained in $\mathbf{\Phi}_t$ during multiple steps but keeps the structure and shape intact, there is no need for a final transformation to the correct waveform dimensions. A more detailed table of the network architecture is found in Table 3.9.

The parameter settings described above are the ones used for this model architecture during the experiments unless other values are explicitly stated for each experiment.

No.	Type	Input Shape	Specifications
Clutter map embedder			
1	Input	$[B, K, N_u, N_r]$	Raw waveform input
2	Reshape	$[B, K, N_u, N_r]$	out=$[BK, 1, N_u, N_r]$
3	Conv2d	$[BK, 1, N_u, N_r]$	out=$[BK, 256, N_u, N_r]$
4	BatchNorm2d	$[BK, 256, N_u, N_r]$	-
5	ReLU	$[BK, 256, N_u, N_r]$	-
6	Conv2d	$[BK, 256, N_u, N_r]$	out=$[BK, 256, N_u, N_r]$
7	BatchNorm2d	$[BK, 256, N_u, N_r]$	-
8	ReLU	$[BK, 256, N_u, N_r]$	-
9	AdaptivePool	$[BK, 256, N_u, N_r]$	out=$[BK, 256, M, N_b]$
10	Flatten	$[BK, 256, M, N_b]$	out=$[BK, 256 \cdot M \cdot N_b]$
11	Linear	$[BK, 256MN_b]$	out=$[BK, d_{latent}]$
MLP ResNet Block			
13	Reshape	$[BK, 1, M, N_b]$	out=$[BK, M \cdot N_b]$
14	Concat	$[BK, MN_b] + [BK, d_{latent}]$	out=$[BK, MN_b + d_{latent}]$
15	Linear	$[BK, MN_b + d_{latent}]$	out=$[BK, d_{mlp}]$
16	GELU	$[BK, d_{mlp}]$	-
17	Linear	$[BK, d_{mlp}]$	out=$[BK, d_{mlp}]$
18	GELU	$[BK, d_{mlp}]$	-
19	Linear	$[BK, d_{mlp}]$	out=$[BK, M \cdot N_b]$
20	Tanh	$[BK, M \cdot N_b]$	-
21	Reshape	$[BK, M \cdot N_b]$	out=$[BK, 1, M, N_b]$
22	Add (Residual)	$[BK, 1, M, N_b] + \boldsymbol{\alpha}_t \cdot [BK, 1, M, N_b]$	$\Phi_{out} = \Phi_{in} + \boldsymbol{\alpha}_t \cdot \Delta\Phi$
Output Head			
23	Reshape	$[BK, 1, M, N_b]$	out=$[B, K, M, N_b]$

Table 3.9: Architectural details of the Residual Network model.

3.3.2.2 Score-based Diffusion Model

The Score-based Diffusion model is built from three components, a time embedder, a clutter map embedder and a score network. Similarly to the Residual Network, this diffusion model starts from a randomly sampled $\boldsymbol{\Phi}_0$, described in Section 3.3.2. The score based diffusion model iteratively updates the waveform $\boldsymbol{\Phi}_0$ during 50 steps.

For each step, the model uses the iteration index t embedded using fourier features, to generate a learnable time embedding vector $\mathbf{e}_t$ of dimensionality $d_t = 256$. The time embedder works by mapping the scalar time index t to a fourier features vector $\mathbf{e}_{raw}$ of dimensionality $d_{freq} = 128$ through:

$$\mathbf{f} = \exp\left(-\frac{i \cdot \log(10000)}{F-1}\right), \quad i \in \{0, 1, \ldots, F-1\}, \tag{3.34}$$

then,

$$\mathbf{e}_{raw} = \text{concat}\left[\sin(t \cdot \mathbf{f}), \cos(t \cdot \mathbf{f})\right]. \tag{3.35}$$

The fourier features $\mathbf{e}_{raw} \in \mathbb{R}^{2F}$, where $F = 64$, are thereafter passed through a two layer MLP with *SiLU* activation to arrive at the time embedding vector $\mathbf{e}_t \in \mathbb{R}^{d_t}$. This time embedding vector $\mathbf{e}_t$ serves as the model's notion of time and where in the refinement process the current gradient prediction takes place.

The clutter map, **CM**, is used as the input to the clutter map embedder, which embedds it into a vector representation of dimensionality $d_{CM} = 256$ by utilizing two 2D convolutional layers with 64 and 128 filters. These layers are interleaved with batch normalization and *SiLU* activation functions and followed by a linear transformation to produce the final embeddeding.

In each step of the update process, the previous refinement step's waveform $\mathbf{\Phi}_{t-1}$, current time embedding $\mathbf{e}_t$ and clutter map embedding are concatenated and fed into a three layer MLP module where the hidden layers of the MLP has a size of $d_{mlp} = 1024$. The MLP module projects the information down to the dimensions of the waveform matrix, which becomes the model's prediction for the gradient of the loss landscape at the current $\mathbf{\Phi}_{t-1}$ point. This predicted gradient, $f(\mathbf{\Phi}_{t-1}, t, \mathbf{CM})$, where t is the time index and **CM** represents the clutter map, is used to compute the refined waveform matrix $\mathbf{\Phi}_t$ using the updated Langevin equation described in Section 3.3.2 through:

$$\mathbf{\Phi}_t = \mathbf{\Phi}_{t-1} + \frac{\epsilon_t}{2} f(\mathbf{\Phi}_{t-1}, t, \mathbf{CM}_i) + \epsilon_t^2 z_t, \tag{3.36}$$

using the definition of the stepsize ϵ_t according to Equation (3.32) with $\beta_{min} = 0.00001, \beta_{max} = 0.5$.

The parameter settings described above are the ones used for this model architecture during the experiments unless other values are explicitly stated for each experiment.

No.	Type	Input Shape	Specifications
Time Embedder			
1	Input t	$[B]$	Diffusion time step
2	Mul (Freqs)	$[B]$	$t \times$ freqs
3	Cat	$[B, 2F]$	$[\sin, \cos]$
4	Linear	$[B, 2F]$	out=$[B, d_t]$
5	SiLU	$[B, d_t]$	-
6	Linear	$[B, d_t]$	out=$[B, d_t]$
7	Expand	$[B, d_t]$	out=$[B, K, d_t]$
Clutter map embedder			
1	Input *img*	$[B, K, N_u, N_r]$	-
2	Reshape	$[B, K, N_u, N_r]$	out=$[BK, 1, N_u, N_r]$
3	Conv2d	$[BK, 1, N_u, N_r]$	out=$[BK, 64, N_u/2, N_r/2]$
4	BatchNorm2d	$[BK, 64, N_u/2, N_r/2]$	-
5	SiLU	$[BK, 64, N_u/2, N_r/2]$	-
6	Conv2d	$[BK, 64, N_u/2, N_r/2]$	out=$[BK, 128, N_u/4, N_r/4]$
7	BatchNorm2d	$[BK, 128, N_u/4, N_r/4]$	-
8	SiLU	$[BK, 128, N_u/4, N_r/4]$	-
9	AdaptivePool	$[BK, 128, N_u/4, N_r/4]$	out=$[BK, 128, 16, 16]$
10	Flatten	$[BK, 128, 16, 16]$	-
11	Linear	$[BK, 32768]$	out=$[BK, d_{CM}]$
12	BatchNorm1d	$[BK, d_{CM}]$	-
13	SiLU	$[BK, d_{CM}]$	-
Score Network (Core MLP)			
1	Input x	$[B, K, M, N_b]$	Noisy tensor
2	Reshape x	$[B, K, M, N_b]$	out=$[BK, M \cdot N_b]$
3	Concat	$[BK, MN_b + d_{CM} + d_t]$	-
4	Linear	$[BK, MN_b + d_{CM} + d_t]$	out=$[BK, 1024]$
5	SiLU	$[BK, 1024]$	-
6	Linear	$[BK, 1024]$	out=$[BK, 1024]$
7	SiLU	$[BK, 1024]$	-
8	Linear	$[BK, 1024]$	out=$[BK, M \cdot N_b]$
Output Head			
1	Reshape	$[BK, M \cdot N_b]$	out=$[B, K, M, N_b]$

Table 3.10: Architectural details of the DiffusionScoreNet.

3.3.3 Training and optimizer configuration

The training of neural networks is, in our case, a process that consists of repeatedly presenting mini-batches of data from a training dataset, evaluating a loss function on each batch and updating the model parameters so as to minimize the expected loss [34]. The loss functions used here are described in Subsection 3.2.1, with differ-

ent formulations for supervised learning, self-supervised learning with single-output and self-supervised learning for multiple outputs. In this subsection, we denote $\mathcal{L}$ as a generic loss function. An optimizer and learning-rate scheduler are responsible for taking the computed gradient information into model parameter updates.

Throughout this report, all neural networks are trained using the `AdamW` optimization algorithm [49] implemented in `PyTorch`, which is a heavily adopted optimizer within the field of deep learning. It is an adaptive gradient algorithm that extends the Adam optimizer described in [50] with decoupled weight decay, showing improved generalization performance [49].

The learning-rate is a parameter that decides the rate of update of the model parameters, recognized as a very important parameter for training of deep neural networks. It can be set to a fixed learning rate or varied throughout the training, via a learning rate schedule. We employ the One-Cycle learning rate scheduler, as proposed in [51], throughout all training runs. This scheduler starts from an initial value and increases monotonically towards a maximum value after a set percent of the total epochs are finished, followed by a cosine annealing schedule towards a smaller final learning rate.

Training algorithm: The training algorithm is initialized with specified hyperparameters and a model with randomly intialized weights, and iterates for the given number of epochs. The dataset used is split into a training dataset and validation dataset, where the training dataset is used to update model weights and the validation set is used to evaluate performance at the end of each epoch. For every epoch, the learning rate is updated according to the learning rate scheduler. The training process is detailed in Algorithm 1. Note that for the Diffusion Model, backpropagation is performed for every timestep in the denoising process.

Algorithm 1 General training loop for regular and diffusion models

Initialise model parameters $\boldsymbol{\theta}$
Initialise optimiser $\mathcal{O} \leftarrow$ `AdamW`(θ)
Initialise scheduler $\mathcal{S} \leftarrow$ `OneCycleLR`$(\mathcal{O})$
Split dataset into training and validation sets
for epoch $e = 1$ **to** E **do**
    Set model to training mode
    for mini-batch $\mathcal{B}$ in training data **do**
        Move $\mathcal{B}$ to device
        $\mathcal{O}$.`zero_grad`()
        if model **is not** Diffusion Model **then**
            Compute prediction $\hat{\mathbf{\Phi}} = \hat{f}_{\boldsymbol{\theta}}(\mathcal{B})$
            Compute loss $L = \mathcal{L}(\hat{\mathbf{\Phi}}, \mathcal{B})$
            Backpropagate L
        else
            Sample initial noisy state $\mathbf{\Phi}_0$
            for diffusion step $t = 1$ **up to** T **do**
                Update $\hat{\mathbf{\Phi}}_t = \hat{f}_{\boldsymbol{\theta}}(t, \hat{\mathbf{\Phi}}_{t-1}, \mathcal{B})$
                Compute step loss $L_t = \mathcal{L}(\hat{\mathbf{\Phi}}_t, \mathcal{B})$
                Backpropagate L_t
            end for
        end if
        $\mathcal{O}$.`step`()
        $\mathcal{S}$.`step`()
    end for
    Set model to evaluation mode
    Evaluate validation loss without gradient computation
    if validation loss improves **then**
        Store current model as best model
    end if
    Log training and validation metrics
end for

Reference configuration: The standard configuration of hyperparameters can be seen in Table 3.11, which are used as default throughout the experiments and training runs unless stated otherwise. The parameters are specified according to the corresponding `PyTorch`[52] implementations of AdamW optimizer and One-Cycle learning rate scheduler, more specifically given for `PyTorch` version 2.2.0+cu121.

3.4 Experiment design

This section presents the experiments designed to assess the proposed neural-network based waveform synthesis approach, evaluate and compare the different models and training regimes, and analyze the interesting properties emerging from this approach.

