2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC

Host Sponsor UNC Charlotte

Paper ID #50710

Experiential Radio Frequency Engineering Education with Machine Learning Focused Signal Processing Labs

Erwin Karincic, Virginia Commonwealth University

Erwin Karincic received B.S. and M.S. degrees in Computer Engineering from Virginia Commonwealth University (VCU) in 2020 and 2021, respectively. He is currently pursuing a Ph.D. degree from Virginia Commonwealth University. He is an experienced security researcher with focus on reverse engineering and exploit development. An avid learner in many different fields, his research interests are cyber security, reverse engineering, exploit development, Internet of Things, software defined radio, antenna analysis, and design. He currently holds 28 cyber security certifications.

Dr. Lauren Linkous, Virginia Commonwealth University

Dr. Lauren Linkous is a Postdoctoral Research Associate specializing in Machine Learning, electromagnetics, and automation. She received her Ph.D. in Engineering from Virginia Commonwealth University. She has extensive teaching experience across electrical engineering and computer science, including courses in circuits, medical device security, electromagnetics, and machine learning. She has developed curricula on RF optimization, AI/LLM applications, and wireless communication, guiding students through hands-on labs with industry-standard tools. She also holds an M.S. in Computer Science and dual B.S. degrees in Electrical Engineering and Physics from VCU.

Dr. Erdem Topsakal, Virginia Commonwealth University

Experiential Radio Frequency Engineering Education with Machine Learning Focused Signal Processing Labs

Abstract

The rapid advancement of machine learning (ML) has created new opportunities within engineering disciplines; however, integrating practical ML techniques into Radio Frequency (RF) curricula remains inadequately addressed. Traditional RF engineering programs build upon a series of signal processing, circuits, and communications theory courses, often reaching core RF implementation several years into the program with little or no exposure to modern ML topics. While RF education tracks are highly specialized, students are increasingly entering the workforce where they encounter projects requiring ML, or even Artificial Intelligence (AI), integration into existing radio systems, yet most lack hands-on experience bridging these domains. This gap in instruction propagates challenges for future wireless engineers, as ML and AI knowledge can significantly boost their career opportunities. This work presents a series of laboratory-based exercises that bridge this divide through progressive, hands-on RF/ML experimentation building up to neural network automatic modulation classification (AMC). We provide eight practical labs that can be integrated into existing RF courses, offering educators options to incorporate ML techniques into their existing curricula.

The labs presented here are designed to reinforce foundational concepts of both RF and ML before attempting their integration. These exercises address unique challenges posed by applying ML to wireless signals, such as the deviation between synthetic and captured signals that often invalidate lab-only trained models when applying them to over-the-air (OTA) signals.

The lab sequence begins with foundations of RF, including spectrum visualization, where students generate and visualize synthetic signals. Students then create Software Defined Radio (SDR) procedures to transmit and receive signals using GNU Radio, understanding how OTA RF signals propagate. This is followed by spectrum analysis and signal classification where signals are identified via feature extraction as a basis for what will be automated with machine learning. Progressive modules introduce neural network development where students will implement a 1D Convolutional Neural Network (CNN) on synthetic datasets, advancing to captured signals mirroring real RF propagation challenges. The labs progress to use TorchSig, an open-source signal processing ML toolkit, to introduce sophisticated signal generation capabilities, including realistic channel impairments and signal-to-noise ratio (SNR) values. Students implement and compare multiple architectures, from basic CNNs to advanced Cross-Covariance Image Transformer (XCiT) models, evaluating their performance under varying RF conditions. The final lab culminates with the real-time use of trained models for live signal classification, demonstrating practical applications of ML in RF systems.

The presented sequence of labs follow a structured progression that reinforces prerequisite knowledge of core RF education where each lab builds upon concepts from previous exercises. Educators can reorder or skip any labs where students have appropriate prerequisite knowledge. Additionally, each lab includes configurable parameters for adjusting complexity levels based on course requirements. These labs can be adopted in undergraduate or graduate programs with clear

pathways for extending content based on learning outcomes.

Our contributions provide educators with reproducible labs for integrating ML and RF education, which prepares students for emerging challenges in next generation wireless systems where dynamic signal processing will be fundamental to system optimization.

