



ARL-TR-10054 • JAN 2025



PyRadarWF: A New Dataset Generator for Machine Learning of Radar Waveforms

by Brent Kraczek, Matthew Ziemann, and Marius Necsoiu

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.



PyRadarWF: A New Dataset Generator for Machine Learning of Radar Waveforms

by Brent Kraczek, Matthew Ziemann, and Marius Necsoiu
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) January 2025		2. REPORT TYPE Technical Report		3. DATES COVERED (From - To) October 2024	
4. TITLE AND SUBTITLE PyRadarWF: A New Dataset Generator for Machine Learning of Radar Waveforms				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Brent Kraczek, Matthew Ziemann, and Marius Necsoiu				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLA-NA Aberdeen Proving Ground, MD 21005-5066				8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-10054	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.					
13. SUPPLEMENTARY NOTES Primary author's email: <brent.e.kraczek.civ@army.mil>.					
14. ABSTRACT Python Radar Waveform (PyRadarWF) is a new dataset generator for machine learning (ML) and analysis of radar waveforms. While ML based on deep learning has developed rapidly for 2-D image recognition, it lags behind for 1-D signal analysis, such as RF data and time-series data. This is true both of ML and the data needed to generate the ML models. Existing radar datasets are primarily proprietary, while open-source and DoD-available datasets are very limited. PyRadarWF has been developed to provide a wide variety of data for ML use, allowing customization to broaden the datasets. The data generation tools are designed to be extensible, including base waveform objects to allow for the straightforward addition of new radar waveforms, and a flexible data labeling format for training on the same data for different properties. In addition to data generation, PyRadarWF includes PyTorch-based ML modules and data plotting and analysis tools. We conclude with ML models trained on our generated data and tested against both emulated data and synthetic data from external sources.					
15. SUBJECT TERMS radar; dataset generation; machine learning; Python; Network, Cyber, and Computational Sciences					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 50	19a. NAME OF RESPONSIBLE PERSON Brent Kraczek
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (Include area code) 410-278-8881

Contents

List of Figures	iv
List of Tables	v
1. Introduction	1
2. Radar Waveform and Dataset Generation	4
2.1 Radar Waveforms Used in this Study	4
2.2 Datasets Used to Guide Development	6
2.2.1 The Synthetic SIDLE Dataset	6
2.2.2 The Emulated FRAME Dataset	9
2.3 Data Import and Analysis Tools	10
2.4 PyRadarWF Dataset Generation	11
2.4.1 Basic Dataset Generation Process	11
2.4.2 PyRadarWF PDW	12
2.5 PyRadarWF Waveform Generation	17
3. ML Methods	19
3.1 ResNet	19
3.2 Transformer	21
4. ML Results	23
4.1 Discussion	25
5. Conclusions and Future Direction	33
6. References	36
Appendix. Input Format for Dataset Generation	38
List of Symbols, Abbreviations, and Acronyms	41
Distribution List	43

List of Figures

Fig. 1	Idealized Barker 7 waveform. a) Block diagram representing the Barker 7 waveform multiplier. b) Idealized Barker 7 signal with carrier wave.	5
Fig. 2	Sample waveforms from the noise-free version of SIDLE. In the Barker 7 waveform, the finite switching time is the most prominent feature, while in the Nested Barker 16 and the Polyphase P1-100, the transients are most prominent. All three pulses have pulse widths of ~ 3000 time steps. a) Barker 7. b) Nested Barker 16. c) P1-100.	8
Fig. 3	Barker 7 waveform construction process used by PyRadarWF. a) The Initial envelope created, including slow switching time. b) Transients are added based on switching times. c) Rise time, frequency, and phase offsets are added; noise is added to give 24-dB SNR. Finally, the signal is normalized to give a standard deviation of 1 for entire signal.	18
Fig. 4	ResNet-34. Architecture diagram of ResNet-34 adapted from He et al. ⁹	20
Fig. 5	Transformer architecture. Overview of our transformer-based NN architecture with detailed transformer encoder layer. ¹¹	22
Fig. 6	Comparison of confusion matrices for same properties and similar data for WF-Mods09 and WF-Mods18. In both cases, datasets generated by PyRadarWF are used for training, with the datasets employing the same parameters and random distributions for data generation, though different random seeds, aside from the additional classes in WF-Mods18. For inference, we use FRAME IQ data, with SNR as the target property and ResNet18 as the ML architecture. a) WF-Mods09. b) WF-Mods18.	27
Fig. 7	Comparison of confusion matrices for inference on FRAME FT WF-Mods18 data, with ResNet18 and transformer NN architectures. a) ResNet18 b) Transformer.	29
Fig. 8	Modulation classification of FRAME IQ data inferred on a transformer NN model trained on PyRadarWF IQ data.	30
Fig. 9	Modulation classification of FRAME FT data inferred on a transformer NN model trained on PyRadarWF FT data.	31
Fig. 10	Modulation classification of FRAME IQ+FT data inferred on a transformer NN model trained on PyRadarWF IQ+FT data. Comparing these classification results with the results found in Figs. 8 and 9 show that the IQ+FT combination is not yet a reliable improvement over the IQ or FT results alone.	32
Fig. 11	Illustration of classification of pulse width ranges. It is common for signals to be classified as having a pulse width from a neighboring range. These inference calculations use WF-Mods09, SIDLE IQ data. a) ResNet18. b) Transformer NN.	34

List of Tables

Table 1	Sets of modulation classes used in this work. WF-Mods09 is a subset of the waveforms available in SIDLE. We have created datasets using the modulations in WF-Mods09 and WF-Mods18 using FRAME and PyRadarWF.	7
Table 2	PDW for PyRadarWF datasets. These are the properties needed to define the specific waveforms used in the dataset. The column, “Can be varied?” denotes properties that should be possible to vary in training. We have not yet implemented this.....	16
Table 3	Different training and inference properties of datasets and their application in ML. The only restriction on the combinations is that the SIDLE dataset does not contain all the modulations in WF-Mods18. For these results, we have trained only on PyRadarWF data, and inferred on three fixed datasets drawn from SIDLE or generated with PyRadarWF or FRAME. Within these constraints, we have computed all possible combinations. The total accuracy rates for all combinations are tabulated in Tables 4 and 5.	24
Table 4	Accuracy rate for each inference calculation for the WF-Mods09 datasets.	25
Table 5	Accuracy rate for each inference calculation for the WF-Mods18 datasets.	25

1. Introduction

Domain-specific machine learning (ML) datasets and methods will be essential to the development of specialized intelligent systems, including radar. As radar is based on radio frequency (RF) transmission and reception, ML for radar is related to other forms Radio Frequency Machine Learning (RFML), such as RFML for communications. RF signals are typically time-domain signals and hence one-dimensional. The development of image recognition has driven much of the deep learning research of past decades, such that the ML methods and training datasets required for two-dimensional (2-D) data are far more developed than for 1-D signals. In addition, due to the proprietary, and often classified, nature of radar system development and use, few radar datasets are available to the public or alternatively at the unclassified level.

In this technical report, we describe PyRadarWF (Python Radar Waveform), a new tool for creating radar datasets and performing ML on these datasets that we are developing at the U.S. Army Combat Capabilities Development Command Army Research Laboratory. PyRadarWF is being developed at the unclassified (Distribution A) level, based on publicly available information, intended to aid basic research into radar-oriented RFML. Written in Python, with an object-oriented structure, PyRadarWF is designed to be free of proprietary or hardware-specific design choices. Even the specific radar waveforms employed are based on their mathematical definitions, available in the public literature. Additional tools for pulse shaping, transients, jitter, and frequency modulation, are intended to be sufficiently flexible to cover waveform properties of more proprietary systems.

The primary objectives of this effort are as follows:

1. Identify the key elements needed to generate synthetic radar datasets that can be used in domain generalization.
2. Produce the computer code capable of generating these synthetic radar datasets.
3. Determine the ML methods needed to accurately and efficiently employ such radar datasets for use in real-world scenarios.
4. Develop means to characterize detected radar signals; in for example a pulse descriptor word (PDW) format.

This is the first technical report specifically about PyRadarWF, and discusses progress toward each of these four objectives. In our estimation, the first three objectives are coupled and best addressed together. For the first objective, we follow the definitions for domain generalization, domain adaptation, and transfer learning, given by Wang et al.¹ The differences between these terms are with regard to how much access the model training has to the test data. Radar detection systems will likely face both known and unknown data sources, requiring a system robust to unknown data, meaning domain generalization. For the fourth objective, a PDW is a list of parameters used to generate or characterize a radar waveform. While there are PDW standards, and PDWs often include similar properties, PDWs are often use-case specific. At present, we are targeting three properties that could be used in constructing a PDW, modulation (or waveform) class, pulse width (w_p), and signal-to-noise ratio (SNR).

We have used two data sources to guide waveform development beyond the mathematical models of the modulations, the Signal Instantiations for Deep Learning Experiments (SIDLE) dataset,²⁻⁵ and data generated by a radar emulation system, Fundamental Radar/EW Application Maturation Environment (FRAME).⁶ Following common convention, we refer to the first of these as *synthetic*, meaning the sample waveforms were generated by software, and the second as *emulated*, meaning the sample waveforms were generated using hardware that emulates a radar system, but were not transmitted over the air. As will be detailed below, the data from SIDLE and FRAME are quite different, even for the same modulations. We consider this to be important, both in guiding the design of PyRadarWF, and also for testing the results of our methods. We believe that ML models trained on PyRadarWF data that accurately classify waveform properties from both SIDLE and FRAME, without the inclusion of any other data, would demonstrate that PyRadarWF is capable of being used for domain generalization. In addition, success on the emulated dataset will be a significant milestone toward being able to classify over-the-air signals.

PyRadarWF is very much a research code, with significant, if specialized capabilities. One of the major design objectives is enough flexibility to efficiently consider different aspects of radar waveforms. Written in Python, it leverages standard libraries for numerical analysis, data analysis and ML libraries, including NumPy, SciPy, Matplotlib, and PyTorch, as well as Python’s object-oriented structure. An

example of the object-oriented design of PyRadarWF is the generation of different modulation classes. Currently, the code generates waveforms from the Barker,⁷ Nested Barker (or Combined Barker), and Polyphase modulation families,⁸ using classes derived from the base `RadarDatasetGenerator` class. Additional modulation families may be added in a straightforward manner, derived from this same base class. Another design choice is that the major code classes and methods of PyRadarWF are designed to be run either within the Jupyter-notebook environment or at the command-line, easing the process of using the tools on a local machine to analyze individual datasets, or alternatively, on a supercomputer to do large-scale ML. With regard to ML, the datasets generated by PyRadarWF employ multiple labeling sets, labeling a variety of waveform properties. As will be demonstrated in the ML results section, Section 4, we can use these labels to study which specific aspects of the waveforms are captured by different ML methods, aiding the improvement of both dataset generation and ML methods. This functionality will enable PDW-generation by classifying input signals on the individual PDW properties.