Table 3.11: Standard trainer and optimizer settings.

Trainer / loss parameter	**Value**
Number of epochs E	25
Batch size	256
Optimizer (AdamW)	
Learning rate (max)	2×10^{-4}
Betas (β_1, β_2)	$(0.9, 0.999)$
Epsilon ϵ	10^{-6}
Weight decay λ	0.01
Learning-rate scheduler (OneCycleLR)	
Learning rate (max)	2×10^{-4}
Percentage of cycle for warm-up	0.30
Annealing strategy	cosine
Div factor (initial LR = max LR / div)	25.0
Final div factor (final LR = init LR / finaldiv)	1000.0

3.4.1 Experiment 1: Supervised versus Physics-informed self supervised machine learning

This experiment aims to showcase how a supervised training paradigm compares against a physics-informed self-supervised training schema for training a neural-network based waveform generator, focusing on the model's capability to generate waveform solutions with high F-value according to Equation (3.21).

The dataset used for this experiment is denoted `65K-DS-M2xNb5`, with waveform dimensions set to $M = 2$ transmit antennas and $N_b = 5$ fast-time samples per pulse. This dataset contains 65000 different clutter maps paired with 65000 waveform matrices that were optimized to suppress clutter, which is split into 80% training and 15% validation data, saving 5% of the dataset for testing. The model used during this experiment is the standard version of the Convolutional Autoencoder (deterministic single-pass) described in Section 3.3.1.1 together with the standard configuration of hyperparameters shown in Table 3.11. This experiment is set up to use $K = 1$ and therefore generate one waveform solution per clutter map.

For comparing supervised and physics-informed self-supervised learning, 10 model training runs are performed for each training paradigm, where the model is initialized and data is shuffled using 10 different seeds. For supervised training, the clutter map and optimized waveform matrix pairs are used as the input and target for the model and its predictions. The waveform matrices generated by the model are compared to the target matrices by computing the MSE loss described in Eq. (3.18), and the resulting loss value is used to update and optimize the model parameters.

In contrast, the physics-informed self-supervised learning schema does not require

any target waveform matrix for each input clutter map and therefore only uses the clutter maps from the same dataset that were used for supervised training. This training approach uses the model's generated waveforms and the input clutter maps to compute the loss value through the SCNR-based loss function described in Section 3.2.1.3 instead of through the MSE loss function. The resulting loss value is thereafter used to optimize the model parameters.

3.4.2 Experiment 2: Single-output performance across dimensions

This experiment investigates how the proposed machine learning models perform across the waveform dimensions $\mathbf{M2 - N_b5}, \mathbf{M4 - N_b15}$ and $\mathbf{M12 - N_b30}$, when the models generate one waveform matrix per clutter map. The experiment focuses on the model's capability to generate waveform solutions with high F-values according to Equation (3.21) and how this performance scales across waveform dimensions.

The clutter maps used during this experiment were generated to on average include one clutter region per clutter map. An overview of the clutter region statistics can be seen in Table A.2, Appendix A. The training datasets used are denoted `1CM-DS-M2xNb5`, `1CM-DS-M4xNb15`, `1CM-DS-M12xNb30` for the three investigated waveform dimensions, each containing 400000 clutter maps and undergoes a data split into 80% training data and 20% validation data. The three held-out test datasets used to evaluate this experiment were supplied by SAAB according to Section 3.1.3. Each test dataset consists of 1000 clutter maps and optimized waveform pairs that were generated using the same settings as the training data. All four models described in Section 3.3 are used with their standard settings across all waveform dimensions, configured such that model sizes, both between models and across waveform dimensions, become comparable. The models used during this experiment were set up to use $K = 1$ and therefore generate one waveform solution per cluttermap.

Each training run was performed with the standard training configuration described in Table 3.11, except for the score-based diffusion model which used a learning rate (max) value of 4×10^{-5} to keep the training stable.

3.4.3 Experiment 3: Multi-output performance and diversity

The purpose of this experiment is to assess how well the neural-network based waveform generators can produce multiple candidate waveforms per clutter scenario and how diverse those candidates are. The effects of multiple, diverse outputs on the overall waveform performance is also studied, to examine wether multiple, diverse outputs enhance performance and increases robustness. Since there are two distinct categories of model architectures presented here, the deterministic single-pass models and the stochastic iterative refinement models, they present two separate methods for approaching diversity. With that background, this experiment is con-

structed around evaluating and comparing the Convolutional Autoencoder (deterministic single-pass) and the stochastic Residual Network (stochastic iterative refinement). For the Convolutional Autoencoder, multiple outputs are achieved by simply extending the final layer to consist of K parallel output heads for K waveform outputs, and the stochastic Residual Network achieves multiple outputs by starting the refinement process from K different sampled waveforms, using the sampling process detailed in Subsection 3.3.2.1. In both cases the same underlying training data and loss function are used, enabling a direct comparison between the two different strategies to approach diverse outputs.

All models are trained on the dataset denoted `1CM-DS-M12xNb30`, introduced in Table 3.6, with waveform dimensions set to $M = 12$ transmit antennas and $N_b = 30$ fast-time samples per pulse. The training corpus consists of 400000 clutter map examples, which is split into 80% training data and 20% validation data during the training. The two model architectures are configured and trained for each chosen number of waveform outputs K:

$$K \in \{1, 4, 8, 16, 32\}. \tag{3.37}$$

Since this experiment is centered around multiple, diverse generated waveforms per clutter map, the loss function used here is $\mathcal{L}_{\text{SCNR+div}}$ with a weighting coefficient $\lambda_{\text{div}} = 10$ on the diversity loss-term, described in Eq. 3.25. With regards to compute resource needs and training time, some modifications to the Residual Network model standard configurations were made, as well as small changes to the training configuration. However, these settings do not vary between different training runs within this experiment, still allowing for a direct comparison between the two models. The model parameter settings that deviate from the default presented in Subsection 3.3.2.1 can be seen in Table 3.12. Additionally, the batch size was set to 128 instead of the default 256.

Table 3.12: Residual Network parameter settings for the multi-output experiment, excluding the previously detailed default settings.

Model Parameter	**Value**
Convolutional filter sizes	[128, 128]
Number of residual blocks L	10
MLP hidden dimension d_{mlp}	256

The models have a total parameter count of 15108752 for the Residual Network and 15889192 for the Convolutional Autoencoder for $K = 1$, but since the output head of the CAE scales with K, it varies throughout the experiment.

The results of this experiment is evaluated using the held-out test dataset corresponding to the previously mentioned training dataset `1CM-DS-M12xNb30`, which was provided by Saab AB as described in Subsection 3.1.3. This test set consists of 1000 clutter map examples generated with the same settings as the training dataset, accompanied by corresponding optimized waveforms as a performance benchmark.

3.4.4 Experiment 4: Single-output performance and generalization across clutter map difficulty levels

The aim of this experiment is to analyze the neural-network based waveform generators ability to generalize across clutter maps with different numbers of clutter regions compared to what it was trained on. The distribution of the number of clutter regions contained in a dataset acts as a proxy for the difficulty level. This experiment focuses on the model's capability to generate waveform solutions with high SCNR and F values according to Equation (3.21) and how this performance scales for different numbers of clutter regions contained in the clutter map.
This is tested using six different datasets described in 3.6 for which the median number of clutter regions per clutter map ranges from 1 to 6. Each dataset contains 400000 data points and use the waveform dimensions $M = 12$ antenna elements and $N_b = 30$ fast-time sample points, and a data split into 80% training and 20% validation data. The model used during this experiment is the standard version of the Convolutional Autoencoder model described in Section 3.3.1.1 together with the standard training configuration in Table 3.11. The convolutional autoencoder was configured with $K = 1$ for each of the six training runs to analyze the models ability to generate one waveform solution for each clutter map.

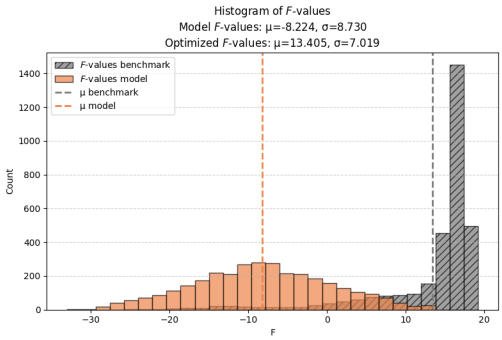
Evaluation of the six trained models for the different difficulty levels were done using six held-out test datasets corresponding to the six different training datasets. These datasets where supplied by Saab AB as described in Subsection 3.1.3. Each generated with the same settings as the training data, resulting in benchmark test datasets with matching spread of clutter regions per clutter map for all six difficulty levels.

4
Results

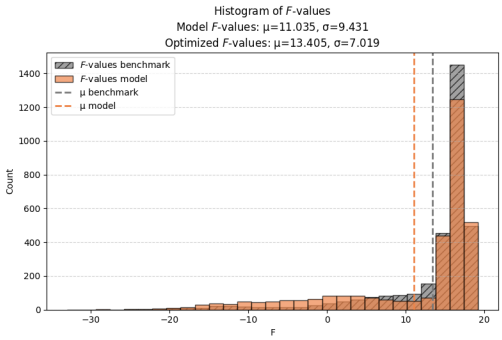
This chapter presents the results obtained from the experiments described in Section 3.4, divided in to subsections presenting results from each experimental setup.

4.1 Experiment 1: Supervised versus Physics-informed self supervised machine learning

These results follow from the experiment setup described in Section 3.4.1 and showcases the difference in the Convolutional Autoencoder's ability to generate waveform solutions, when trained using supervised learning compared to a physics-informed self-supervised learning. The model performance is displayed as a histogram over the F-values computed using the generated waveforms to the clutter maps contained in the held-out test dataset. The model generated histogram overlays the corresponding histogram of the benchmark optimized waveforms within the test dataset.



(a) F-value performance of the Convolutional Autoencoder when its trained using supervised learning.



(b) F-value performance of the Convolutional Autoencoder when its trained using physics-informed self-supervised learning.

Figure 4.1: The figures showcase the F-value performance differences between when the Convolutional Autoencoder is trained using supervised learning (left) compared to training the model using physics-informed self-supervised learning (right) over the held-out test dataset.

These figures shows that the two learning paradigms leads to distinctively different F-value performance histograms. The histogram for the model trained using supervised learning shows a gaussian-shaped performance distribution centered around

a mean F-value of ≈ -8.22 while the model trained using physics-informed self-supervised learning results in a more concentrated F-value distribution centered at a F-value of ≈ 11.04. Comparing the results to the benchmark histogram of F values with a mean F-value of ≈ 13.41, the supervised learning approach yields a distribution that is significantly shifted towards lower F-values while the model trained using physics-informed self-supervised yields a performance distribution similar to the benchmark's performance. This showcases the difficulty of using direct supervision to train a model to generate waveforms, and that a physics-informed self-supervised training paradigm can be used to train a model to generate waveform matrices with F-value performance similar to the benchmark.

4.2 Experiment 2: Single-output performance across dimensions

These results focus on how F-value performance differs between models across the three waveform dimensions $\mathbf{M2 - N_b5}$, $\mathbf{M4 - N_b15}$ and $\mathbf{M12 - N_b30}$. The result is the outcome of the experiment described in Section 3.4.2.