Introduction

Machine learning has been at the forefront of many recent technological innovations, including those within radio frequency engineering. The convergence of machine learning (ML) and radio frequency (RF) engineering represents a shift in wireless communications, fundamentally changing how students need to be prepared for modern industry demands. Modern wireless networks increasingly rely on ML-driven approaches for vital functions/operations including signal classification, interference mitigation, resource allocation, and cognitive ratio operations.

The rapid advancement of machine learning has created new opportunities within engineering disciplines [1, 2, 3]; however, integrating practical ML techniques into RF curricula remains inadequately addressed. This creates a critical gap between traditionally in-demand capabilities required by RF engineering roles and the educational preparation provided by traditional academic programs. Traditional RF curricula are structured around sequential courses in signal processing, circuits, and communications theory, which typically provide minimal exposure to ML or artificial intelligence (AI) methodologies, if at all [4]. As graduates enter the workforce, they increasingly encounter projects requiring ML integration into wireless communication systems, yet most lack the hands-on experience necessary to effectively bridge these two domains [1, 2]. This issue is further compounded by longstanding challenges in RF instruction, where limited access to hands-on experiential opportunities has historically made it difficult for students to connect theoretical knowledge with practical implementation [5, 6, 4, 7]. This situation is further complicated by traditional RF education requiring expensive instrumentation such as spectrum analyzers and signal generators, significant investments that are beyond many educational institutions. Equipment aside, current challenges exist/extend beyond knowledge transfer between fields [6, 8]; applying ML strategies and tools to hardware RF systems introduces unique challenges/complexities that are absent from conventional ML applications [2, 9].

The intersection of ML and RF fields introduces domain-specific considerations that draw on different foundational skills and applications. Successfully implementing ML in RF systems requires proficiency in both domains, requiring engineers to understand not only ML architectures and training methodologies, but also the physical characteristics of electromagnetics and signal processing fundamentals [1, 2, 9, 10]. This includes substantial differences between near perfect synthetic signals and real-world over-the-air (OTA) transmissions, such as channel impairments, noise characteristics, and realistic propagation effects that can invalidate ML models trained in simulation/with synthetic signals only. For instance, real RF environments exhibit features such as frequency offsets, phase noise, and variations due to hardware irregularities, which are all impairments that simulated datasets often fail to capture [10]. This mismatch results in ML models with poor generalization performance results when used to classify actual transmitted

signals.

Addressing this gap requires hands-on, experiential learning to allow students to experience the practical challenges of RF/ML integration [4, 10]. This knowledge deficit poses significant challenges for emerging wireless engineers, particularly as proficiency in ML and ML techniques becomes essential for next-generation communications. To address this disconnect, we present a structured series of eight lab exercises that progressively integrate RF and ML concepts through hands on experimentation. These labs culminate in the implementation of NN-based automatic modulation classification (AMC) with data collected using an SDR.

The laboratory curriculum follows a scaffolding progression that established essential RF competencies before introducing ML integration, which each lab building upon concepts from previous exercises. Initial exercises focus on fundamental RF concepts through spectrum visualization where students capture and visualize data from one of several software defined radio (SDR) options. Subsequent modules guide students in developing SDR workflows using GNU Radio to receive signals, providing realistic experience with real-world OTA signal propagation. Educators transmit signals for students to receive, ensuring strictly compliant operation within regionally authorized frequency bands. Building on this, students perform spectrum analysis and signal classification through manual feature extraction techniques, establishing a conceptual framework for ML automation introduced in later modules. Advanced modules use TorchSig [11], an open-source ML toolkit build on PyTorch, to generate training data with realistic impairments, including channel distortions and configurable signal-to-noise ratio (SNR).

These practical modules are designed for seamless integration into established RF courses, providing educators with adaptable/modifiable pathways to incorporate contemporary ML techniques within their current instructional frameworks. The material presented guides students through the activities to provide a working knowledge background, and then presents interactive activities/challenges for students to try. Labs can be reordered or omitted when appropriate prerequisite knowledge has been established through other coursework. This framework supports implementation across both undergraduate and graduate programs, with extendable content that can be tailored to a broad range of learning outcomes and curricular objectives.

This work presents a comprehensive sequence of eight laboratory exercises specifically designed to bridge the ML and RF gap through hands-on experimentation. The lab activities are included in modules that have a mix of primers and supplementary material to support both students and educators. These labs can be integrated into existing RF curricula without requiring large-scale course restructuring, providing educators with flexible options for introducing relevant ML concepts at appropriate points in their courses.