There are three main components to the current version of PyRadarWF, each of which will be described further:

1. Data generation tools: These tools are used to generate synthetic data with the different label sets described above.
2. Data import and analysis tools: Based on the capabilities found in the widely used Python libraries listed above, this component can be used to import data from different formats to convert into the native format used by the ML routines. It can also be used for plotting various aspects of the data to aid the analyst.
3. ML tools: Based on PyTorch and designed to be run on GPUs, these tools are primarily based on different neural network (NN) architectures, the residual NN (ResNet), adapted to 1-D, complex signals, and a companion transformer-based NN architecture, designed by one of us.^{9,10}

This report is not a user manual, but is intended to help interested researchers understand the design principles and capabilities of PyRadarWF and to aid the evaluation

of results obtained using PyRadarWF. The target audience is a computational scientist, working on data science or ML for RF transmissions, with some background in numerical analysis. While the current implementation is radar-specific, the framework could be extended to other RF signals. At present, researchers who may wish to use PyRadarWF may contact the authors, who can provide Python scripts and Jupyter notebooks to aid in understanding how to use the code.

The balance of this report is broken into the following sections. Section 2 discusses radar waveform and dataset generation, including a very brief introduction to radar waveforms. We also include in this section a brief discussion of the import and analysis tools. Section 3 discusses the ML methods currently deployed within PyRadarWF. Section 4 briefly summarizes results obtained by PyRadarWF. Section 5 concludes the material presented and discusses future research directions.

2. Radar Waveform and Dataset Generation

We break this section into five parts: A basic description of the radar waveforms used in this study, the datasets we have used to guide this development, the import and analysis tools used to work with the datasets, individual waveform generation and dataset generation. The datasets involved are typically composed of tens of thousands to millions of waveforms. We note that, at present, the objective of PyRadarWF is not to generate some canonical radar dataset. Instead, we seek to develop many datasets, based on the same and different distributions of waveform properties, in order to identify the key properties that ML methods will need to be effective and efficient.

2.1 Radar Waveforms Used in this Study

Radar waveforms are RF signals designed to optimize in some way the interpretation of the return signal. While it is outside the scope of this report to discuss why waveforms are constructed as they are, a basic understanding of waveform construction will aid in understanding the balance of this report. RF signals are commonly transmitted as in-phase (I) and quadrature (Q) components. Typically, the combined signal is referred to as I/Q or IQ. The I and Q may be understood in terms of the real Fourier components of the signals, where the cosine and sine components correspond to the I and Q components of the signal. As with the Fourier transform, it is often simpler to write this in terms of the complex numbers, with I and Q be-

ing represented by the real and imaginary parts of the complex numbers. The total signal power is given by the absolute square of the complex signal, $|z|^2$.

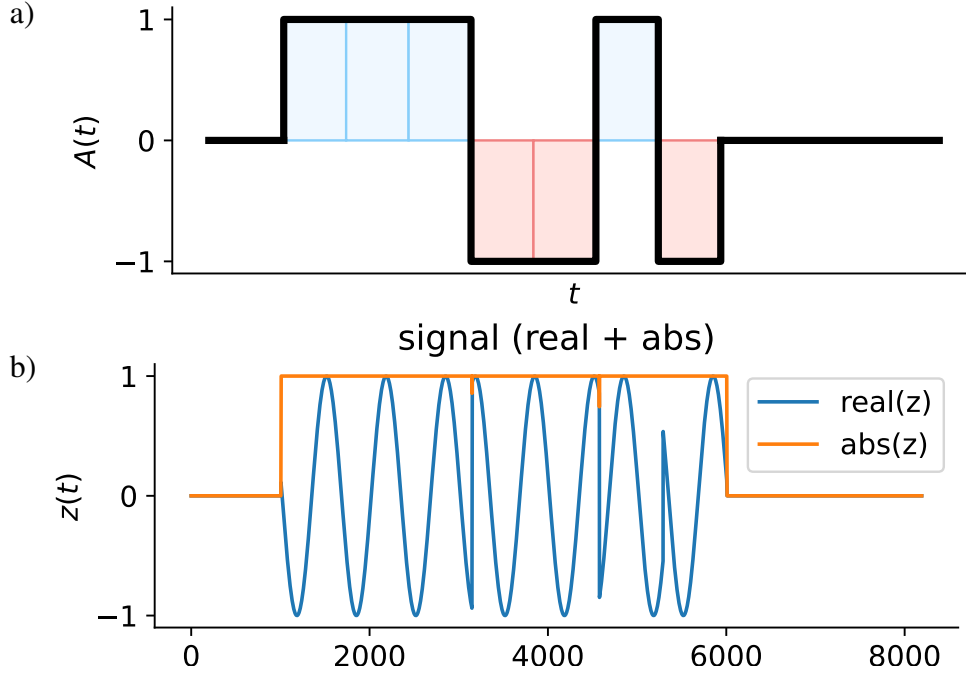


Fig. 1 Idealized Barker 7 waveform. a) Block diagram representing the Barker 7 waveform multiplier. b) Idealized Barker 7 signal with carrier wave.

We can illustrate this with a simple radar waveform, the Barker 7 waveform. Barker 7 is composed of cosine and sine waves at a given carrier frequency, with amplitude set by an envelope. In many radar waveforms, it is the envelope that gives the waveform name. As illustrated in Fig. 1a, the Barker 7 waveform is composed of seven square chips (subpulses) of equal length in time with an amplitude of ± 1 , where $A(t)$ denotes the waveform amplitude multiplier. The signal $z(t)$ with a set carrier frequency is shown in Fig. 1b. The total length of the pulse in time is the *pulse width*, so that the pulse width of the waveform represented in Fig. 1b is approximately 5000 (time units). We stress that these are a mathematical form, not the final waveform. In real world systems, a number of deviations from this ideal will occur. These include finite switching rates, transients caused by switching and noise, both internal to the transmitter and external, from the environment.

In the present study we employ three families of modulation classes, using either

9 or 18 modulation classes in our studies. We denote these sets “WF-Mods09” and “WF-Mods18.” These modulation class families were chosen because they exist in the SIDLE dataset and can be generated using the FRAME emulator. The three families we include are the Barker, Nested Barker (alternatively referred to as Combined Barker), and Polyphase families. All three families are based on square amplitude envelopes, in the same manner as the Barker 7 illustrated in Fig. 1, though in the case of the Polyphase family, the chip multipliers are typically complex numbers of amplitude 1, not strictly ± 1 . The Nested Barker waveforms are obtained by taking the Kronecker product of Barker waveforms, such that

$$(\text{NestedBarker49}) = (\text{Barker7}) \otimes (\text{Barker7}) .$$

The standard set of Polyphase waveforms, or *P-codes* are P1, P2, P3, and P4. The number of chips in each P-code can be chosen as squares of integers, with 16 being a common choice. The SIDLE dataset employs 64 chips for P2 and P3, and 100 chips for P1 and P4. In our studies, we have chosen to include some or all of 16, 64, and 100, and find that this poses a significant test, given the similarities between these modulation classes. We denote the different Polyphase classes by appending the number of chips (zero-padding for readability). Thus, P4 with 64 chips will be “P4-064,” etc. The specific modulation sets used in this work are given in Table 1.

2.2 Datasets Used to Guide Development

We have already discussed some of our design choices, based on the SIDLE and FRAME datasets. Here, we explain the properties observed in these datasets, to aid in understanding the design choices made for PyRadarWF. We also use these datasets as inference data for testing the ML models trained on PyRadarWF data, as presented in Section 4. In particular, we stress that these results are examples of how well our PyRadarWF datasets and ML architectures work for domain generalization on these datasets. As such, it is important to note the differences between SIDLE and FRAME data, and how they influence the properties we include in the PyRadarWF datasets.

2.2.1 The Synthetic SIDLE Dataset

The synthetic SIDLE dataset was developed by Pavy et al., at the Air Force Research Laboratory.^{2–5} SIDLE is widely used within the Department of Defense (DoD) for radar research, and has become, as such, a standard dataset. SIDLE con-

Table 1 Sets of modulation classes used in this work. WF-Mods09 is a subset of the waveforms available in SIDLE. We have created datasets using the modulations in WF-Mods09 and WF-Mods18 using FRAME and PyRadarWF.

WF-Mods09	WF-Mods18
Barker 7	Barker 7
Barker 11	Barker 11
Barker 13	Barker 13
Nested Barker 16	Nested Barker 16
Nested Barker 49	Nested Barker 49
	Nested Barker 169
	P1-016
	P1-064
P1-100	P1-100
	P2-016
P2-064	P2-064
	P2-100
	P3-016
P3-064	P3-064
	P3-100
	P1-016
	P4-064
P4-100	P4-100

sists of 23 radar modulations. In the present work, we use SIDLE’s data from the “WF-Mods09” waveform set given in Table 1. (We note that the P2 in SIDLE is the complex conjugate of the commonly used P2 signal. We correct this when reading-in the data.)

Within Barker, Nested Barker, and Polyphase families, the SIDLE dataset exhibits some interesting idiosyncrasies, each of which are different departures from the mathematical forms given in Fig. 1. These idiosyncrasies are common within the families (e.g., Barker 7 and 13 exhibit similar behavior), but different between families (e.g., Nested Barker 49 and P2-064 are quite different). As seen in Figs. 2 a, b, and c, which show Barker 7, Nested Barker 16, and P1-100, each of these classes of waveforms exhibit very different switching behavior, including a fairly slow switching time for the Barker 7 and some very strong transients induced in the Nested Barker 16 and the P1. Note: The most commonly used form of the SIDLE dataset has noise, with the SNR ranging from -15 to 15 dB. We are plotting the

noise-free version of the dataset, to help illustrate the waveform behavior.