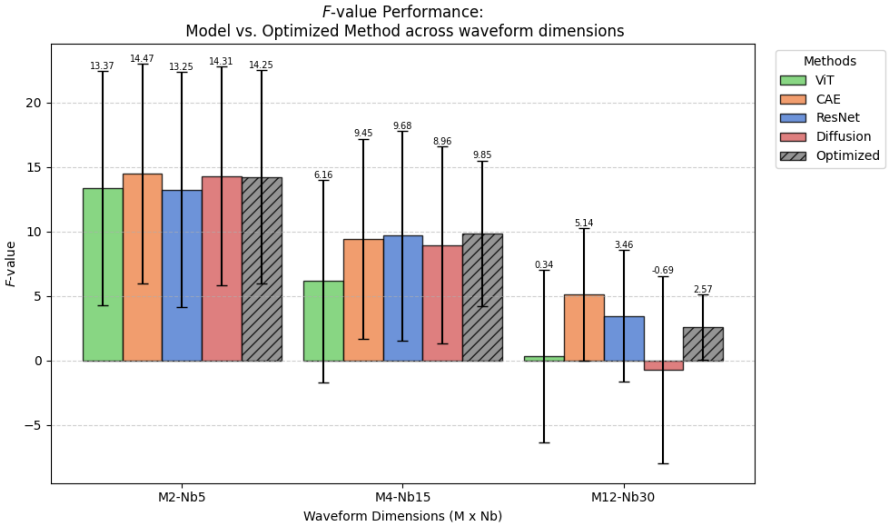


Figure 4.2: The figure displays the mean (placed above the standard deviation error bars) and standard deviation (displayed as error bars) of the F-value performance metric over the generated solutions to the held-out test dataset. This visualization includes the performance for each of the four different models across the three waveform dimensions that were evaluated, benchmarked against optimized waveforms for each waveform dimension.

Figure 4.2 shows the models ability to generate waveform solutions for three different waveform dimensions compared to the benchmark of optimized waveforms. The lowest waveform dimensionality ($\mathbf{M2 - N_b5}$) is stable across models and on par with the benchmark of optimized waveform solutions. The mean F-values for the evaluated models range from ≈ 13.35 to ≈ 14.47 while the benchmark reaches a mean F-value of ≈ 14.25, and the standard deviation remains similar between

models and the benchmark.

Increasing the waveform dimensionality to $\mathbf{M4 - N_b15}$, the mean F-values for the Convolutional autoencoder, Residual Network (ResNet) and the score based Diffusion Model (Diffusion) ranges from ≈ 8.98 to ≈ 9.68, while the vision transformer model achieves a lower mean F-value of ≈ 6.16. Comparing to the benchmark that reaches a mean F-value of ≈ 9.85, all models except the vision transformer reach similar F-value performance levels. However, the increased waveform dimensionality results in a larger standard deviation of the F-values for all models compared to the benchmark.

For the highest waveform dimensionality $\mathbf{M12 - N_b30}$, the figure shows that the performance of the Convolutional Autoencoder and the Residual Network (ResNet) achieves high mean F-values of ≈ 5.14 and ≈ 3.45 respectively, compared to the benchmark's mean F-value of ≈ 2.57. The result also shows that the vision transformer and the diffusion model fall short for this waveform dimensions in terms of mean F-values, with values at ≈ 0.34 and ≈ -0.69 respectively. Similarly to the results with waveform dimensionality $\mathbf{M4 - N_b15}$, the standard deviation of the model generated waveforms is visibly larger than the corresponding benchmark waveforms.

Overall, these results show that the model performance can degrade with increased waveform dimensionality. All models perform similarly and are comparable to the benchmark for lower dimensions, while the Convolutional Autoencoder and the Residual Network (ResNet) show a better scalability for waveforms of higher dimensionality compared to the Vision Transformer and Diffusion model. Furthermore, the results indicate that increasing the waveform dimensionality leads to larger differences in standard deviation of F-values relative to the benchmark.

Figure 4.3 gives some qualitative examples from the evaluated test dataset, with a side-by-side view of clutter maps, benchmark waveform ambiguity functions and model generated waveform ambiguity functions for each of the three waveform dimensions. The Convolutional Autoencoder is used to represent model generated examples. Each ambiguity function is normalized according to Eq. (3.26), with a maximimum possible value of 0 dB, and the abiguity functions are displayed together with their maximum value and corresponding F-value. The examples show that the optimization benchmark results in more local clutter suppression compared to the model generated waveforms, which instead use larger patterns of suppression covering substantially more area than the clutter itself. These large suppression patterns produced by the model do however cover the clutter regions well, resulting in good F-values relative to the benchmark.

4.2.1 Inference latency

All of the trained models for this experiment were evaluated on inference latency. This was measured using the held-out test dataset, timing each generated waveform

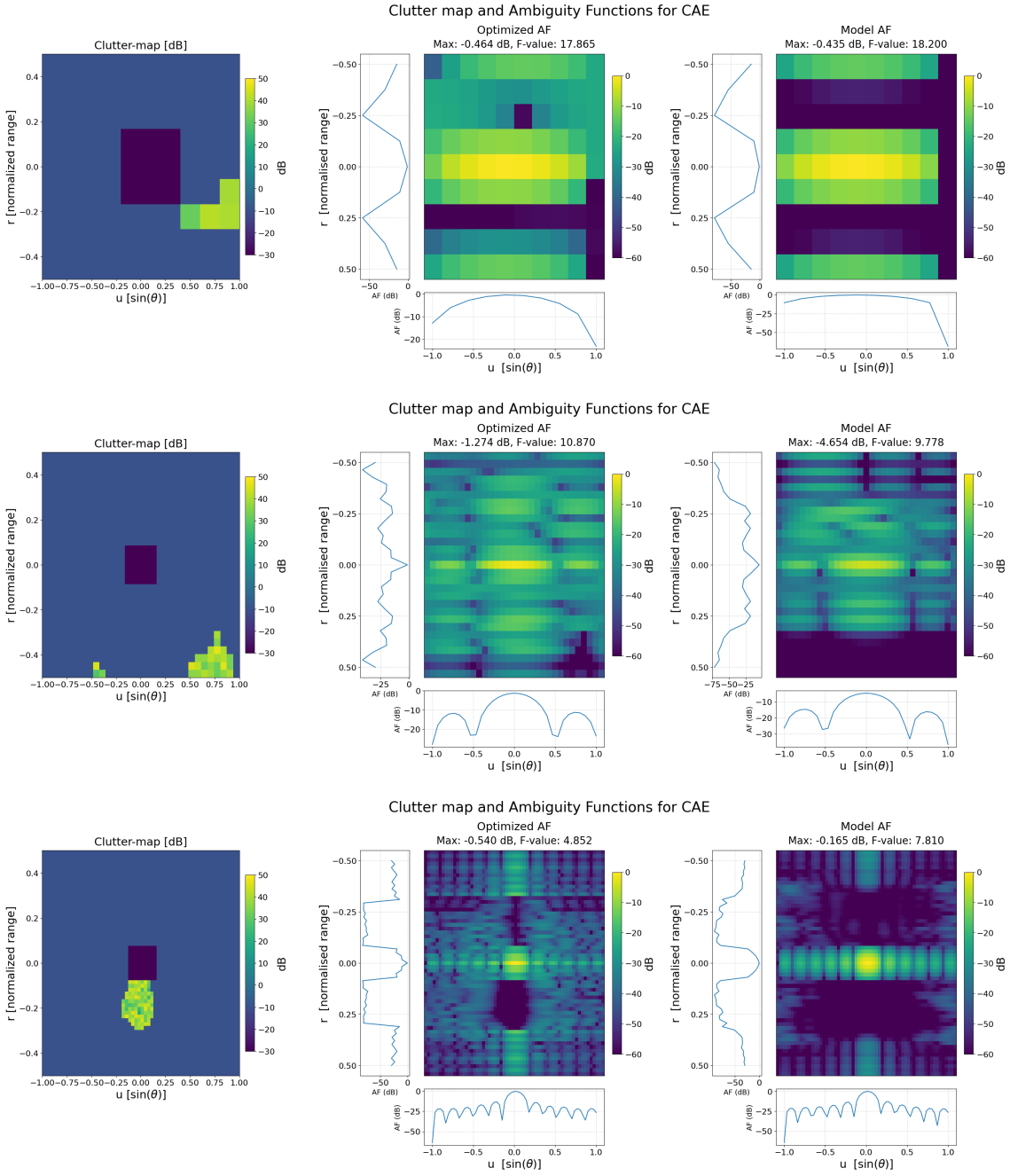


Figure 4.3: Examples of ambiguity functions derived from waveform matrices generated by the optimization benchmark (middle) versus the Convolutional Autoencoder (CAE) model (right) across the waveform dimensions $\mathbf{M2 - N_b5}$, $\mathbf{M4 - N_b15}$ and $\mathbf{M12 - N_b30}$ for the given clutter map (left). The visualization includes two panel plots showing the cross sections of the ambiguity function at $(r = 0)$ and $(u = 0)$. The ambiguity functions are normalized such that their maximum possible values are 0 dB.

using the `perf_counter`-function from the Python built-in module `time`. A warm-up period of 200 examples were discarded, such that the presented measurements were taken over the other 800 examples in the test set. Measurements were taken

as the time to perform a single forward pass, corresponding to a batch size of 1. The models are timed separately running on CPU only and with GPU acceleration through the `pytorch` library with which the models were implemented. The CPU measurements were taken using an `Intel(R) Xeon(R) Platinum 8462Y+`, and the GPU measurements used an `NVIDIA L40S`.

	CPU Inference [ms]		
Name	$\mathbf{M2 - N_b5}$	$\mathbf{M4 - N_b15}$	$\mathbf{M12 - N_b30}$
ViT	5.01 ± 1.61	13.84 ± 3.52	14.06 ± 1.21
CAE	1.98 ± 1.56	2.12 ± 1.41	2.49 ± 0.92
ResNet	1.95 ± 0.05	2.87 ± 0.14	5.47 ± 2.16
Diffusion	222.45 ± 28.78	233.77 ± 33.52	247.88 ± 31.78

Table 4.1: CPU inference times in milliseconds [ms] across the three evaluated waveform dimensions from the experiment setup described in Section 3.4.2.

	GPU Inference [ms]		
Name	$\mathbf{M2 - N_b5}$	$\mathbf{M4 - N_b15}$	$\mathbf{M12 - N_b30}$
ViT	1.84 ± 0.02	1.97 ± 0.05	1.98 ± 0.21
CAE	0.29 ± 0.02	0.27 ± 0.02	0.26 ± 0.04
ResNet	0.98 ± 0.03	0.96 ± 0.02	1.24 ± 0.02
Diffusion	15.44 ± 0.16	15.59 ± 0.09	15.81 ± 0.46

Table 4.2: GPU inference times in milliseconds [ms] across the three evaluated waveform dimensions from the experiment setup described in Section 3.4.2.

Both Table 4.1 and Table 4.2 shows that inference times varies substantially depending on architectural choices and model type. The Tables shows a larger variety for the inference times across waveform dimensions evaluated on CPU compared to using GPU. The CAE model achieves the most stable and lowest inference times across the three waveform dimensions with inference times on CPU of $\approx 2.49 \pm 0.92$ ms and $\approx 0.26 \pm 0.04$ ms for the highest dimensions $\mathbf{M12 - N_b30}$, while the inference result for the diffusion model showed slowest inference times of 247.88 ± 31.78 on CPU and 15.81 ± 0.46 on GPU for the same waveform dimension.

4.3 Experiment 3: Multi-output performance and diversity

As described in Section 3.4.3, this experiment evaluates the performance of the CAE and ResNet models when producing multiple waveform outputs for each clutter map, as well as the diversity of the generated waveforms. The models are trained to generate different number of waveforms K per clutter map, using the same training

data, and are evaluated using a test set of 1000 clutter maps.

For each model we record the scalar objective F-value, the bandwidth-regularized SCNR-value presented in (3.21), and evaluate the average performance of all outputs for the test set compared with the best-of-K value. The best-of-K value is obtained by, for each clutter map in the test set, selecting the best output candidate produced by the model. The average value is calculated over all K outputs for each clutter map in the test set. The results of this can be seen in Figure 4.4, displaying the mean and best-of-K values for each model architecture as a function of K.