The key contributions of this work are as follows:

1. We present activities relevant to modern wireless communication and dynamic signal processing that have direct applications to the current workforce
2. We provide a series of eight lab modules complete with examples, student challenges, and instructions for both hardware and software tool usage to prepare students for emerging challenges in next generation wireless systems.
3. We demonstrate key techniques and material where required domain knowledge acts as a

barrier to entry for both students and educators

4. We provide detailed creation methodology for example datasets, using TorchSig 2.0, comparable to real-world samples with realistic impairments, allowing students to train ML models on synthetic data and accurately predict real world modulation schemes

This paper is organized as follows: Section 1 describes the hardware and software utilized in the lab series, complete with cost estimates. Section 2 presents the eight lab modules with detailed descriptions and learning outcomes. Section 3 discusses the solution materials available for educators upon request. Finally, section 4 concludes this paper.

1 Lab Structure and Components

This section describes the laboratory environment, hardware requirements, software dependencies, and customization options available to educators. The platform combines commercial hardware with open-source software to ensure accessibility across institutions with varying resources.

1.1 Lab Module Setup/Environment

The lab series is organized into eight module directories including all written materials needed for the lab. Each module has a core lab, and may contain three types of content. Primer material presents background information that supports understanding but does not directly affect the core lab exercises. If an instructor chooses to present this material differently or skip it entirely, subsequent labs remain fully functional. The core labs constitute the primary instructional content and form the sequential backbone of the series. Supplementary material includes additional skill-building activities, extended examples, or previews of concepts addressed in future modules. This supplementary content enriches the learning experience but is not prerequisite for completing subsequent core labs.

All core lab material is written in Python using Jupyter notebooks, which provide an interactive environment where students can execute code cells individually while viewing explanatory text and visualizations. This format offers several pedagogical advantages: students can re-run cells to observe how parameter changes affect outputs, virtual environment installations can be executed directly from notebook cells reducing command-line complexity, and the combination of code, output, and documentation in a single file creates a self-contained learning resource. We also provide a selection of GNU Radio flowgraphs to assist with signal generation and spectrum analysis.

1.2 Hardware and Operating System Requirements

The lab series has been tested on both Linux-based and Windows operating systems (OS). Lab modules without hardware will work on either OS, though the installation process will vary. However, due to differences in drivers and hardware, not all hardware is guaranteed to work on Windows systems. For Windows users requiring SDR functionality, Windows Subsystem for Linux (WSL) provides a viable pathway, though configuration complexity varies by hardware. macOS compatibility depends on specific SDR driver availability.

Table 1 lists the hardware platforms supported by the lab series. This includes three SDR options, and a custom, open-source ESP32/CC1101 assembly [12]. Educators need not acquire all of the hardware platforms; the labs can be adapted based on available equipment. At minimum, the instructor station requires transmission capability for generating signals that students will capture and analyze. Student stations only require the ability to receive signals.

Table 1: Supported SDR Hardware

Device	**Capability**	**Frequency Range**	**Cost**
RTL-SDR v3	RX only	24-1766 MHz	$25-40
HackRF One	Half-duplex TX/RX	1 MHz-6 GHz	$300-350
Ettus USRP B200/B205	Full-duplex TX/RX	70 MHz-6 GHz	$1,500-2,500
CC1101 + ESP32	TX/RX (ISM bands)	300-928 MHz	$30

The RTL-SDR is favored for introductory exercises due to its low cost and broad community support, while the HackRF One and Ettus USRP provide the transmission capabilities essential for advanced experimentation [13, 14]. The USRP series offers research-grade fidelity and full-duplex operation suitable for complex systems [15], however the high cost can be a barrier to widespread student ownership. Conducting these exercises requires a minimum of two radios: one with transmission capabilities and one for reception. To address the need for accessible transmission hardware, we introduce a custom microcontroller-based solution designed to generate and transmit configurable RF signals.

The CC1101 radio transceiver paired with an ESP32 microcontroller provides an affordable transmission source for generating known modulation signals [16]. CC1101 operates in ISM bands at 433.92 MHz and supports multiple modulation schemes including 2-FSK, 4-FSK, GFSK, MSK, and ASK/OOK. The CC1101 setup serves as the alternate signal source for dataset creation and real-time classification exercises in later modules. This setup was developed for educational usage needing cost effective and accessible TX/RX capabilities.