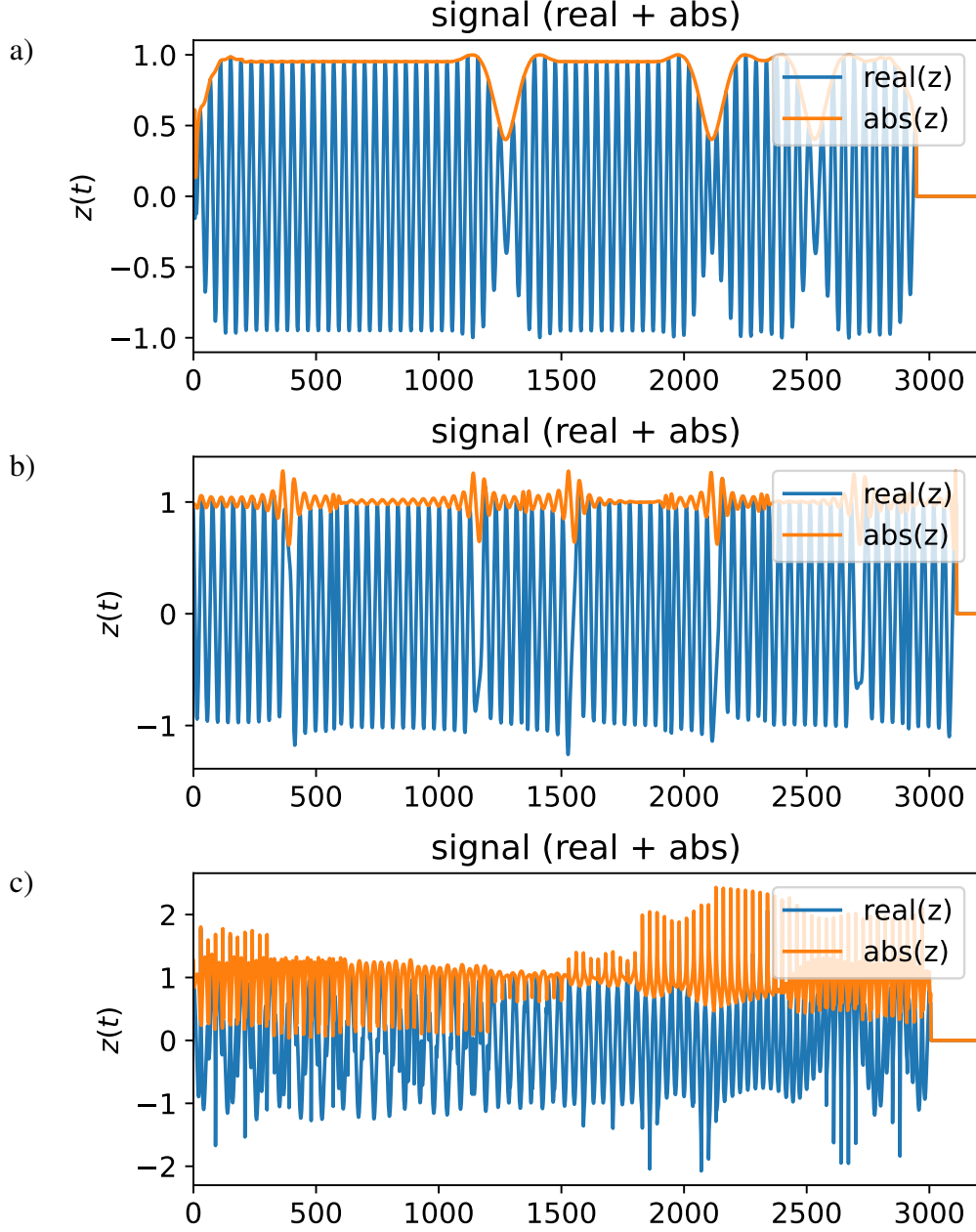


Fig. 2 Sample waveforms from the noise-free version of SIDLE. In the Barker 7 waveform, the finite switching time is the most prominent feature, while in the Nested Barker 16 and the Polyphase P1-100, the transients are most prominent. All three pulses have pulse widths of ~ 3000 time steps. a) Barker 7. b) Nested Barker 16. c) P1-100.

Another idiosyncrasy of the SIDLE dataset is that the samples of each waveform in the dataset use a small set of distinct pulse widths. For example, of the 11,000 samples of Barker 7, there are 184 distinct pulse widths, while the 11,000 samples

of P4 have 64 distinct pulse widths, which are generally longer than the Barker 7 pulse widths.

These idiosyncrasies are presumably due to design choices made by SIDLE developers, mimicking certain hardware features or system tendencies. For the purpose of developing general ML models that can be used for domain generalization, they are potential sources of bias. This is not a criticism of SIDLE—it is not promoted as a general dataset used for ML training. But the waveform-specific properties have prompted questions that have helped shape the design of PyRadarWF. Most importantly, which properties do the ML methods detect? Can they differentiate pulse widths or SNRs? If I work solely with SIDLE as my training platform and my inference data includes a Barker 7 waveform with a longer pulse width than is found in SIDLE, will the models detect a different radar modulation because of the pulse width? As the focus of this work is in generating flexible datasets for transfer learning and domain generalization, we seek to prepare PyRadarWF to handle such features, but not rely on them for identifying target properties.

2.2.2 The Emulated FRAME Dataset

The hardware used to generate and record the FRAME datasets employed in this study are described in Necsoiu et al.,⁶ and consist of a radar signal generator and real time signal analyzer, connected via cable. We will not display the emulated waveforms generated by the FRAME system. In general, we find that the FRAME waveforms appear much closer to the mathematical forms than SIDLE. While we cannot obtain noise-free signals, a 24-dB SNR is readily obtainable, and we can decrease the generation amplitude to approximate lower SNRs, covering the -15 to 15 dB range that is standard for SIDLE. Careful examination at 24-dB SNR shows that the FRAME subpulse boundaries are very near square, with very small switching time and transient orders of magnitude smaller than those in the SIDLE dataset.

FRAME generates pulse trains that have a fixed relationship between the pulse widths and the SNRs. Specifically, we set the signal generator to generate pulses from a fixed modulation class with widths uniformly sampled within a certain time range (effectively 1000 to 6000 time samples per pulse) and a decreasing series of amplitudes, which we estimate would yield approximate SNRs of 24, 14, 10, 6, 2, -2 , -6 , -10 , and -14 dB. The signal generator will then generate pulse trains

composed of nine pulses, each from the target modulation class and with the same pulse width, and with successive pulses having the same series of amplitudes, corresponding to the target SNRs given above. The rising edge of the pulses occur at a regular time interval, known as the *pulse repetition interval* (PRI). So, one pulse train could have nine Barker 7 pulses, each with pulse width 2431 time samples, and the individual pulses with amplitudes giving 24, 14, ..., -14 dB SNRs. The pulses in the next pulse train would be another nine Barker 7 pulses, with a different, random pulse width, but the same series of SNR values. When adapting the sampled data for use in ML, we use the 24-dB pulses for triggering—to determine pulse rise time and pulse width—and we use the other eight pulses for inference, recording with them the pulse width and the approximate SNR. The Barker and Nested Barker FRAME-generated waveforms used with WF-Mods09 and WF-Mods18 have randomly varying pulse widths, drawn from the same distribution as the PyRadarWF training data, with pulse trains exhibiting the relationship between SNRs and pulse width described above.

The main idiosyncrasy in our FRAME data is due to time constraints on access to the FRAME hardware; when creating the current FRAME dataset, we did not generate P-codes with random pulse widths. In the data used here, all FRAME-generated P-codes have the same pulse width of approximately 2898 time steps. We will address this in the future.

2.3 Data Import and Analysis Tools

The data import and analysis tools in PyRadarWF are not a standard set of signal analysis tools. They were developed prior to and in conjunction with the data generation tools, for the purpose of understanding, comparing, and contrasting the available SIDLE and FRAME data, and to reformat the data for use with PyRadarWF. The analysis tools find pulse rise times and pulse widths, approximate the input carrier frequency (which is often near, but not equal to zero), approximate the SNR, etc. The associated plotting tools plot time-domain, frequency-domain, using the fast Fourier transform (FFT) and Wigner–Ville distribution (WVD). In their current form, these tools are adequate to convert SIDLE and FRAME data to the PyRadarWF format, including the multiple labeling format, converting the signals to baseband, and determining trigger pulse rise times to aid in centering/offsetting pulses within the time window.

2.4 PyRadarWF Dataset Generation

The current philosophy behind PyRadarWF dataset generation is not to have a single, canonical dataset, but to have multiple datasets that allow us to probe how different waveform properties are handled by the ML architectures. As such, we have identified a list of properties to vary within the dataset, and set these using a user-modified Python script. The set of properties required to define a radar waveform is called the PDW, which we introduced above.

For domain generalization, we want the sample waveforms within a dataset to cover as broad a variety of possible parameters, but uniform in how the random properties are sampled, to ensure the ML model learns the correct features during training. So, we treat the PyRadarWF definitions as distributions to be followed, using the same distributions for all waveforms from all modulation classes.

2.4.1 Basic Dataset Generation Process

The specific process of how we go from dataset definitions to dataset or ML model is as follows:

1. The user modifies a Python script, `gen_datasetJSON.py`, to set the desired property ranges and distributions used to set the individual waveform PDWs, as well things like property units and human-readable labels for plotting.
2. When `gen_datasetJSON.py` is run, it generates a JSON file* that contains the specific definitions for the dataset. This JSON file will be stored as part of the dataset, pickled in the dataset's NumPy `.npz` file as its Python dictionary equivalent.
3. The dataset generator is run, using the JSON file as input.
4. The dataset is used either for ML training or inference.

While this tech report is not a user guide, we have included in the Appendix a section of code from `gen_datasetJSON.py` to show how these parameters are set, and the format for defining parameter distributions.

*JSON is a standard, human-readable text format used for data exchange. Python dictionary structures can be converted readily between Python and JSON, making this a straightforward means of parameter transfer.

The creation of the JSON file may seem like an extra step in this process. We have chosen to add this step to enable compact storage of dataset generation information, and to aid analysis of a given dataset. The same dataset can be regenerated at any time using the JSON file and the same version of PyRadarWF on the same hardware. If needed, the dataset can also be regenerated by extracting the JSON file from the dataset `.npz` file. (We note that the JSON file also includes the random number seed employed, and the default behavior is to create a new random seed each time a new JSON file is created.)

One important aspect of generating datasets for ML model training is that models tend to fit to the desired parameters better if all other parameters are randomized as much as possible. For example, we could add noise to the signals during dataset creation, or we could add the noise during model training, working from the parameter distributions. A disadvantage of the former choice is that it could cause the model to learn the specific noise patterns, leading to over-fitting on undesirable waveform properties. To date, we have chosen to make all such modifications to waveform properties during dataset generation, and plan to add in the near future the capability to vary as many parameters as computationally reasonable during training. For inference datasets, we plan to set all properties during dataset generation.

2.4.2 PyRadarWF PDW

For hardware systems, the exact parameters contained in the associated PDW depend on the system design and requirements. For example, a radar detection system may not be able to resolve all the parameters employed to generate a radar pulse, so it may have fewer parameters in its PDW than the generation system. Here, we have chosen as PDW parameters those necessary to create a reasonable representation of a SIDLE or FRAME radar pulse using PyRadarWF. All of these parameters may be varied according to defined random distributions to a given PyRadarWF dataset, with the exception of the number of time samples per PRI, N_s , which must be the same for a given dataset. In addition, many parameters may be designated to be set either during dataset construction or during ML training, an option which we discuss further below.

The final four entries in our PDW would not typically be found in a hardware PDW, but are necessary for approximating different hardware systems in our synthetic waveform generator. These are related to changing the pulse shapes from being

based on a mathematically defined square pulse to what is generated by actual hardware. A square pulse is discontinuous, by definition, but the physical quantity is not, and should at least be C^0 , not accounting for noise. In reality, we should expect there to be some finite time required for the system to switch the value of the envelope from 0 to 1, for example. There should also be some sort of transient caused by this switching. These properties are present in the SIDLE examples given in Fig. 2. We designate four separate values needed to define these properties, the time required for a system to switch the amplitude from 0 to 1, and three for defining the transient: frequency, amplitude, and decay constant. In a hardware radar system, all of these properties are influenced by the specific system hardware design and implementation, and can vary between systems built to the same specifications by the same manufacturer. As we are targeting domain generalization, we do not wish to simulate a specific system, but cover what would be a wide variety of systems. Thus, these parameters are also typically given as ranges of parameters that are then set during waveform construction or during ML model training.

We have found the following properties necessary to include in our PDW, defining each waveform; they are also summarized in Table 2.

Modulation class: The modulation class can be taken from either WF-Mods09 or WF-Mods18; these were discussed in greater detail above. We define all datasets in terms of number of waveforms per modulation class. Typically, we use 10 times as many waveforms per class for training as for inference.

N_s = **Number of time samples:** There are three variables involved with determining the time sampling, any two of which determine the other: PRI, sampling frequency f_s , and number of samples in the PRI, N_s .

$$N_s = \text{PRI} \times f_s . \quad (1)$$

We work in N_s , since all waveform samples to have the same length for the ML architectures employed, both for training and inference. However, for plotting waveforms, we have retained the PRI and f_s structure, including time units.

We have chosen to work with $2^{13}=8192$ time samples per PRI, as this allows efficient use of the FFT, and have chosen a PRI on FRAME to give roughly

8400 samples. The number of time samples per PRI in the noise-free SIDLE dataset is 11,000. We truncate the SIDLE data to 8192 time samples and exclude all pulses from SIDLE that have pulse widths wider than 8192.