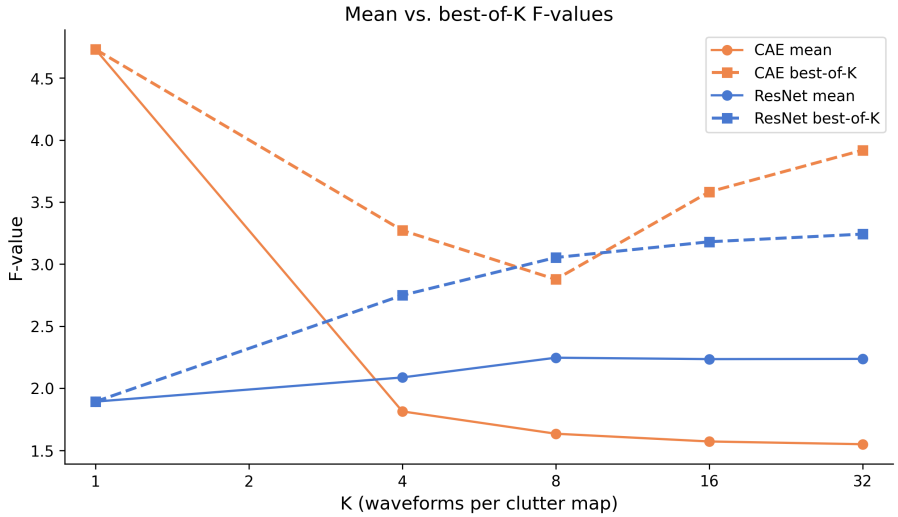


Figure 4.4: Mean F-values and best-of-K F-values for the test set, comparing the CAE and ResNet models with different numbers of K generated waveforms. The x-axis uses a logarithmic base-2 scale to emphasize the exponential increase in number of candidates.

The two model architectures show distinctly different behaviors when increasing the amount of generated outputs, where the CAE steadily decreases it's mean performance when increasing K, while it's best-of-K value decreases from $K = 1$ to 8 and subsequently increases again up to the final $K = 32$. The ResNet model on the other hand steadily increases both it's mean and best-of-K F-value, and also maintaining a smaller difference between the two throughout all tested values of K.

In order to quantify the diversity of outputs and compare the two model architectures, the Vendi Score introduced Section 3.2.2.2 is used, measuring the effective number of unique waveforms in the set of K generated outputs. Figure 4.5 illustrates the Vendi Score as a function of K candidate waveforms for both models, averaged over the full test dataset.

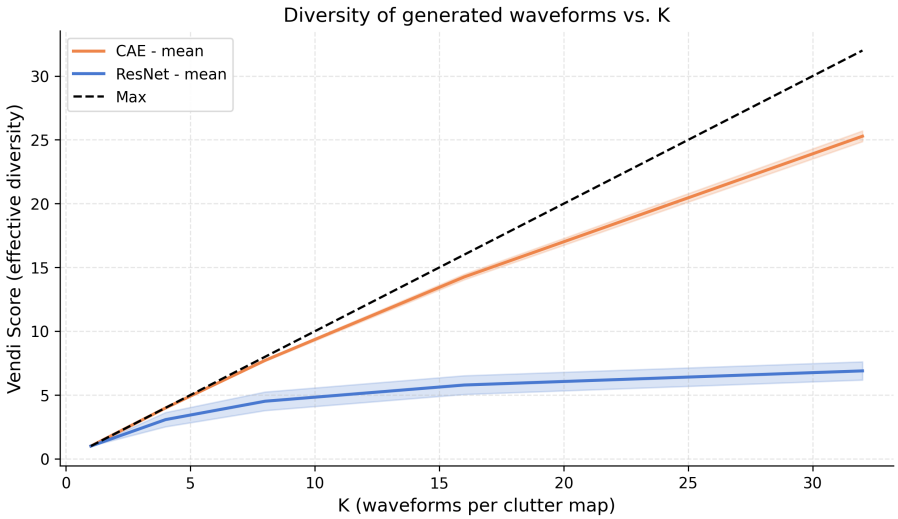


Figure 4.5: Vendi Score vs K for the CAE and ResNet model architectures, evaluated over the full 1000 example test set. The shaded region shows the standard deviation, and the dashed line shows the maximum theoretical Vendi Score ($VS_{\max} = K$)

There is a significant difference between the two model architectures. The CAE seems to obtain a large effective diversity, with Vendi Scores relatively close to the theoretical maximum illustrated by the dashed line. The ResNet on the other hand gets much lower Vendi Scores and increasingly diverges from the theoretical maximum with increased values of K, indicating a so-called mode collapse with low variations between different outputs.

4.3.1 Example waveforms and similarity matrices

To illustrate an example of how the two model architectures behave in detail, we select a representative clutter map from the test set, shown in Figure 4.6, and visualize the model generated waveforms and corresponding similarity matrices. The similarity matrices are displayed together with corresponding Vendi Scores, in this case using the models trained to generate $K = 4$ waveforms per clutter map.

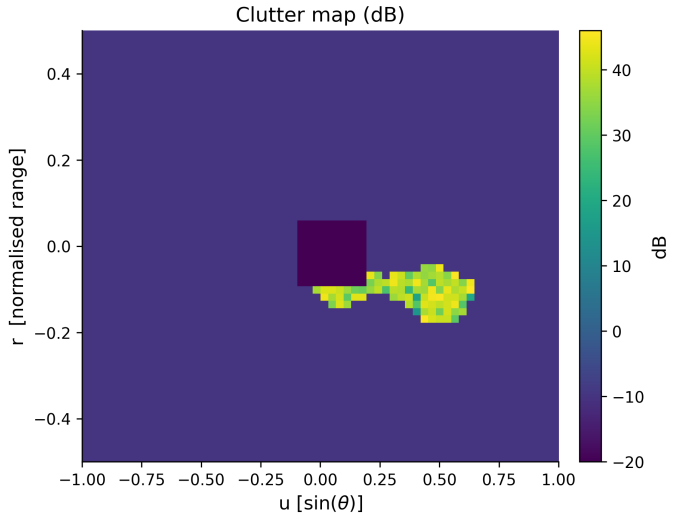


Figure 4.6: The selected clutter map example from the test set used to visualise model outputs and performance in more detail. The clutter map is visualised in a dB scale.

In Figure 4.7, the waveforms generated by the CAE model are shown together with corresponding ambiguity functions. The ambiguity function is normalised such that its maximum possible value is 0 dB as described in Eq. (3.26). Figure 4.8 shows the waveforms generated by the ResNet model for the example clutter map, together with corresponding ambiguity functions.

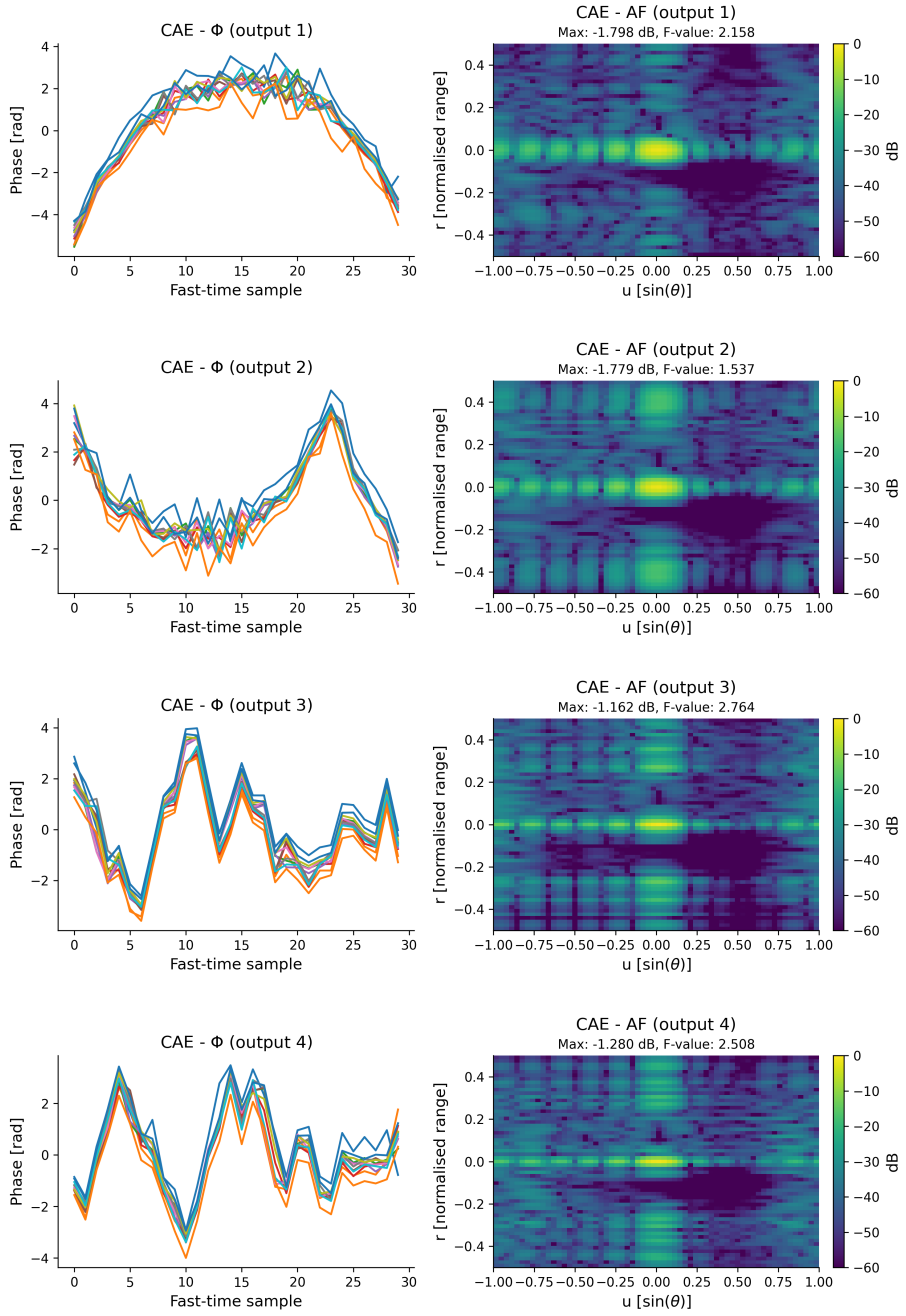


Figure 4.7: Four waveform candidates generated by the CAE for the clutter map of Figure 4.6. Each panel shows the time-domain waveform (left) together with its ambiguity function (right). Each ambiguity function panel shows the max value of the normalised ambiguity function and the F-value of the corresponding waveform.