Pulse width: Pulse width can be measured in time units or number of samples. We work with both, and choose the one most appropriate to the situation—typically, time units for plotting and number of time samples for ML calculations. Most likely, this parameter will not be varied during ML model training.

As discussed above, in SIDLE, a given modulation class has its pulse widths drawn from a small set of choices. FRAME can be configured to give random pulse widths within a specific range, and then varies these based on pulse trains. For PyRadarWF datasets, we typically set the desired range or ranges for the pulse widths, and then sample uniformly within that range or ranges. Most pulse widths in the SIDLE and FRAME data are less than 8192, and we drop any sample waveforms with pulse widths longer or shorter than the range of pulse widths chosen for a given set of calculations.

SNR: For our synthetic data, we can set the SNR based on the noise-free signal. For the FRAME system we must estimate the SNR from high amplitude signals and then extrapolate to lower signal amplitudes. Variation of noise during ML model training is straightforward and will be used in future calculations.

In our current testing, we use the same functions for adding noise to the SIDLE and PyRadarWF datasets. To date this has been additive Gaussian white noise. There is no guarantee that the noise in FRAME is from the same distribution.

Carrier frequency offset: As discussed above, radar waveforms employ two carriers with the same frequency and (ideally) amplitude, out of phase by a quarter wave. Since we assume that we work in baseband, the offset from baseband is more relevant. Variation of carrier frequency offset during ML model training is straightforward and will be used in future calculations.

The SIDLE and FRAME datasets each have a given carrier frequency that is typically uniform within the set of modulations. To date, we have modulated these signals as near to baseband as we can, and have worked solely in baseband with PyRadarWF. We have not accounted for any errors in our baseband

conversions.

Carrier phase: The carrier phase in both SIDLE and FRAME datasets is, by our measurements, a uniform random variable,

$$\Phi \in [0, 2\pi) .$$

We have done the same for the PyRadaWF calculations used here. However, it is possible to discretely set these values in PyRadarWF, if needed. We note that carrier phase is still important in baseband, as the real and imaginary parts of the signal must still be a quarter wavelength out of phase. Variation of carrier phase during ML model training is straightforward and will be used in future calculations.

Pulse rise time: The pulse rise time is the time within the PRI when the pulse starts. Variation of pulse rise time during ML model training is straightforward and will be used in future calculations.

For our current calculations, we chose this for SIDLE and PyRadarWF such that the pulse may, with uniform distribution, be anywhere within the time window, provided the entire pulse is within that window. For FRAME, when generating the data, we specifically included the 24-dB trigger pulses to help identify where the pulse lie, as well as estimate the SNR. Based on this triggering, when we prep the FRAME data for use as an inference dataset, we ensure that the pulses lie entirely within the time window, typically near the center.

Switching time: The amount of time required for a change in pulse value with magnitude 1. The net result is a ramp at the leading edge of the square pulse, rather than a sharp edge. Variation of switching time during ML model training may be difficult to implement.

Examination of the SIDLE data shows that these ramps vary depending on modulation class. In FRAME, these switches are very quick, happening over roughly four sample points.

Table 2 PDW for PyRadarWF datasets. These are the properties needed to define the specific waveforms used in the dataset. The column, “Can be varied?” denotes properties that should be possible to vary in training. We have not yet implemented this.

Property	Input type	Can be varied?
Modulation class	From WF-Mods09/18	No
Number of time samples	Single integer	No
Pulse width	Integer value from range	No
SNR	Real number from range	Yes
Carrier frequency offset	Real number from range	Yes
Carrier phase	Real number from range	Yes
Pulse rise time	Real number from range	Yes
Switching time	Real number from range	Yes
Transient frequency	Real number from range	Yes
Transient amplitude	Real number from range	Yes
Transient decay time const.	Real number from range	Yes

Transient frequency: Transients are caused by multiple factors, but tend to be related to the particular solution of an ordinary differential equation and the carrier frequency. Thus, while we work in baseband, we need to estimate the carrier frequency, to be used in the transients added to the waveforms. Variation of transient frequency should be used in future calculations.

Transient amplitude: This varies considerably within the SIDLE data, and tends to be much smaller in the FRAME data. Variation of transient amplitude should be used in future calculations.

Transient decay time constant: The transient decay behavior tends to be a decaying exponential of form $e^{-t/\tau}$, where τ is the decay time constant. The decay time constant varies considerably within the SIDLE data, and appears to be quite short in the FRAME data. One design choice we have made in PyRadarWF is to treat this as a decaying exponential to a certain amplitude multiplier (typically 0.2), and then truncate it using a parabola, fit to ensure the decay is C^1 . Variation of the transient decay time constant should be used in future calculations.

2.5 PyRadarWF Waveform Generation

We briefly describe here the process of generating individual radar waveforms, based on the specified properties. Within the dataset, we specify the number of waveforms per modulation class, and generate these in a sequence, drawing random variables from the appropriate distributions for each waveform properties. We will illustrate this by describing the creation of a Barker 7 pulse, including a series of figures.

First, we define the waveform amplitude function $A(t)$, based on N_s , modulation class, pulse width and switching time. Note that $A(t)$ is real for the Barker and Nested Barker classes, but not for the P-codes. This is shown in Fig. 3a. Next we add the transients, based on the transient parameters, as shown in Fig. 3b. Finally, we add a time shift to the pulse rise time, a carrier frequency offset, a carrier phase, and noise with amplitude sufficient to obtain the desired SNR. We normalize this signal such that its standard deviation is 1 over the entire time period. This final signal is shown in Fig. 3c.

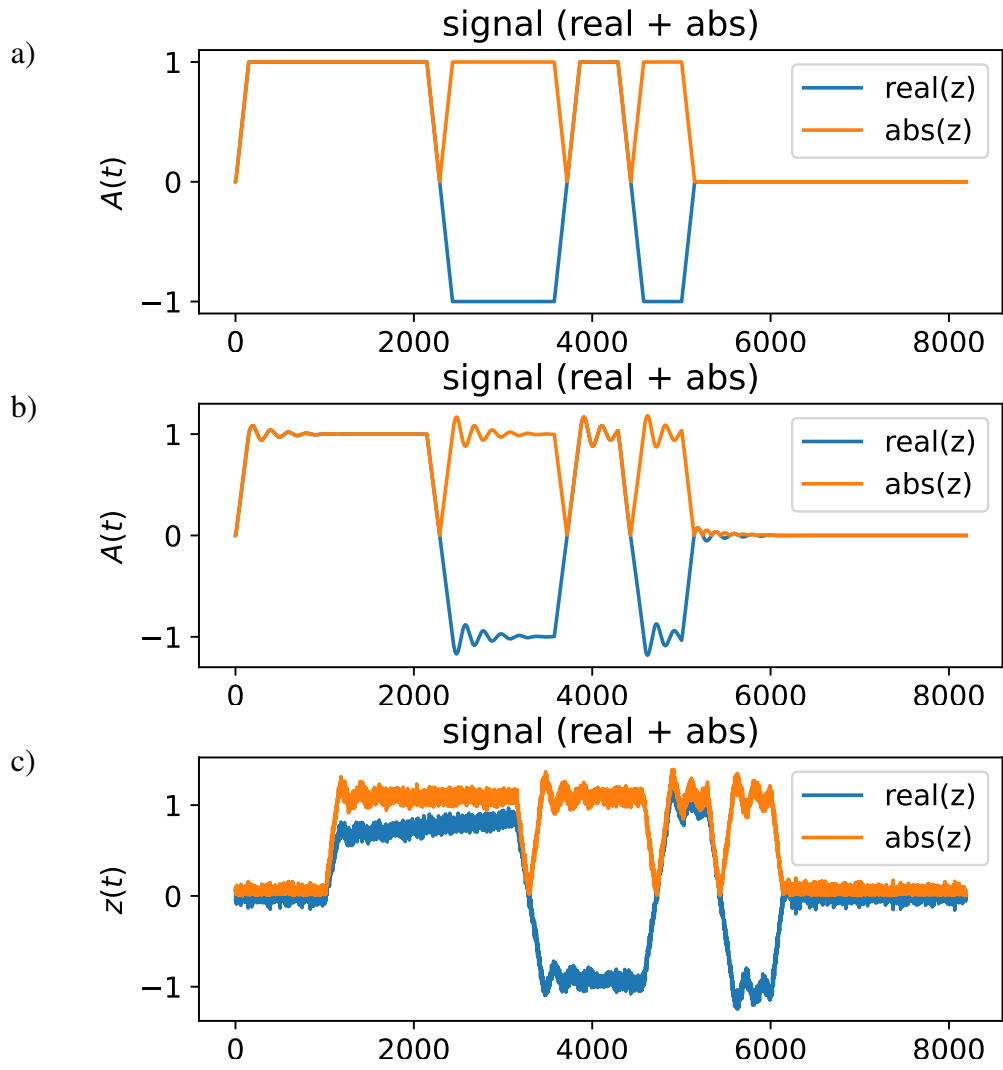


Fig. 3 Barker 7 waveform construction process used by PyRadarWF. a) The Initial envelope created, including slow switching time. b) Transients are added based on switching times. c) Rise time, frequency, and phase offsets are added; noise is added to give 24-dB SNR. Finally, the signal is normalized to give a standard deviation of 1 for entire signal.

3. ML Methods

We currently include two common NN architectures—convolution-based and attention-based—with implementations based on ResNet⁹ and a transformer-based architecture.¹¹ ResNet has largely been developed for 2-D data, often with different color channels (e.g., red, green, and blue). Our implementation here adapts these models to 1-D, and uses two “color” channels, for the real and imaginary parts of the IQ signal, or its FFT. Because the FFT data is the same length and type as the IQ data, it can be dropped in as a replacement. We have also experimented with using four-channels, with the IQ and FFT effectively concatenated (one real and one imaginary channel each for IQ and FFT). From here on, we refer to these as IQ, FT and IQ+FT, respectively, and stress that the data are not independent. In all cases, the FT data used is the (invertible) FFT of the corresponding IQ data. We have also looked at Fourier neural operators,¹² but thus far do not find any advantage over using the base models with FFT data, and do not discuss these further. In the balance of this section, we describe the ResNet and transformer NN architectures.

3.1 ResNet

Our first network is based on the common convolutional NN (CNN) ResNet.⁹ This network uses a series of convolutional layers (e.g., 18 layers in ResNet-18) with increasing channel width. ResNet utilizes *residual learning*, which applies skip connections every two layers to preserve gradients (see Fig. 4). This network performs well on a large variety of tasks, including classification on the SIDLE dataset.^{10,13} We use a 1-D variant of ResNet-18 in our experiments. It is composed of four main blocks, each with [64, 128, 256, 512] channels and each with [6, 8, 12, 6] layers.

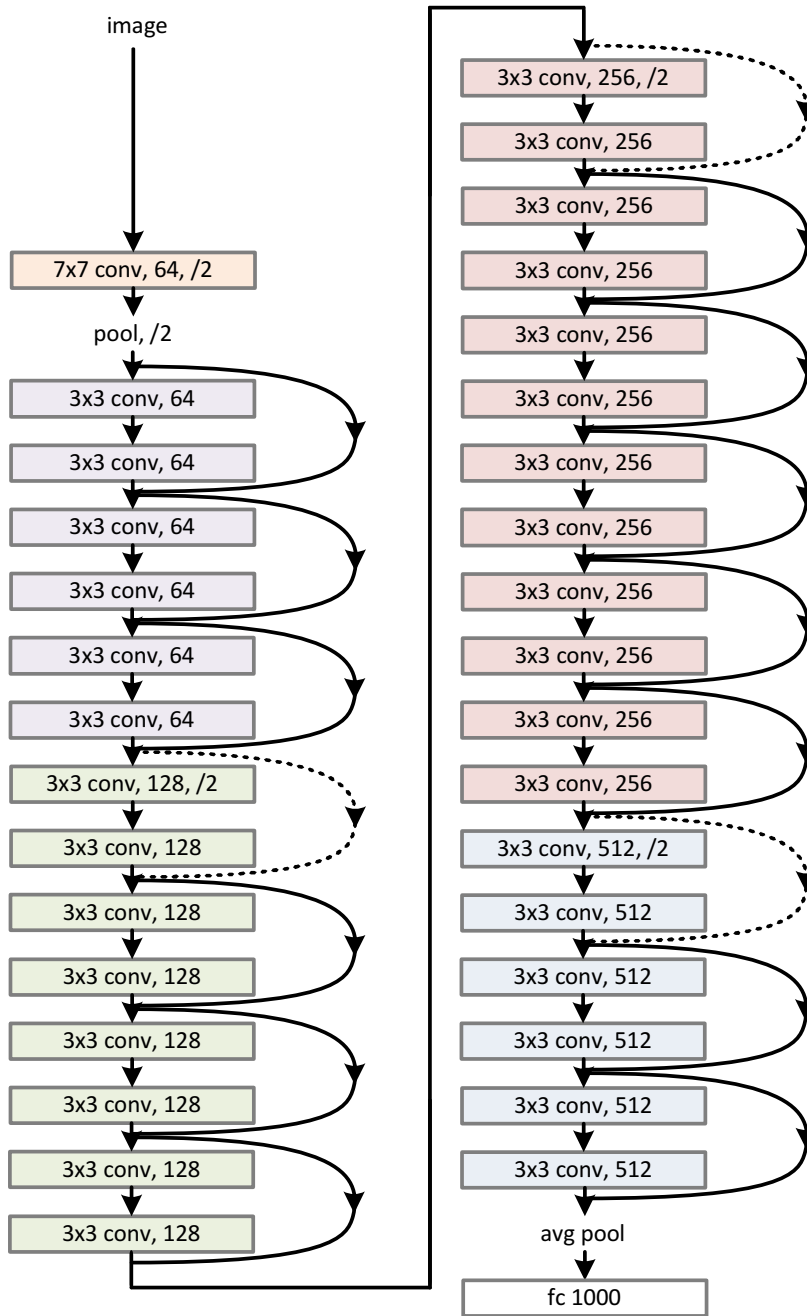


Fig. 4 ResNet-34. Architecture diagram of ResNet-34 adapted from He et al.⁹

3.2 Transformer

Our second network uses a transformer-inspired architecture as in Ziemann et al.¹⁰ An overview diagram of this network can be seen in Fig. 5.

This network uses the transformer encoder layer first proposed in Vaswani et al.¹¹ and detailed in Fig. 5. This layer uses self-attention to compute the “attention” between input features. Attention is computed with a set of query (Q), key (K), and value (V) matrices to quantify the relationship between input features as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where d_k is the dimension of the Q and K matrices. This is known as scaled dot-product attention Vaswani et al.¹¹ The transformer furthers this with *multi-head* attention, which applies N parallel self-attention “heads” to learned projections of the Q , K , and V matrices. This allows the model to learn many attention relationships for a given input across different subspaces, which improves performance compared to a single attention head. This operation can be described as

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_N)W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \end{aligned}$$

where W_i^Q , W_i^K , W_i^V are learned projection matrices and W^O is the learned output weights.

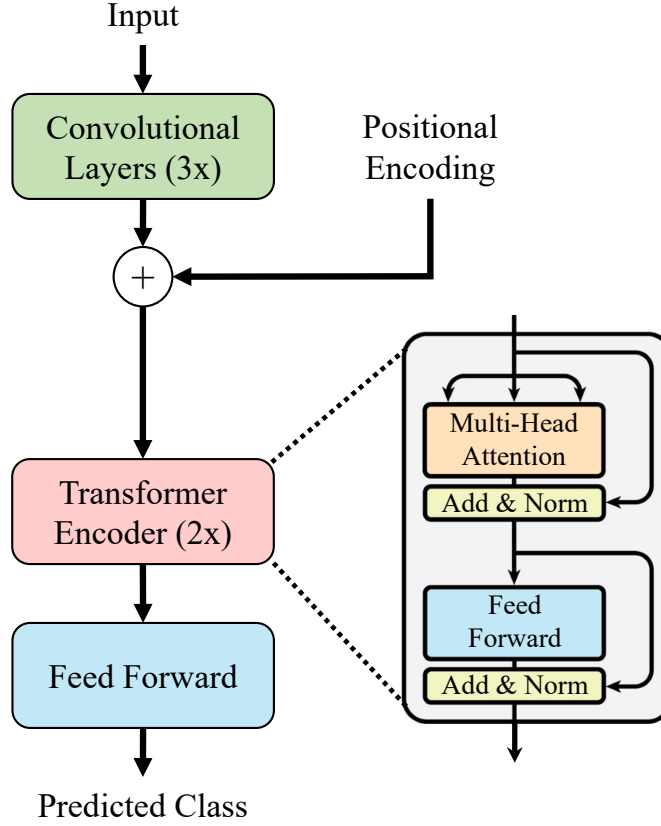


Fig. 5 Transformer architecture. Overview of our transformer-based NN architecture with detailed transformer encoder layer.¹¹

Our NN begins with three convolutional layers, each followed by a ReLU activation function, that encode the raw input into a compact latent representation. Each of these layers utilizes a kernel size of 11, stride of 2, and 64 output channels. This compact representation reduces the memory requirement of the self-attention layers by reducing the size of its inputs. We then sum a sine/cosine based positional encoding to the compact representation

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}).$$

This is necessary for the transformer encoder layer to retain information on the temporal ordering of the input features. The encoded features are then passed to two transformer encoder layers, each with four attention heads, and then to a densely connected layer to generate the final output. The cross-entropy loss is then calcu-

lated and back-propagated to train the network. We use 10% dropout within the transformer encoder layer and before the final dense layer to prevent overfitting.

4. ML Results

In this section, we give the results for classification of the three properties: modulation class, pulse width, and SNR. We have used PyRadarWF as our training data. We have inferred on separately generated PyRadarWF datasets, as well as the relevant data taken from the SIDLE dataset and data we have generated using FRAME. Because of the relative ease of generating new PyRadarWF data, the PyRadarWF datasets used for training and inferring WF-Mods09 and WF-Mods18 are distinct—four different files with different random seeds and different numbers of waveforms, generated using the same random distributions. All data series used in the calculations were 8192 ($= 2^{13}$) time steps in length. In the case of SIDLE, we have started with the noise-free version of SIDLE data, randomly shifted the pulse in time to lie somewhere, entirely within the time window, then added noise and normalized using the same functions used on the PyRadarWF data. In the case of the FRAME data, we have mostly centered the pulses within the time window and normalized in the same manner. We use the triggering pulses in the data to determine pulse width and normalization for each data series. Any waveform from any source (PyRadaWFr, SIDLE, or FRAME) with pulse width outside of the window of 1000 to 6000 time steps were omitted. For we train on 10,000 waveforms per modulation class (90,000 total for WF-Mods09; 180,000 total for WF-Mods18) and infer on 1000 waveforms per class (9000 total for WF-Mods09; 18,000 total for WF-Mods18).

Currently, our testing follows a combinatorial approach. As with any combinatorial study, this means the number of results grows quickly. The different sets of properties are listed in Table 3. The total accuracy rates (defined here as correct classifications divided by total classifications) for WF-Mods09 and WF-Mods18 are given in Tables 4 and 5, respectively. There were 36 final training calculations, one each for IQ, FT, and IQ+FT data for each column of Tables 4 and 5, and 90 inference calculations, one for each entry in Tables 4 and 5. Each entry in these tables can be further interpreted by referring to its corresponding confusion matrix (90 total). We provide sample confusion matrices as figures in the discussion below to help interpret the results in Tables 4 and 5.

Table 3 Different training and inference properties of datasets and their application in ML. The only restriction on the combinations is that the SIDLE dataset does not contain all the modulations in WF-Mods18. For these results, we have trained only on PyRadarWF data, and inferred on three fixed datasets drawn from SIDLE or generated with PyRadarWF or FRAME. Within these constraints, we have computed all possible combinations. The total accuracy rates for all combinations are tabulated in Tables 4 and 5.

Property type	Set of options
Waveform set	{ WF-Mods09, WF-Mods18 }
Waveform property	{ Modulation class (WF), Pulse Width (PW), SNR }
Data type	{ IQ, FT, IQ+FT }
ML Architecture	{ ResNet18, Transformer }
Inference data	{ PyRadarWF, FRAME, SIDLE }

Table 4 Accuracy rate for each inference calculation for the WF-Mods09 datasets.

Inference dataset	Modulation class		Pulse width		SNR	
	ResNet18	Transf.	ResNet18	Transf.	ResNet18	Transf.
PyRadarWF IQ	0.889	0.925	0.801	0.866	0.913	0.951
PyRadarWF FT	0.916	0.827	0.824	0.667	0.916	0.854
PyRadarWF IQ+FT	0.916	0.635	0.822	0.681	0.910	0.886
FRAME IQ	0.202	0.789	0.572	0.661	0.245	0.430
FRAME FT	0.685	0.794	0.673	0.159	0.104	0.119
FRAME IQ+FT	0.619	0.689	0.687	0.114	0.085	0.136
SIDLE IQ	0.731	0.563	0.736	0.858	0.918	0.948
SIDLE FT	0.846	0.761	0.723	0.581	0.901	0.854
SIDLE IQ+FT	0.847	0.627	0.713	0.591	0.893	0.873

Table 5 Accuracy rate for each inference calculation for the WF-Mods18 datasets.

Inference dataset	Modulation class		Pulse width		SNR	
	ResNet18	Transf.	ResNet18	Transf.	ResNet18	Transf.
PyRadarWF IQ	0.853	0.877	0.852	0.889	0.940	0.952
PyRadarWF FT	0.853	0.769	0.838	0.724	0.931	0.915
PyRadarWF IQ+FT	0.853	0.784	0.848	0.723	0.915	0.912
FRAME IQ	0.198	0.729	0.498	0.760	0.335	0.460
FRAME FT	0.656	0.639	0.598	0.039	0.070	0.161
FRAME IQ+FT	0.676	0.722	0.665	0.038	0.199	0.230

4.1 Discussion

Tables 4 and 5 show the overall accuracy rates (fraction of correct inferences out of the total) for datasets containing modulation classes from WF-Mods09 and WF-Mods18, respectively. The results are consistent between the two tables. In some cases, the results for SIDLE and FRAME are quite close, suggesting that for those properties the input data are similar, or PyRadarWF covers the characteristics of both sources. On the other hand, the accuracy rates for inference on PyRadarWF data suggest that the data or ML methods could be improved or at least better converged by adjusting parameters or using more data.

In Fig. 6 we show the confusion matrices for modulation classifications for FRAME

IQ data using the transformer NN, showing very comparable results.