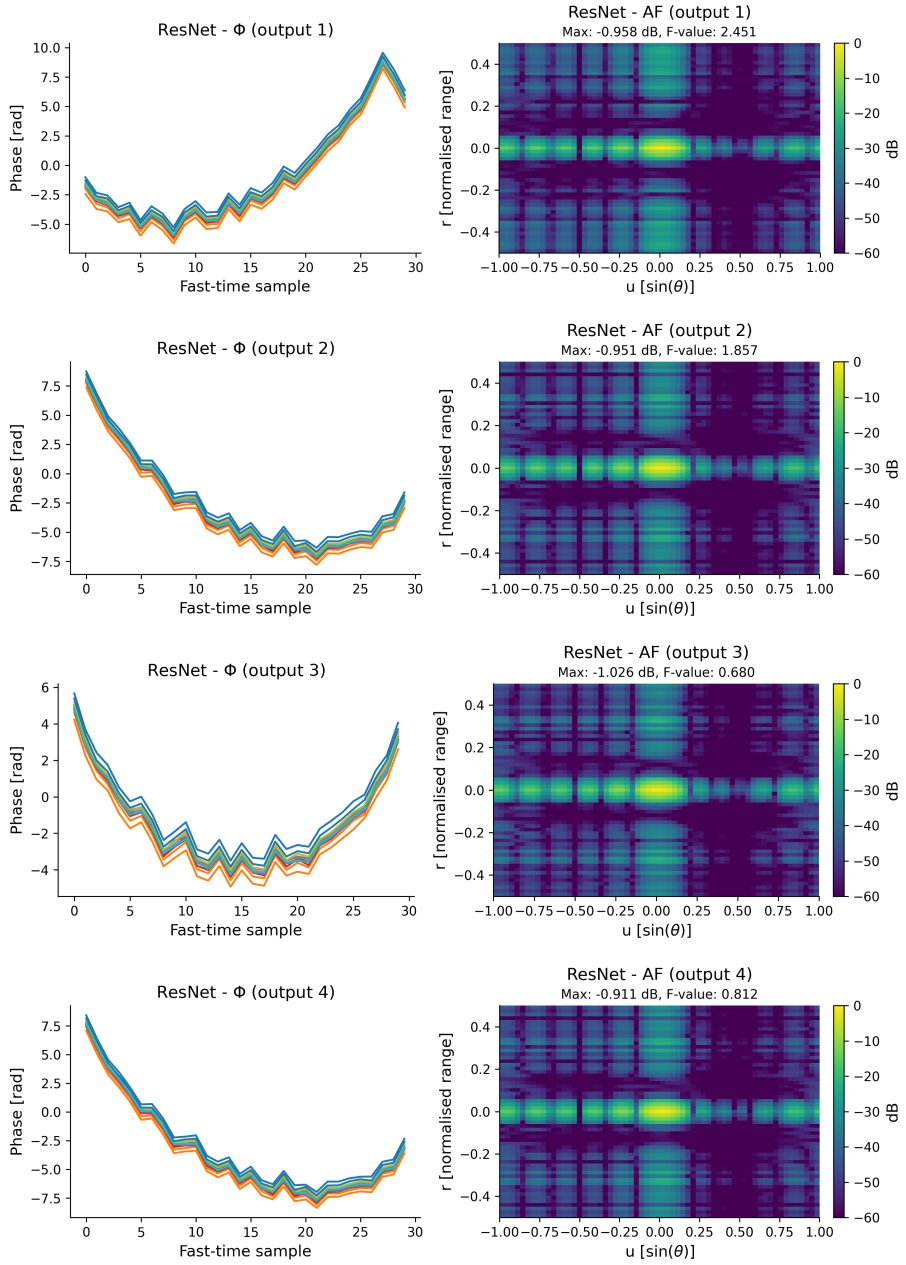
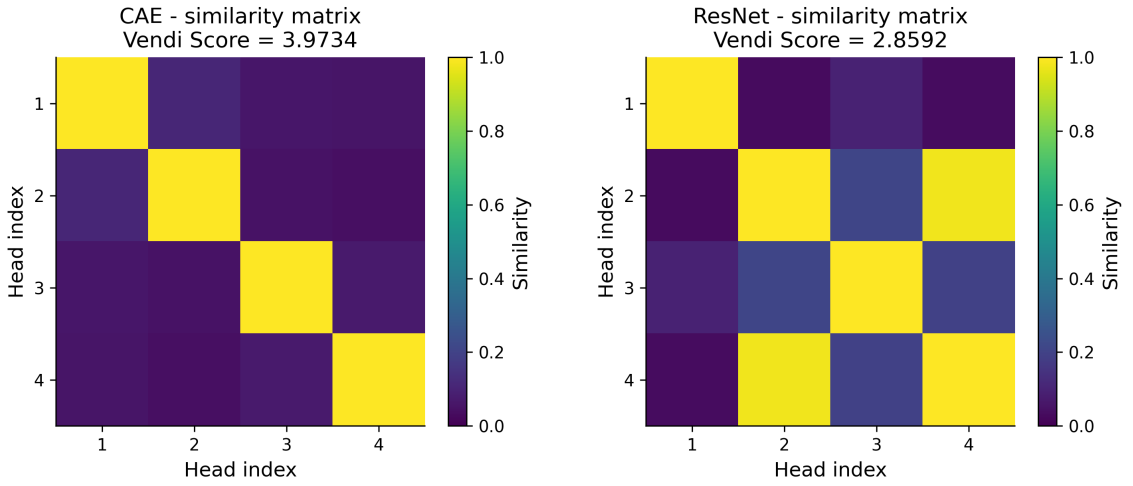


Figure 4.8: Four waveform candidates generated by the ResNet for the example clutter map. Each panel shows the time-domain waveform (left) together with its ambiguity function (right). Each ambiguity function panel shows the max value of the normalised ambiguity function and the F-value of the corresponding waveform.

The ResNet model outputs in Figure 4.8 tend to be more similar in character compared to the CAE waveforms in Figure 4.7, especially the outputs indexed two and four for the ResNet model. The general characteristics of the waveforms also differ between the model architectures, where the ResNet produces more correlated waveforms between the different antenna elements. The CAE examples have a more varied waveform shape between antenna elements, and also produces an ambiguity function that is more locally adapted to the clutter map in Figure 4.6, whereas the

ResNet-generated waveforms have ambiguity functions with more widespread suppression. Another aspect to note in Figures 4.7 and 4.8 is the varying F-values, particularly waveforms 2 and 4 from the ResNet output, that differ by ≈ 1 dB despite the strong similarity. This highlights the intricacy of the solution space, where small changes in phase-coding can have a large influence on the waveforms overall performance.

Using the same waveform outputs generated by the models as previously displayed in Figures 4.7 and 4.8, two similarity matrices are calculated and shown in Figure 4.9. The similarity matrix is calculated using the far-field similarity function $\text{sim}[i, j]$ as formulated in Eq. (2.32). It can be noted that the CAE waveforms have low pair-wise similarity throughout all possible pairs, resulting in a Vendi Score of ≈ 3.97 which is close to the highest possible value of 4. As previously noted waveforms number 2 and 4 from the ResNet's outputs are visually similar in terms of their phase-coding, which is confirmed by the similarity matrix in Figure 4.9b, leading to a lower Vendi Score of ≈ 2.86.



(a) CAE similarity matrix **(b)** ResNet similarity matrix

Figure 4.9: Pair-wise similarity matrices for the four candidates generated by the (a) CAE and (b) ResNet models for the example clutter map. The Vendi Score computed from each matrix is displayed above their respective subfigures, with the highest possible value being equal to $K = 4$.

4.3.2 Inference latency

All of the trained models for this experiment were evaluated on inference latency. This was measured using the held-out test dataset, timing each generated waveform using the `perf_counter` function from the Python built-in module `time`. A warm-up period of 200 examples were discarded, such that the presented measurements were taken over the other 800 examples in the test set. Measurements were taken as the time to perform a single forward pass, corresponding to a batch size of 1. The models are timed separately running on CPU only and with GPU acceleration

through the `pytorch` library with which the models were implemented. The CPU measurements were taken using an `Intel(R) Xeon(R) Platinum 8462Y+`, and the GPU measurements used an `NVIDIA L40S`.

Table 4.3: Inference time for all models trained for this experiment, shown as mean ± standard deviation. The times are presented in milliseconds [ms].

Model (K)	**CPU [ms]**	**GPU [ms]**
ResNet ($K = 1$)	2.48 ± 0.59	0.75 ± 0.05
ResNet ($K = 4$)	3.18 ± 0.14	0.77 ± 0.07
ResNet ($K = 8$)	4.41 ± 0.45	0.87 ± 0.04
ResNet ($K = 16$)	9.31 ± 0.75	1.07 ± 0.05
ResNet ($K = 32$)	18.27 ± 1.03	2.00 ± 0.11
CAE ($K = 1$)	2.26 ± 0.45	0.27 ± 0.02
CAE ($K = 4$)	2.78 ± 0.97	0.29 ± 0.02
CAE ($K = 8$)	3.45 ± 0.95	0.39 ± 0.05
CAE ($K = 16$)	4.33 ± 0.91	0.77 ± 0.00
CAE ($K = 32$)	6.38 ± 0.86	1.33 ± 0.02

The CAE model architecture obtains slightly lower inference latency across all tested values of K, on both CPU and GPU alike, although both models are within comparable time scales. The largest deviation is for the $K = 32$ example using CPU only, where the CAE is rougly three times faster. One should however note that the results are highly dependent on the underlying `pytorch` library functionality, as well as `CUDA` functionality for the GPU-accelerated inference latencies.

4.4 Experiment 4: Single-output performance and generalizability across clutter map difficulty levels

The following results show the Convolutional Autoencoder's ability to generate well-adjusted waveforms to clutter maps of six different difficulty levels. The difficulty level is parametrized as the median number of clutter regions per clutter map denoted as m_{CM}.

Figure 4.10 shows the CAE model performance as a train-test difficulty matrix with separate panels showing mean and standard deviation of F-values, where the x-axis represents the difficulty level of the training dataset (1-6), and the y-axis represents the difficulty level of the test dataset (1-6). For all training configurations, the

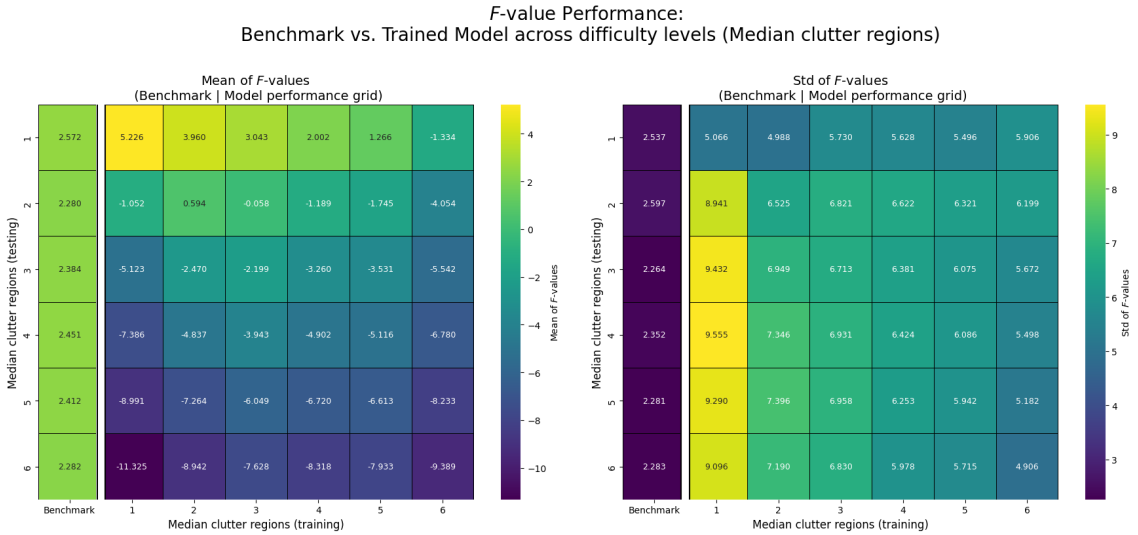


Figure 4.10: Model performance across different difficulty levels, benchmarked against the Saab-provided test datasets. The performance is shown through the mean (left) and standard deviation (right) of the resulting F-values computed through (3.21) using the 1000 clutter maps in the test datasets. The result is shown together with the corresponding optimized waveforms contained in the test data set (benchmark). The model performance is shown for different combination of m_{CM} that the model is trained on (x-axis) versus tested on (y-axis). This displays how performance changes and the generalizability of the model to unseen clutter map distributions. The mean and standard deviation describing the benchmark performance is shown as a column to the left of the model performance grids.

performance is noticeably correlated with the test difficulty level with higher mean F-values for lower difficulties and vice-versa. On the other hand, the difficulty level used during training has comparatively smaller effects on the test performance. Across all combinations in the train-test difficulty matrix, the mean F-value ranges from approximately -11.32 to 5.23. The benchmark mean F-values instead remains stable, varying in the range ≈ 2.28 to ≈ 2.57 across different values of m_{CM}.

Focusing on the diagonal of the train-test difficulty grid, where the value of m_{CM} is the same for the training dataset and the testing dataset, Figure 4.10 shows a clear trend of decreasing mean F-values when the number of clutter regions m_{CM} is increased. These performance levels verify that the median number of clutter regions contained in the datasets m_{CM} can be used as a proxy to the difficulty of training a waveform generator model since m_{CM} is the only thing varying between the training runs, while total area covered by clutter and distribution of clutter amplitudes remain the same.

Comparing the model's performance to the benchmark (left column), the figure shows that the model manages to reach mean F-values higher than the benchmark for models trained on datasets with $m_{CM} \in [1, 2, 3]$ when tested on the single clutter region test dataset $m_{CM} = 1$, but fall short of the benchmark for all other combinations of training and testing datasets in terms of mean F-values. Focusing on the

standard deviation of the F-values, this result shows that the benchmark performance is overall more stable than the model performance. The standard deviation of the F-values for the benchmark ranges from ≈ 2.26 to ≈ 2.60 while the model's standard deviation of F-values ranges from ≈ 4.91 to $\approx 9,56$. This shows that the benchmark's F-value distribution is both more stable across difficulty levels m_{CM} and significantly lower than what any combination of training and testing m_{CM} can achieve.