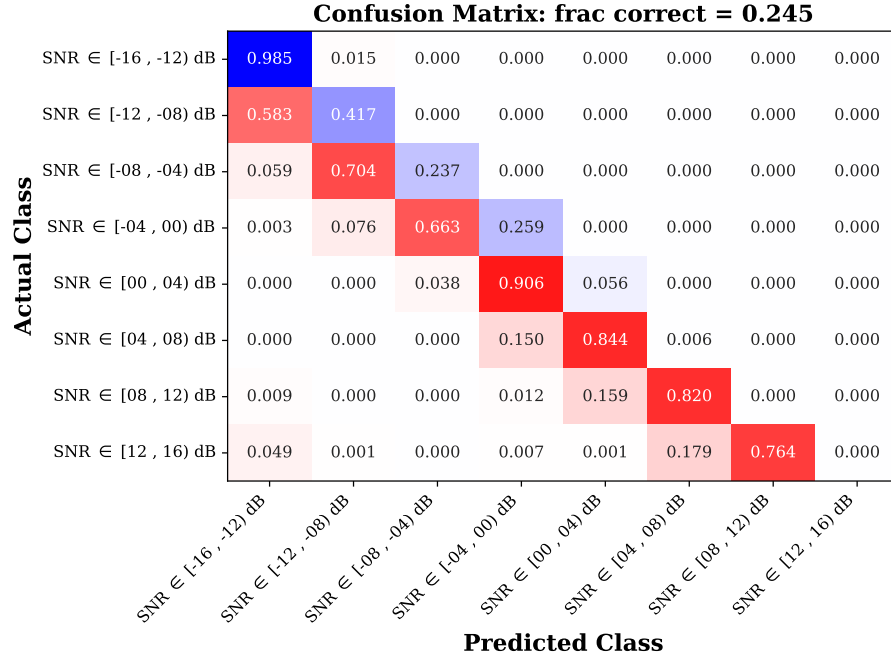
A comparison of the entries in Tables 4 and 5 showcases a major strength of the approach we have taken with this research; the diversity of data sources and properties examined highlight strengths and weaknesses of the different ML architectures and the PyRadarWF datasets. These will help guide further development on both the data and ML architecture fronts. Many ML studies employ a dataset with labels only for their targeted task, such as modulation classification. This limits their ability to explore the nuances of feature extraction for such datasets and to understand the failure modes of different approaches.

Overall, the similar values found between the similar entries in Tables 4 and 5 suggest that these results are generally systematic. For example, inference results on the FRAME IQ data for the modulation class using ResNet18 are anomalously bad in both tables. If we had only classified on modulation type, we might think that this is due to poor convergence or an error somewhere in data preparation or model training. Alternatively, we might conclude that the transformer NN is a stronger method for this problem. The confusion matrices corresponding to similar entries in the two tables show similar results as well, despite the significantly greater complexity of the WF-Mods18 datasets.

Two more observations may quickly be drawn from the for inferences of modulation class on the FRAME IQ, contrasted with their paired results for FRAME FT data. First, the differences between the IQ and FT results for ResNet18 suggest there is something particular about how ResNet18 handles the IQ data. Second, the stronger performance of ResNet18 on the SIDLE IQ data suggests this is FRAME-specific.

More research is required, but other results summarized in Tables 4 and 5, along with observations found in the literature, provide some direction on what may be happening here. In the SNR classification results, it is clear that something is different about FRAME’s SNR classifications. Because we used the same functions to add noise the SIDLE inference dataset and the PyRadarWF training and inference datasets, it should not be surprising that the SNR classifications for SIDLE data are similar to the corresponding ones for PyRadarWF data. In fact, the SNR accuracy rates are the only ones consistently above 0.85 for inference on either SIDLE or PyRadarWF data. However, SNR classification on the FRAME dataset can be less 0.1. Turning to the confusion matrices for SNR classification on FRAME IQ data

a)



b)

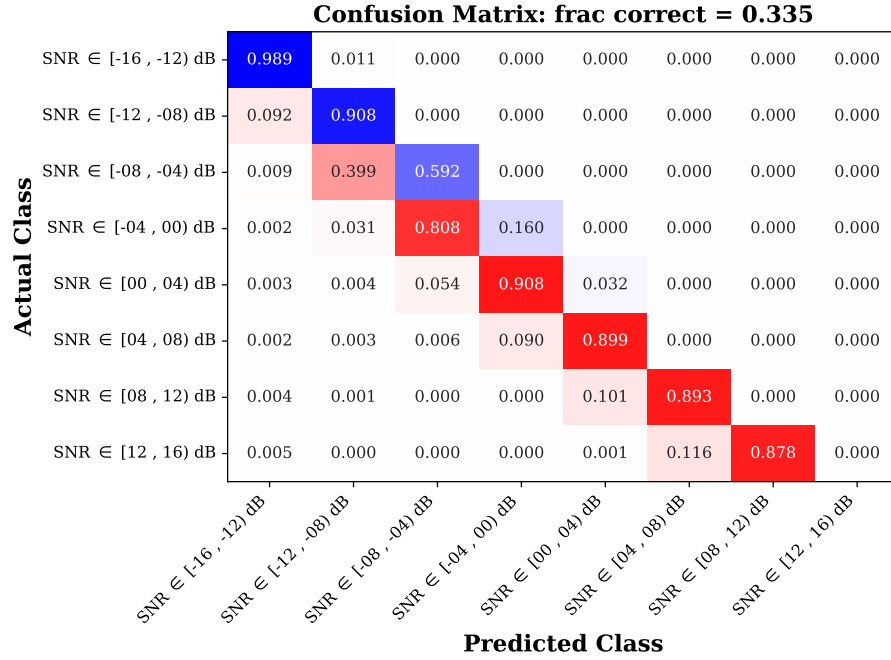


Fig. 6 Comparison of confusion matrices for same properties and similar data for WF-Mods09 and WF-Mods18. In both cases, datasets generated by PyRadarWF are used for training, with the datasets employing the same parameters and random distributions for data generation, though different random seeds, aside from the additional classes in WF-Mods18. For inference, we use FRAME IQ data, with SNR as the target property and ResNet18 as the ML architecture. a) WF-Mods09. b) WF-Mods18.

shown in Fig. 6, one possible interpretation would be that we overestimated the SNR in the FRAME data. This may be true and merits further investigation. However, the confusion matrices for inference on the SNRs of FRAME FT data, given in Fig. 7 show that ResNet18 absolutely does not work for this data. Even in the limited portions of the confusion matrix where ResNet18 does not default to the highest SNR, it underestimates the noise more than it does for the IQ data.

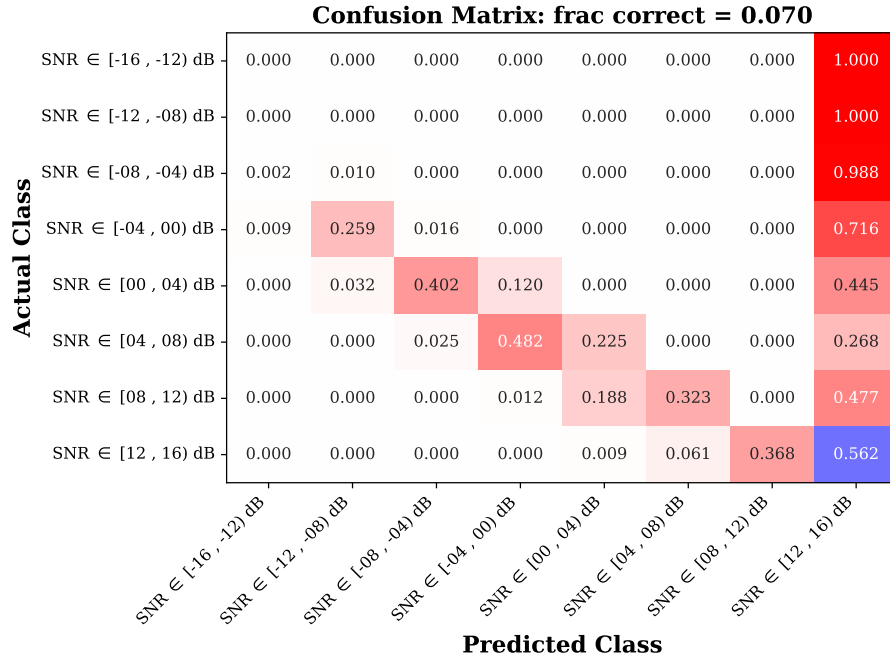
In the literature, Hazza et al.¹⁴ have shown that the performance of other signal classification methods can be strongly impacted by the choice of noise distribution. Similarly, our prior work¹⁰ demonstrates the fragility of ResNet18 and transformer architectures under adversarial attacks, where tiny perturbations to the model inputs can severely impact the classification accuracy. These papers support our findings in this work: the models are highly sensitive to the training noise distribution, and further work should explore methods of improving robustness to changes in the noise distribution.

Turning to the question of whether using the combination IQ+FT data can improve results, the full set of results in Tables 4 and 5 suggest that typically either or both of the IQ or FT data get better results. Our hope with IQ+FT was that it would find the best of both data types, but this is not the case. In Figs. 8—10 we show the confusion matrices for modulation classifications on FRAME IQ, FT, and IQ+FT data using the transformer NN trained on WF-Mods18 PyRadarWF data. Comparing the results in these confusion matrices we see that when classifying the Barker 7, 11, and 13 modulations, using IQ+FT data does better than the individual IQ and FT data. However, in some other cases, the use of combined IQ+FT data appears to average some IQ and FT results, and in other cases it does worse than either one.

Another interesting result relates to the P-codes with the same number of chips. Looking back at Figs. 8—10 (and at the other confusion matrices for modulation class classification on WF-Mods18), it is somewhat common for the classifiers to confuse P-codes of the same length such as P1-100 for P2-100, and the reverse. Interestingly, though, the classifiers do not typically confuse any 16 chip P-code with a Nested Barker 16.

Classifications of ranges of values, such as the pulse width or SNR calculations, are, by their nature, noisy. For example, if there is finite switching time, how does one define the pulse width? Unsurprisingly, a significant fraction of the incorrect infer-

a)



b)

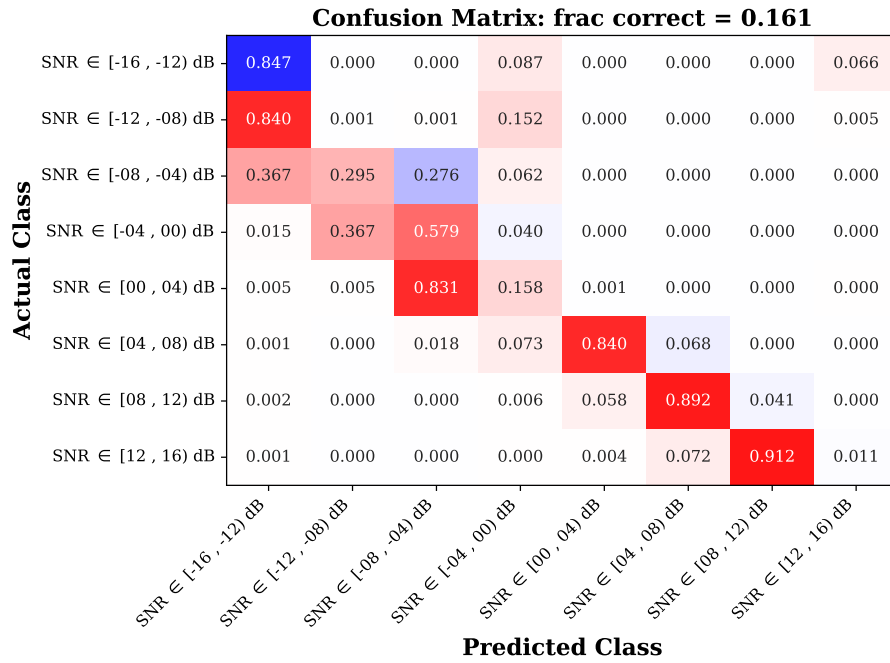


Fig. 7 Comparison of confusion matrices for inference on FRAME FT WF-Mods18 data, with ResNet18 and transformer NN architectures. a) ResNet18 b) Transformer.

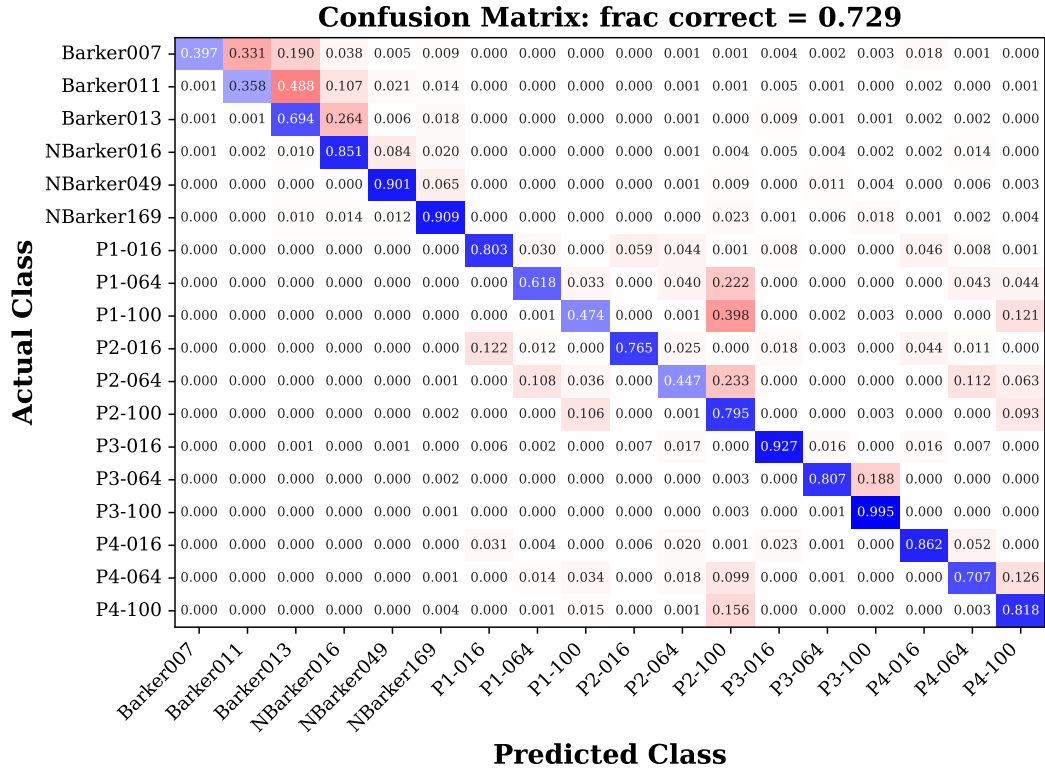


Fig. 8 Modulation classification of FRAME IQ data inferred on a transformer NN model trained on PyRadarWF IQ data.

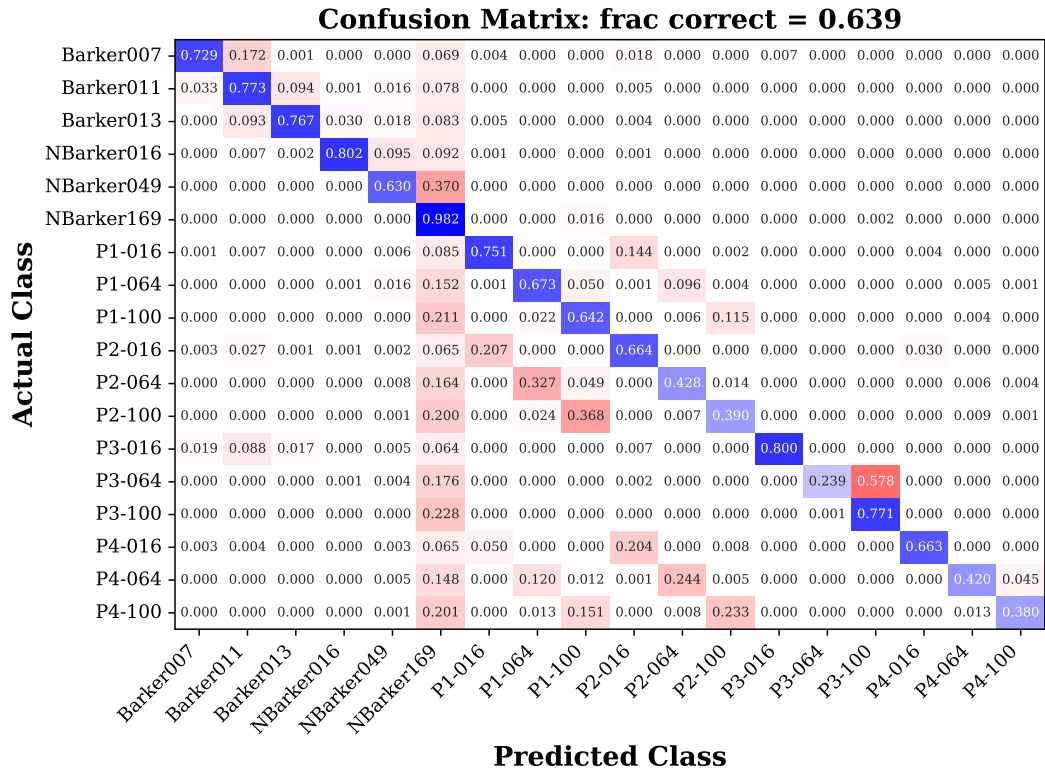


Fig. 9 Modulation classification of FRAME FT data inferred on a transformer NN model trained on PyRadarWF FT data.

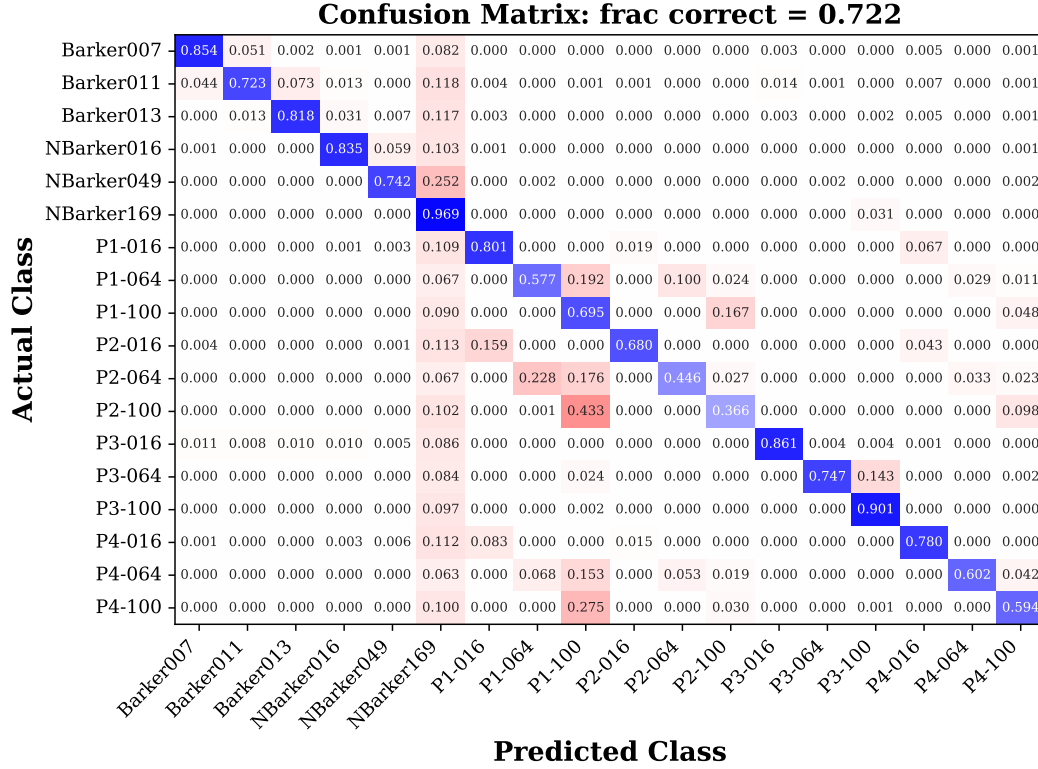


Fig. 10 Modulation classification of FRAME IQ+FT data inferred on a transformer NN model trained on PyRadarWF IQ+FT data. Comparing these classification results with the results found in Figs. 8 and 9 show that the IQ+FT combination is not yet a reliable improvement over the IQ or FT results alone.

ences identify the bordering regions, as seen in Figs. 6 and 11. This suggests that regression would work better for pulse widths and SNRs, rather than classification of ranges.

The final area of concern we address here is that many of the inference accuracy rates on PyRadarWF data are less than 0.9. The most likely explanation is that we vary too many parameters for the number of waveforms used in training. Recall that four PyRadarWF datasets employed in training and inference for WF-Mods09 and WF-Mods18 each use different random seeds and set all the variable properties prior to training the ML models. The simplest way to improve these results may be to train with more data. In the long run, varying as many of the waveform parameters as possible during training may do the most to improve the quality of these results.

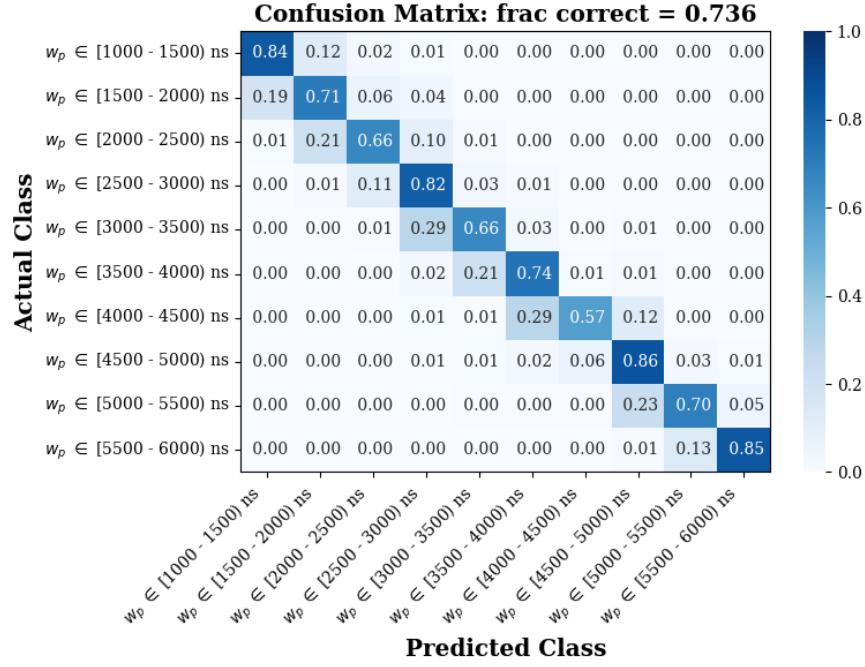
5. Conclusions and Future Direction

We have described and demonstrated the functionality of the new PyRadarWF dataset generator. PyRadarWF was developed to generate datasets and ML methods to enable more efficient and capable radar systems. We have described the three primary functions of PyRadarWF: radar dataset generation, radar data import and analysis, and 1-D ML architectures suited to IQ RF signals and their FFTs. Finally, we trained ML models on PyRadarWF data to run inference on data PyRadarWF and from two other sources: SIDLE, another synthetic radar dataset, and FRAME, a hardware radar emulator.