The model performance grid also shows that increasing the number of clutter regions m_{CM} contained in the training data leads to an increased test performance for test data with higher values of m_{CM}, up to the point where $m_{CM} = 3$ for the test set. Thereafter the model's mean F-value performance does not increase as clearly compared to the initial three difficulty levels.

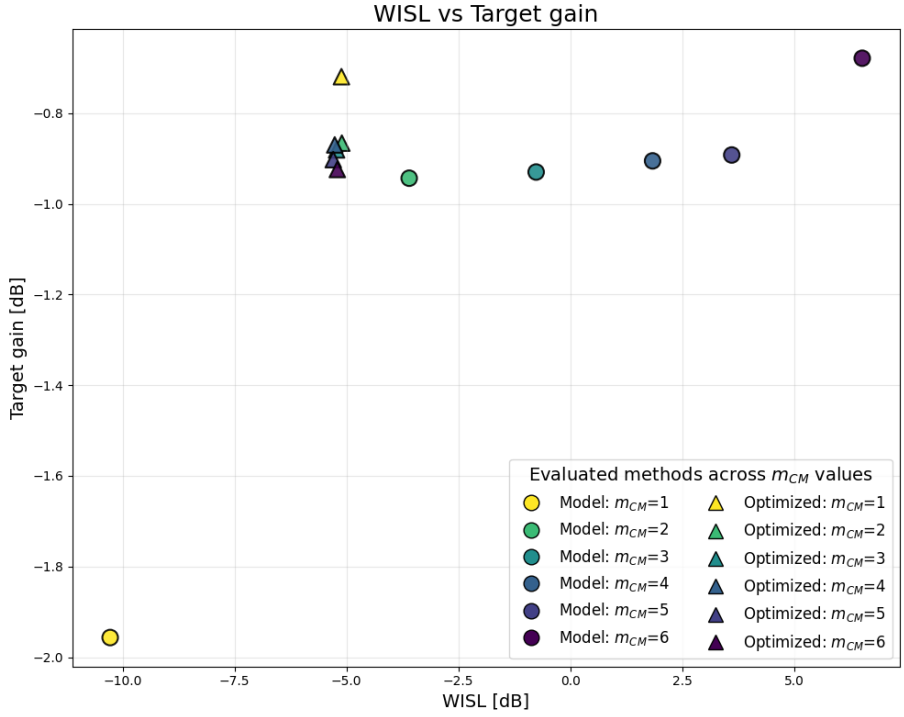


Figure 4.11: Target gain vs WISL for model generated and optimized waveforms across difficulty levels m_{CM} in dB.

Figure 4.11 displays the tradeoff between target gain and WISL that the model makes across difficulty levels when the models train and test data has the same value of m_{CM}. The target gain is computed as $\widetilde{G}(u_0)$ introduced in Eq. (3.27), which is the return power from the target after matched filtering, normalized such that it's maximum possible value is 0 dB. The WISL represents the total energy returned from the clutter, computed using $\widetilde{\text{WISL}}$ from Eq. (3.28). The WISL is computed using an ambiguity function normalized to a maximum possible value of 0 dB, same as the target gain. The result shows the stability of the benchmark optimization method while the WISL values for the model generated waveforms increase with the difficulty level, which describes that the model's suppression of clutter re-

gions decreases with increasing m_{CM}. The visualization also shows that the target gain stays relatively stable for $m_{CM} \in 2, 3, 4, 5, 6$. Relating this to the diagonal of the model train-test difficulty grid shown in Figure 4.10, the reason for decreased mean F-value performance for higher m_{CM} values is caused by a decreasing ability to suppress clutter returns when the clutter is distributed into a higher number of small clutter regions.

To give some visual examples, Figure 4.12 shows one clutter map from each dataset with different levels of m_{CM}. In Figure 4.13, the corresponding ambiguity functions for the model generated waveforms are shown, with each ambiguity function displayed together with it's F-value.

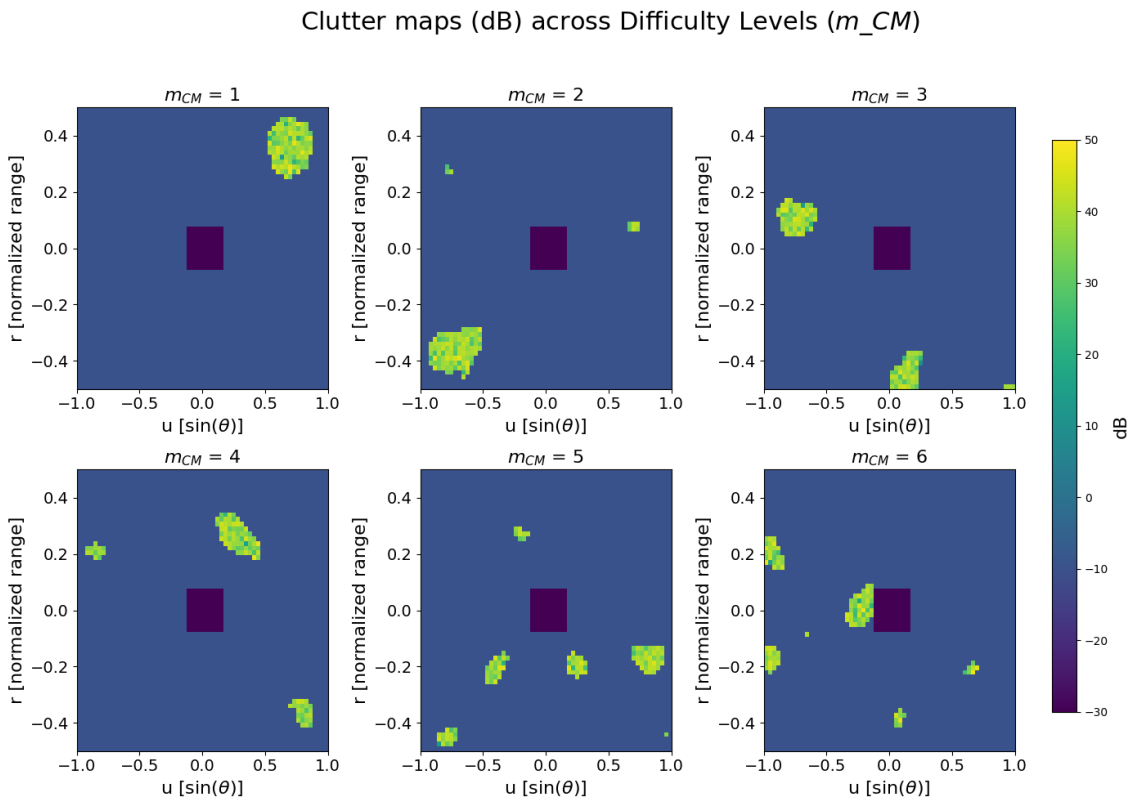


Figure 4.12: Examples of clutter maps from the six different datasets across m_{CM} values.

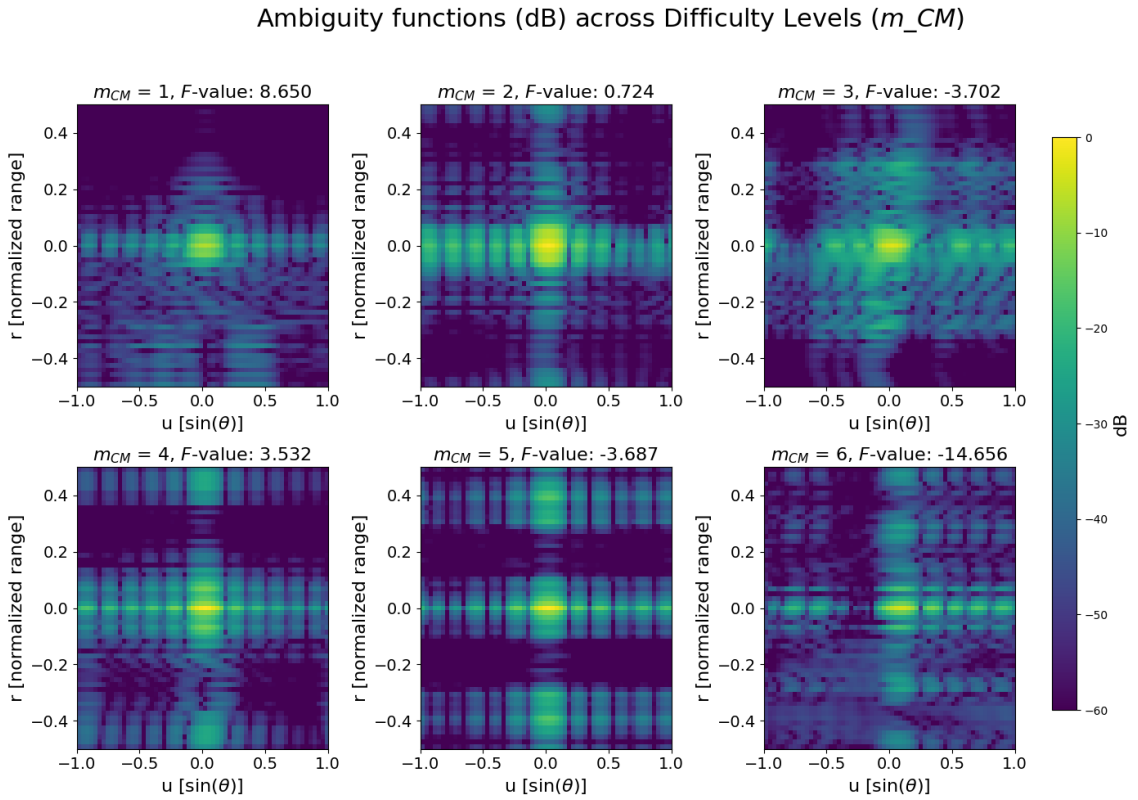


Figure 4.13: Ambiguity functions for the model generated waveforms, corresponding to the clutter maps shown in Figure 4.12.

5
Discussion

This chapter presents a discussion based on the found results and implemented methods, in relation to the aims presented in Section 1.3. The discussion focuses on key results from each experiment, as well as general characteristics of how the developed neural networks generate waveforms for given clutter environments.

5.1 Learning paradigm

In Experiment 1, Section 4.1, the two different learning paradigms of supervised learning and physics-informed self-supervised learning where evaluated using the proposed CAE model. The two learning paradigms are compared using the same dataset, where the supervised learning uses clutter map - optimized waveform pairs to train using an MSE-based loss, whereas the physics-informed self-supervised learning trains directly on the radar performance metric F-value, not using any optimized waveforms during training. The difference between the two methods can be seen in Figure 4.1, where the physics-informed self-supervised learning shows an F-value distribution ($\mu \approx 11.0$) close to the benchmark optimized waveforms ($\mu \approx 13.4$). The supervised learning regime instead results in a substantially worse performance ($\mu \approx -8.2$).

These findings indicates that supervision in the waveform matrix is excessively restrictive. The supervised loss forces the network to approximate the distribution of waveforms in the training dataset, and in the case of non-convex, multi-modal loss landscapes it might not be able to sufficiently generalize patterns between samples. Further, given the sensitivity of the SCNR-based F-value where small changes in the waveform phase-matrix might cause large differences in performance, a supervised loss can miss intricate differences that have large performance impacts while still obtaining low loss values. In other words, the supervised loss might signal good outputs when their performance is poor, by just being close to a good solution in terms of waveform coding. On the other hand, since the waveform performance is directly verifiable it can be leveraged directly as a loss function for training the neural networks. This self-supervised approach proves far superior to the implemented supervised learning in this setting, and circumvents the previously stated issues with a supervised loss function.

The strength of the proposed self-supervised learning approach is further supported in Experiment 2 in Section 4.2, where it is shown to obtain results comparable with

the benchmark optimized waveforms across all three of the tested waveform dimensionalities. As the degrees of freedom in waveform increases from the 2×5 used in Experiment 1 to 12×30, one could expect even more difficulty for a supervised learning regime, as variations in training data might grow and generalizing patterns can become increasingly difficult.

Another notable strength for using a self-supervised regime is the question of data, where a supervised regime demands annotated data which might not be feasible to obtain in sufficient amounts and quality, the self supervised approach is instead limited to having sufficient amounts of clutter map examples only. With synthetically generated data, one can in theory use an unlimited amount of clutter maps for training, which can give a better coverage of the variations encountered in testing or deployment settings.

5.2 Performance scaling across waveform dimensionality

Figure 4.2 displays that the dimensionality of waveforms and clutter maps, which compose the waveform generation problem, strongly influence F-value performance of the trained models. This can be seen when analyzing the models F-values across the three waveform dimensions $2 \times 5, 4 \times 15$ and 12×30. At the lowest dimensionality 2×5, all evaluated models perform comparably and on par with the optimized waveforms, while a discrepancy between model architectures forms when the dimensionality increases. This indicates that architectural choices for a waveform generator model becomes increasingly more important for higher waveform dimensions. One interesting aspect shown in Figure 4.2 is that the performance of the convolutional autoencoder (CAE) and the residual model (ResNet) scales robustly across dimensions among the evaluated model architectures under fixed amounts of data, trainable parameters and training process. The performance trajectory suggests that these model architectures can utilize the current data distribution, data size and model complexity more effectively when scaled to higher dimensions compared to the vision transformer (ViT) and Diffusion models.

Focusing on the higher waveform dimension 12×30, the results find that the Convolutional Autoencoder (CAE) and the Residual Network (ResNet) shows a higher mean F-value performance at 5.14 and 3.45 respectively, compared to the optimized benchmark with a mean F-value of 2.57. The models high performance in this dimension together with the robust scaling trajectory across dimensions discussed in the previous section, provides an empirical foundation that further waveform dimensionality expansion is promising. However, the variance of the machine learning models F-values is still higher compared to the benchmark. This result points towards a tradeoff between high mean performance and stability when training machine learning models to generate waveforms.

Table 4.1 and Table 4.2 shows that the CAE and ResNet models achieved the lowest

inference times among the four evaluated model architectures, where the CAE model has the overall lowest inference times on both CPU and GPU, and seems to be more stable under changes of waveform dimensions compared to the ResNet model. The combination of high mean F-value performance together with the lowest inference times that the CAE model accomplishes throughout waveform dimensions, indicates that this model architecture is a suitable option both if waveform dimensions where to increase and if there is strict latency requirements for the waveform generation process.

5.3 Characteristics of model generated waveforms

Figure 4.3 shows that the model finds waveform solutions that suppresses larger areas of the range-angle map compared to the benchmark method across the evaluated dimensions. One can see that the model has learned to factorize the clutter suppression problem by generating waveforms matrices that suppress returns across a large range of r or u bins, instead of suppressing more precisely at clutter regions. By using factorization, the examples shows that it is possible to generate waveform solutions with high F-values to the given clutter map without the need for precise local suppression. This way of handling the clutter suppression problem is reasonable for an adaptive waveform generator that should be able to handle clutter in various range-angle positions. Factorizing the problem and focusing on learning how to generate more general waveform solutions can be a key factor that makes it possible for the model to generalize and handle a large variety in clutter maps, where reusable relationships and suppression patterns can lead to a better average performance.

5.4 Multi-output generation and diversity of waveforms

The results from Experiment 3 in Section 4.3 indicate that diversity of waveforms with similar performance for the same clutter map is achievable, but presents a tradeoff between performance and diversity and varies between different types of model architectures. From Figure 4.4, we can see that the average performance of the CAE degrades when increasing the number of generated waveforms (K), whereas the stochastically seeded ResNet model improves with increased number of outputs. For the CAE, the best F-value performance is achieved when training and testing the model on only generating a single output, as the $K = 1$ case outperforms even the average best-of-K solutions when increasing K. The ResNet model instead improves both its average performance and best-of-K performance steadily when increasing K.

When analyzing the achieved diversity between the two tested models, the CAE shows a significant advantage over the ResNet model indicated by the Vendi Scores illustrated by Figure 4.5, which measures the effective number of diverse samples in a set. The CAE model achieves Vendi Scores close to the theoretical maximum,

slightly dropping of with increased K, while the ResNet reaches a plateau at around a Vendi Score of $\sim 5-6$. Relating this result to the pure waveform performance, the indication is that there is a tradeoff between obtaining diversity and maintaining performance. The qualitative examples for both models further illustrate the difference in behaviour, where the CAE produces waveforms with greater differences both in their visual representation and the similarity matrix together with the corresponding Vendi Score. Overall, the results point towards that the ResNet gives rise to a more local and limited search of the output landscape, while the CAE is able to more effectively search for distinctly different waveforms.

Looking at the produced ambiguity functions for the generated examples, in Figures 4.7 and 4.8, the CAE produces ambiguity functions more locally adapted to the clutter environment at hand, and the ResNet instead seems to suppress larger areas of the spatial landscape. This indicates that the ResNet converges on more reusable patterns that might be applicable to more than one scenario. This pattern also occurs in other examples and from other models, such as the CAE-generated waveform ambiguity functions in Figure 4.13 and in Figure 4.3, where the same widespread suppression of the ambiguity function can be seen.

The difference in behavior between the two model architectures when trying to achieve diversity might be counterintuitive, since the ResNet is stochastically seeded and thereby incorporates diversity directly into its architecture, producing different outputs by initializing with separate random seeds. However, the CAE achieves diversity by having separate output heads, producing a predetermined amount of different waveforms in parallel, which in practice means that different neural-network weights are used for the separate K outputs. During training, these separate heads can more easily be molded towards different regions of the output space, while the ResNet propagates through the same weights for every separate output. This seemingly presents a more demanding challenge when tasked to both produce waveforms with good SCNR-based performance and a diversity between samples generated with different seeds.

Despite the stronger results in terms of diversity achieved by the CAE, there are some notable disadvantages to approaching diversity by simply scaling with separate output heads. Mainly, more output heads lead to a larger model and thereby demands more memory. As seen in Table 4.3, the CAE still achieves better inference time than the ResNet even when scaling up K both on CPU and GPU, but the scaling of output heads can limit its ability for deployment on devices with limited memory and compute resources. Moreover, the ResNet-approach with stochastic seeding can be used to generate an arbitrary amount of waveforms with the same model.

5.5 Generalizability across clutter map difficulty levels

When analyzing the model performance grid and benchmark performance across different difficulty levels of the clutter maps in Figure 4.10, it is clear that the benchmark's performance is much more stable across different difficulty levels compared to the model performance, both in terms of mean and standard deviation of the F-values. The benchmark presents high-performing waveform solutions regardless of the difficulty levels, while the model is more sensitive to changes of the number of clutter regions.

The columns of Figure 4.10 also shows that when increasing the number of clutter regions per clutter map, m_{CM} for the test data, the mean F-values decreases regardless of the data used during training. The diagonal of the performance grid, where the model is trained and tested on the same difficulty level, also confirms a clear decrease in performance when m_{CM} increases. This monotonic decrease in mean F-values with increasing m_{CM} confirms that the value of m_{CM} is a valid quantization of the difficulty level of the clutter map data used for training a model to become an adaptive waveform generator.

Since the same model, training setup and amount of data is used throughout all training runs, this result confirms that increasing m_{CM} makes the adaptive waveform optimization problem harder to solve for the model. Figure 4.11 shows two different components that builds up the performance F-value, target gain and WISL. The result shows that one of the main reasons to why the model performance decrease when increasing m_{CM} is higher WISL levels. When inspecting the result, one can see that the model does a tradeoff between target gain and WISL levels, where the model manage to achieve good clutter suppression and lower target gain for larger, cohesive clutter regions ($m_{CM} = 1$) while having worse clutter suppression and higher target gain for more difficult clutter maps ($m_{CM} = 6$). This shows that increasing difficulty level shifts which parts of the training objective that can be optimized more easily, resulting in a training focused on low WISL for simpler clutter scenarios compared to high target gain for higher difficulty levels. Comparing the results to how the optimized benchmark perform, the plot also verifies the stability of the benchmark waveforms across difficulty levels, and that the trained model manages to perform similarly when it comes to target gain, but fall short of the benchmark for clutter suppression and WISL levels.

One reason for why the machine learning model performs worse compared to the benchmark when trained on higher m_{CM} is the increased data complexity it brings. It both leads to a larger spread of clutter regions per clutter map, see Table A.2, and an increased number of possible permutations of where the clutter is located for a given map. This means that the data the model is trained on for higher difficulty levels contains a much larger variation with both increased σ_{CM} and clutter location combinations. Having a large variety in the data a model is trained on generally makes it harder to extract usable patterns from any training data distri-

bution. Combining this with the sensitive objective function defined in Equation (3.22), these possible reasons makes it harder to train a machine learning model when m_{CM} is high.

While the model demonstrates sensitivity to the difficulty level across the performance grid, it is worth noting that the model trained on $m_{CM} \in 1, 2, 3$ and tested on data with $m_{CM} = 1$ shows better performance than the benchmark for these combinations of difficulty levels in the data. The results here can be related to previously identified characteristics of how the model generates waveforms with larger and more reusable suppression patterns, as discussed in Section 5.3. This can be advantageous when dealing with large clutter regions that are fewer in number, but it can also be more difficult to apply to larger numbers of disconnected clutter regions with high variability in spatial distribution.

6
Conclusion and Future work

This thesis has investigated the use of machine learning for generating clutter-suppressing MIMO radar waveforms, comparing to a benchmark of optimized waveforms. Several different approaches and settings were explored, with varied training approaches, neural network architectures and variations in clutter environment. Additionally, the ability to generate a diverse set of waveforms for each given clutter environment was investigated.

The results show that training neural networks can be a feasible approach to waveform design in clutter environments, with performance similar to the optimization-based waveforms given as benchmark over the three different waveform dimensions tested. Given the ability of neural networks to leverage off-line training and generalization over a wide array of potential scenarios for a low-latency waveform synthesis during deployment, the results are promising for future research and testing. The ability to train models directly using a verifiable objective function evaluating waveform performance is shown to be an advantageous approach, especially in comparison to plain supervised learning.

By testing the performance of different model architectures using different waveform dimension and comparing to a benchmark of optimized waveforms, neural networks are shown to achieve comparable results throughout different degrees of freedom. With increased waveform dimensions, the tested Diffusion- and Vision Transformer models showed an inferior scaling compared to the Convolutional Autoencoder and Residual Network, possibly due to being less data-efficient during training. This indicates that the waveform design problem becomes increasingly difficult with increased scale, demanding more from the model and training setup.

Neural networks are also shown to be able to generate multiple proposed waveforms suitable for the same clutter environment, but with a strong difference in results between the two model architectures evaluated for this purpose. The stochastic ResNet model showed signs of mode collapse, with more local search of waveforms with similar properties. The Convolutional Autoencoder instead managed to produce more diverse sets of waveforms, by having parallel output heads with decoupled weights producing the different output waveforms instead of relying on stochastic seeding. Moreover, the diminished average performance of the Convolutional Autoencoder when increasing the number of generated waveforms points towards a tradeoff between generating a diverse output set and maintaining performance.

A central result of the neural-network based approach to waveform design is that the performance can be highly dependent on shape and distribution of clutter in the environment surrounding the target. Performance is comparable to the benchmark of optimized waveforms in the test set when the clutter regions are larger and fewer in number rather than a larger number of small clutter regions. This can be connected to an observed pattern of models using broader patterns of ambiguity function suppression, aiming for more reusable patterns suitable for several different clutter environments. This finding gives an important insight into the strengths and weaknesses in using machine learning models for the waveform design problem at hand.