Our approach to building datasets moves beyond generating a standard dataset and training for a single data property, such as modulation classification. PyRadarWF is designed to provide distributions of datasets with the goal of using these datasets for domain generalization and transfer learning. The structures of the individual waveforms generated by PyRadarWF are derived from the basic mathematical models for these waveforms and an analysis of the SIDLE and FRAME datasets employed, but not the data from either SIDLE or FRAME. The data in each PyRadarWF dataset are labeled for three key properties in a radar code description, the modulation class, the pulse width and the SNR. The structure of the data handling is such that it can allow additional properties to be added in a straightforward manner.

Section 4 shows the strength of the flexible approach used in the design of PyRadarWF. The flexibility in dataset construction and labeling allow us to test how well the ML

a)



b)

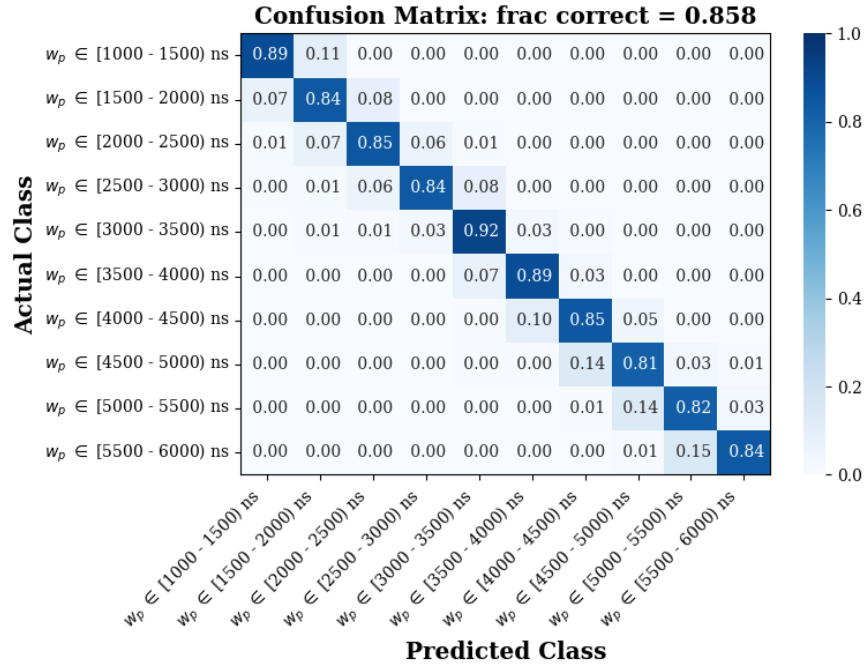


Fig. 11 Illustration of classification of pulse width ranges. It is common for signals to be classified as having a pulse width from a neighboring range. These inference calculations use WF-Mods09, SIDLE IQ data. a) ResNet18. b) Transformer NN.

architectures work to classify the waveforms for the differently labeled properties, modulation class, pulse width, and SNR. We use two different sets of modulation classes: a set of 9 classes and a set of 18 classes, WF-Mods09 and WF-Mods18. Finally, we train and infer on the IQ data, FT data (the Fast Fourier Transform of the IQ data), and a pairing of the IQ and FT data. In total we train 36 different ML models based on 2 different architectures, and run 90 different inference calculations.

The results are both consistent between the WF-Mods09 and WF-Mods18, and distinct between different properties, different forms of the data (IQ, FT and IQ+FT), and ML architectures, showing that the two ML architectures have different strengths and weaknesses, and that the results are different between the SIDLE and FRAME datasets. In many cases we have results of 0.7 to 0.95, which are reasonable results for domain generalization, while at other times our results are 0.1 or below. This gives a much more nuanced portrait of the ML architectures employed than many ML studies, which target a single dataset and a single property of that dataset. Here, we find that each ML architecture has strengths and weaknesses, although in some cases, both struggle with certain aspects of the data. This is especially true of the SNR and pulse width classifications of the FRAME data, which may have a different noise distribution than the additive Gaussian white noise assumed in the PyRadarWF and SIDLE datasets.

Moving forward, the flexibility of PyRadarWF enables a number of future tests, allows us to generate larger training datasets, if needed, or to vary certain aspects of the training data during training, including but not limited to the noise distribution, the range of frequency offset from baseband, and the variety of transients added to the signals. In addition to these, we plan to explore the use of saliency methods to identify sources of disagreement between the trained models and the inference datasets.

6. References

1. Wang J, Lan C, Liu C, Ouyang Y, Qin T, Lu W, Chen Y, Zeng W, Yu PS. Generalizing to unseen domains: A survey on domain generalization. *IEEE Transactions on Knowledge and Data Engineering*. 2023;35(8):8052–8072.
2. Rigling BD, Roush C. ACF-based classification of phase modulated waveforms. In: 2010 IEEE Radar Conference; p. 287–291.
3. Pavy AM, Rigling BD. Phase modulated radar waveform classification using quantile one-class SVMs. In: 2015 IEEE Radar Conference (RadarCon); p. 0745–0750.
4. Pavy A, Rigling B. SV-Means: A fast SVM-Based level set estimator for Phase-Modulated radar waveform classification. *IEEE J Sel Top Signal Process*. 2018;12(1):191–201.
5. Chakravarthy RV, Liu H, Pavy AM. Open-Set radar waveform classification: Comparison of different features and classifiers. In: 2020 IEEE International Radar Conference (RADAR); p. 542–547.
6. Necsoiu M, Ziemann M, Kraczek B, Diehl W. Domain generalization via data augmentation for machine learning-based radar pulse classification. DEVCOM Army Research Laboratory (US); 2023. Report No.: ARL-TR-9846.
7. Barker RH. Group synchronizing of binary digital systems. In: *Communication Theory*. Butterworth; 1953. p. 273–287.
8. Levanon N, Mozeson E. *Radar signals*. Wiley–Interscience; 2004.
9. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*;;
10. Ziemann MR, Metzler CA. Transformers for robust radar waveform classification. In: 2022 56th Asilomar Conference on Signals, Systems, and Computers; p. 759–763.
11. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser ŁU, Polosukhin I. Attention is all you need. In: Guyon I, Luxburg UV, Bengio S,

Wallach H, Fergus R, Vishwanathan S, Garnett R, editors. Advances in Neural Information Processing Systems; Vol. 30; Curran Associates, Inc.; 2017.

12. Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A, Anandkumar A. Fourier neural operator for parametric partial differential equations. <https://arXiv.org>; 2021. Report No.: 2010.08895.
13. Wharton M, Pavy AM, Schniter P. Deep neural networks for radar waveform classification. In: 2021 55th Asilomar Conference on Signals, Systems, and Computers; p. 1623–1627.
14. Hazza A, Shoaib M, Saleh A, Fahd A. Robustness of digitally modulated signal features against variation in HF noise model. EURASIP J Wirel Commun Netw. 2011;2011(24):1–12.

Appendix. Input Format for Dataset Generation

Although this report is not intended as a user guide for PyRadarWF, we include here a sample of an entry from `gen_datasetJSON.py` to demonstrate how parameter distributions are handled by PyRadarWF. In this example, we show the Python dictionary that sets the signal-to-noise ratio (SNR) parameters and discuss how these are used by PyRadarWF. Output associated with this entry can be seen in Fig. 6, including the ranges defined, the number format string, and the units label “dB.” Here is the listing:

```
'snr'      : {'randType'      : 'random_sample',
              'range_minMax'  : [[-16., -12.],
                                [-12., -8.],
                                [-8., -4.],
                                [-4., 0.],
                                [0., 4.],
                                [4., 8.],
                                [8., 12.],
                                [12., 16.]],
              'defaultValue'  : None,
              'bSetInTraining': False,
              'numFmtStr'     : 's%2.2d',
              'printUnits'    : 'dB',
              'description'    : 'Signal-to-noise ratio',
              'symbolicLabel' : 'SNR',
            },
```

The fields in this entry are set as follows:

- 'randType'** This sets how this particular value may be determined. The choices for this entry typically correspond to the name of the random number function from NumPy used to generate the values, and to the entry in this Python dictionary that provides the information needed to define the distribution (e.g., 'range_minMax' for 'random_sample):
- 'choice'** A value number chosen from the list 'fixedValues'.
- 'normal'** A normally distributed random number, with mean and standard deviation set by 'meanStd'.

- '**random_sample**' A uniformly distributed random number, taken from the ranges given by '**range_minMax**'.
- '**single_value**' A single value is given in the entry '**singleValue**'. For example, when working solely in baseband, the carrier frequency offset is set to the single value 0.
- '**range_minMax**' A list of numerical ranges setting both the range of values for random sampling and the bins employed in classification. This specific entry is dependent on the '**randType**' selected, as discussed under '**randType**'.
- '**defaultValue**' If, for some reason another value is not set using the above values, it will default to the value given here. In this case, the `None` value for SNR yields a signal without noise (effectively infinite SNR).
- '**bSetInTraining**' A boolean used to specify whether the property should be applied when constructing the dataset (`False`), or when training an ML model (`True`). As noted in the text above, at the time of writing this tech report, we do not have this functionality working, so the value should always be set to `False`, though we intend to add this functionality in the near future. For data intended for inference only, this should be set to `False`.
- '**numFmtStr**' Format string specifying how to format numbers when using for things like plot ranges, as in Fig. 6.
- '**printUnits**' Units to add to plots. Any LaTeX formatting, such as " μHz ," will be passed through to Matplotlib and properly rendered.
- '**description**' Brief text description that will be passed to be used in plot titles, etc.
- '**symbolicLabel**' Symbolic label for quantity, such as " N_s ", for number of samples. If appropriate, LaTeX formatting will be passed through to Matplotlib and properly rendered.

Most parameters in the PDW are handled in the manner given above, with the exception of PRI, f_s , N_s , which are the same for all waveforms in the dataset, and the modulation class, which, along with its labeling, are handled from separate lists within the JSON file.

List of Symbols, Abbreviations, and Acronyms

1-D	One-dimensional
2-D	Two-dimensional
CNN	Convolutional neural network
DoD	Department of Defense
FFT	Fast Fourier transform
FT	Fourier Transform (typically shortening FFT)
GPU	Graphics Processing Unit
IQ+FT	Combination of IQ and FFT data, typically as parallel vectors
IQ	In-phase and quadrature
ML	Machine Learning
NN	Neural Network
P1, P2, P3, P4	The four canonical Polyphase radar modulations
P-code	One of the four canonical Polyphase radar modulations
PDW	Pulse descriptor word
PRI	Pulse repetition interval
PW	Pulse width
PyRadarWF	Python Radar Waveform
ResNet	Residual Neural Network
RF	Radio Frequency
RFML	Radio Frequency Machine Learning
SIDLE	Signal Instantiations for Deep Learning Experiments
SNR	Signal-to-noise ratio

WF-Mods09 Set of 9 waveform modulations used in this work.

WF-Mods18 Set of 18 waveform modulations used in this work.

WF Wave form class

WVD Wigner–Ville distribution

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

1 DEVCOM ARL
(PDF) FCDD RLB CI
TECH LIB