In summary, a neural network-based approach to waveform design in clutter environments shows promising results, with the ability to generate waveforms comparable to a benchmark of optimized waveforms on previously unseen clutter scenarios. However, it has both strengths and weaknesses, with better results when dealing with fewer and larger, more cohesive clutter regions. It's also shown possible to generate a diverse set of solutions to the same clutter environment, with the level of diversity distinctly dependent on the chosen method to approach multiple outputs. The results show that neural networks can be a feasible approach to waveform synthesis, and gives a promising indication for future development and extensions to other waveform design objectives, or other parts of the radar system in general.

6.1 Future work

While this work has shown how machine learning models can be used to train adaptive and diverse waveform generators, several aspects of it would be interesting to explore further.

Since the results has shown that it is possible to generate waveforms comparable to the benchmark for several waveform dimensions, further investigation about how the methods and performance scales to higher waveform dimensions is of interest.

This work focuses on generating waveforms with fixed dimensions and thereby fixed degrees of freedom that the model is constrained to and needs to utilize. Different clutter scenarios may require different degrees of freedom to enable high levels of clutter suppression, which makes it interesting to research how machine learning can be used to generate waveforms with an arbitrary number of fast-time samples. This would make for a natural extension, since although the number of antenna elements is normally fixed and restricted by the radars hardware, the pulse length can often be varied.

One limitation that this work followed was to train a waveform generator to suppress clutter in the range-angle domain. A natural development of this work is to expand to the Doppler domain. This makes is possible to both have Doppler-dependent clutter and to enforce other Doppler-specific criteria. On the same note, it would be natural to extend this work to other waveform design objectives as well. One such

example would be multiple targets, or waveform design for a so called search mode, where the objective is to scan a large search volume for potential targets.

While several modeling techniques have been utilized to handle the sensitive and complex waveform design objective, further loss-landscape analysis for different waveform dimensions could build a strong foundation for how the informed machine learning approach should be improved.

Bibliography

[1] B. Friedlander, "Adaptive signal design for mimo radars," in *MIMO Radar Signal Processing* (J. Li and P. Stoica, eds.), ch. 5, pp. 193–234, John Wiley & Sons, Ltd, 2008.

[2] M. I. Skolnik, *Radar Handbook*. New York, NY: McGraw-Hill Education, 3rd ed., 2008.

[3] M. S. Greco and S. Watts, "Chapter 11 - radar clutter modeling and analysis," in *Academic Press Library in Signal Processing: Volume 2* (N. D. Sidiropoulos, F. Gini, R. Chellappa, and S. Theodoridis, eds.), vol. 2 of *Academic Press Library in Signal Processing*, pp. 513–594, Elsevier, 2014.

[4] S. Sun, Y. Zhang, J. Cao, and Y. Feng, "Phase-coded waveform design for mimo radar based on nonlinear conjugate gradient method," in *2025 7th International Conference on Information Science, Electrical and Automation Engineering (ISEAE)*, pp. 601–605, 2025.

[5] J. Li and P. Stoica, "Mimo radar with colocated antennas," *IEEE Signal Processing Magazine*, vol. 24, no. 5, pp. 106–114, 2007.

[6] X. Yu, K. Alhujaili, G. Cui, and V. Monga, "Mimo radar waveform design in the presence of multiple targets and practical constraints," *IEEE Transactions on Signal Processing*, vol. PP, pp. 1–1, 03 2020.

[7] Z. Xie, L. Wu, X. Huang, C. Fan, J. Zhu, and W. Liu, "Cross ambiguity function shaping of cognitive mimo radar: A synergistic approach to antenna placement and waveform design," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 62, pp. 1–13, 2024.

[8] K. Zhong, W. Zhang, Q. Zhang, J. Hu, P. Wang, X. Yu, and Q. Zhou, "Mimo radar waveform design via deep learning," pp. 1–5, 05 2021.

[9] V. Saarinen and V. Koivunen, "Mimo radar waveform synthesis using generative adversarial networks," in *2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2023.

[10] S. Guo, R. E. White, and M. Low, "A comparison study of radar emitter identification based on signal transients," in *2018 IEEE Radar Conference (RadarConf18)*, pp. 0286–0291, 2018.

[11] L. Xu, H. Liu, S. Zhou, J. Liu, and J. Yan, "Colocated mimo radar waveform design against repeat radar jammers," in *2018 International Conference on Radar (RADAR)*, pp. 1–5, 2018.

[12] G. Cui, A. D. Maio, A. Farina, and J. Li, "9. cognitive local ambiguity function shaping with spectral coexistence and experiments," in *Radar Waveform Design Based on Optimization Theory*, Institution of Engineering and Technology (The IET), 2020.

[13] L. Wu, P. Babu, and D. P. Palomar, "Cognitive radar-based sequence design via sinr maximization," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 779–793, 2017.

[14] A. Aubry, A. DeMaio, A. Farina, and M. Wicks, "Knowledge-aided (potentially cognitive) transmit signal and receive filter design in signal-dependent clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 93–117, 2013.

[15] H. He, J. Li, and P. Stoica, *Waveform Design for Active Sensing Systems: A Computational Approach*. Cambridge University Press, 2012.

[16] J. Tabrikian, "Performance bounds and techniques for target localization using mimo radars," in *MIMO Radar Signal Processing* (J. Li and P. Stoica, eds.), ch. 4, pp. 153–191, John Wiley & Sons, Ltd, 2008.

[17] D. Rabideau and P. Parker, "Ubiquitous mimo multifunction digital array radar," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 1, pp. 1057–1064 Vol.1, 2003.

[18] J. Bergin and J. Guerci, *MIMO Radar: Theory and Application*. Artech, 2018.

[19] J. Li and P. Stoica, "Mimo radar — diversity means superiority," in *MIMO Radar Signal Processing* (J. Li and P. Stoica, eds.), ch. 1, pp. 1–64, John Wiley & Sons, Ltd, 2008.

[20] M. C. Budge and S. R. Jr. German, "1. radar basics," in *Basic Radar Analysis (2nd Edition)*, ch. 1, Artech House, 2020.

[21] M. A. Richards, *Fundamentals of Radar Signal Processing*. New York: McGraw Hill, 3rd ed., 2022.

[22] O. Rabaste, L. Savy, M. Cattenoz, and J.-P. Guyvarch, "Signal waveforms and range/angle coupling in coherent colocated mimo radar," in *2013 International Conference on Radar*, pp. 157–162, 2013.

[23] P. Stoica, J. Li, and Y. Xie, "On probing signal design for mimo radar," *IEEE Transactions on Signal Processing*, vol. 55, no. 8, pp. 4151–4161, 2007.

[24] A. Aubry, A. De Maio, and Y. Huang, "Mimo radar beampattern design via psl/isl optimization," *IEEE Transactions on Signal Processing*, vol. 64, no. 15, pp. 3955–3967, 2016.

[25] G. Turin, "An introduction to matched filters," *IRE Transactions on Information Theory*, vol. 6, no. 3, pp. 311–329, 1960.

[26] P. Stoica, H. He, and J. Li, "New algorithms for designing unimodular sequences with good correlation properties," *IEEE Transactions on Signal Processing*, vol. 57, no. 4, pp. 1415–1425, 2009.

[27] J. Song, P. Babu, and D. P. Palomar, "Sequence design to minimize the weighted integrated and peak sidelobe levels," *IEEE Transactions on Signal Processing*, vol. 64, no. 8, pp. 2051–2064, 2016.

[28] L. Huang, X. Deng, L. Zheng, H. Qin, and H. Qiu, "Joint design of colocated mimo radar constant envelope waveform and receive filter to reduce sinr loss," *Sensors*, vol. 21, no. 11, 2021.

[29] B. Boashash, "1.3 instantaneous frequency (if) and spectral delay (sd)," in *Time-Frequency Signal Analysis and Processing - A Comprehensive Reference*, Elsevier, 2016.

[30] D. A. Hague, "Continuous phase modulation of phase coded transmit waveforms using multi-tone sinusoidal frequency modulation," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pp. 1393–1397, 2021.

[31] T. Liu, J. Sun, G. Wang, X. Yao, and Y. Qiao, "Optimal design of group orthogonal phase-coded waveforms for mimo radar," *Mathematics*, vol. 12, no. 6, 2024.

[32] L. Xu, S. Zhou, H. Liu, and J. Liu, "Repeat radar jammer suppression for a colocated mimo radar," *IET Radar, Sonar & Navigation*, vol. 13, no. 9, pp. 1448–1457, 2019.

[33] C. Bishop, *Pattern recognition and machine learning*, vol. 4. Springer New York, 2006.

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

[35] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, New York: Springer, 2nd ed. ed., 2009.

[36] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *CoRR*, vol. abs/1902.06162, 2019.

[37] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, and J. Schuecker, "Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 614–633, 2023.

[38] T. X. Nghiem, J. Drgoňa, C. Jones, Z. Nagy, R. Schwan, B. Dey, A. Chakrabarty, S. D. Cairano, J. A. Paulson, A. Carron, M. N. Zeilinger, W. S. Cortez, and D. L. Vrabie, "Physics-informed machine learning for modeling and control of dynamical systems," 2023.

[39] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

[40] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 6000–6010, Curran Associates Inc., 2017.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[43] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," in *International Conference on Learning Representations*, 2021.

[44] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," *CoRR*, vol. abs/1907.05600, 2019.

[45] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th International Conference on International*

Conference on Machine Learning, ICML'11, (Madison, WI, USA), p. 681–688, Omnipress, 2011.

[46] J. Terven, D.-M. Cordova-Esparza, J.-A. Romero-González, A. Ramírez-Pedraza, and E. A. Chávez-Urbiola, "A comprehensive survey of loss functions and metrics in deep learning," *Artificial Intelligence Review*, vol. 58, no. 7, p. 195, 2025.

[47] D. Friedman and A. B. Dieng, "The vendi score: A diversity evaluation metric for machine learning," *Transactions on Machine Learning Research*, 2023.

[48] O. Roy and M. Vetterli, "The effective rank: A measure of effective dimensionality," in *2007 15th European Signal Processing Conference*, pp. 606–610, 2007.

[49] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[51] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472, 2017.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

A

Clutter map dataset statistics

The clutter weights for each dataset were generated using $seed = 42$ which resulted in a clutter amplitude spread within each clutter region across datasets to remain stable and reproducible which can be seen in Table A.1. The distribution of clutter amplitudes follows a Weibull distribution for all datasets, which is described in the following table:

Table A.1: Distribution of the clutter amplitudes per dataset (rounded).

Dataset name	median	mean	std	p75	p99	max
65K-DS-M2xNb5	6956	9970	9866	13926	45748	119032
1CM-DS-M2xNb5	7015	10023	9924	13914	45743	110578
1CM-DS-M4xNb15	6922	9991	9981	13864	45969	138135
1CM-DS-M12xNb30	6935	10002	10000	13870	46044	161259
2CM-DS-M12xNb30	6930	10000	9998	13862	45981	161259
3CM-DS-M12xNb30	6933	9998	10002	13854	46042	158236
4CM-DS-M12xNb30	6931	9996	9995	13857	46020	158235
5CM-DS-M12xNb30	6934	10001	9999	13864	46036	158235
6CM-DS-M12xNb30	6934	10001	9993	13865	46039	144748

In the report, the number of clutter regions that are present in a clutter map is used as a proxy for the difficulty of generating a well-suited waveform matrix $\mathbf{\Phi}$ given the clutter map. Table A.2 describes the distribution of the number of clutter regions across clutter maps per dataset used throughout this report.

Table A.2: Statistics for the number of clutter regions per the dataset.

Dataset name	m_{CM}	μ_{CM}	σ_{CM}
65K-DS-M2xNb5	1	1.35	0.55
1CM-DS-M2xNb5	1	1.15	0.46
1CM-DS-M4xNb15	1	1.29	0.61
1CM-DS-M12xNb30	1	1.29	0.61
2CM-DS-M12xNb30	2	2.03	0.97
3CM-DS-M12xNb30	3	3.01	1.31
4CM-DS-M12xNb30	4	3.98	1.52
5CM-DS-M12xNb30	5	5.00	1.60
6CM-DS-M12xNb30	6	6.04	1.87

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se