1.3 Software Requirements

As this lab series is centered on the exploration of signal processing and machine learning, an OS agnostic model was chosen to create the software environment. This ensures that the setup and operation of the labs is as consistent across devices as possible. The software environment utilizes on Visual Studio Code with the Jupyter extension, providing an integrated development experience for notebook execution. Python 3.8 or later is required, with dependencies managed through virtual environments to ensure reproducibility. Core libraries include NumPy, SciPy, and Matplotlib for signal processing and visualization.

Machine learning components require PyTorch for the primary neural network implementation, with Scikit-learn used for classification accuracy reporting and statistical analysis. The labs include provisions for GPU acceleration through CUDA when available, with automatic fallback to CPU execution. For institutions without local GPU resources, computationally intensive training tasks can be offloaded to Google Colab, which provides free GPU access with an account. The notebooks include Colab-specific setup cells that handle environment configuration

automatically. Instructions for setup and when to use these cells are included in the lab material.

TorchSig 2.0, an open-source signal processing library built on PyTorch, is required for advanced modules. TorchSig installation requires the Rust compiler due to performance-critical components. On Windows systems, Visual Studio Build Tools must be installed prior to installing Rust. Detailed installation instructions are provided in the setup module.

GNU Radio Companion [17] provides the graphical interface for SDR signal flow development. Pre-built flowgraphs are included for signal generation and analysis tasks, reducing the learning curve for students unfamiliar with GNU Radio.

All lab course material will be released open source on GitHub at [18].

1.4 Customization Options

The modular structure of these labs allows educators to adapt the lab sequence to existing course content. Labs that address concepts already covered in prerequisite courses can be abbreviated or omitted entirely. Each lab includes configurable parameters that adjust exercise complexity: signal-to-noise ratios can be modified to increase classification difficulty, dataset sizes can be scaled based on available computation time, and the number of modulation classes can be adjusted to match learning objectives.

The primer and supplementary sections within each module provide natural extension points. Educators teaching advanced courses may assign supplementary challenges as required work, while introductory courses may treat them as optional enrichment. This flexibility ensures the material remains appropriate across undergraduate and graduate programs with varying prerequisite structures.

2 Lab Modules

This section presents the eight laboratory modules that comprise the core instructional sequence. The modules progress from foundational RF concepts through increasingly sophisticated ML applications, culminating in real-time signal classification. Table 2 summarizes each module's focus area and hardware requirements.

2.1 Concept and Implementation

The lab sequence follows a scaffolded progression where each module builds upon concepts established in previous exercises [4]. Early modules establish foundational understanding of RF signals and SDR operation before introducing machine learning concepts [19, 20]. This structure ensures students develop intuition for signal characteristics through manual analysis before automating classification with neural networks.

The adaptable configuration allows educators to adjust complexity based on course objectives and available hardware. Labs requiring SDR equipment include alternative pathways using pre-captured datasets for institutions with limited hardware access. Students engaged in hands-on hardware-based instruction develop stronger conceptual understanding and intuition for RF

Table 2: Lab Module Overview

Module	Title	Focus Area	Hardware
0	Setup	Environment configuration	None
1	RF Foundations	Signal fundamentals, FFT	None
2	SDR Toolchain	SDR hardware, GNU Radio	SDR
3	Modulation	Signal generation, schemes	Optional SDR/CC1101
4	Spectrum Analysis	Feature extraction, visualization	Optional SDR/CC1101
5	Narrowband Classifier	ML introduction, CNN/RNN	None
6	Dataset Creation	OTA signal capture	SDR/CC1101
7	Advanced Classification	TorchSig, transfer learning	Optional GPU
8	Real-time Classification	Live inference	SDR/CC1101

concepts compared to purely theory-driven approaches [5]. The following modules are designed to bridge the theory-practice gap through exposure to real-world propagation effects while providing individualized hands-on access without expensive proprietary equipment.

2.2 Module Hardware Requirements

Table 2 indicates the hardware requirements for each module. Modules 0, 1, and 5 require only a computer with Python and the specified libraries for the lab. Modules 2, 6, and 8 require SDR hardware for signal capture; pre-captured datasets are provided for institutions without hardware access [18]. Modules 3 and 4 benefit from SDR hardware but can be completed using synthetic signals or provided I/Q files. Module 7 benefits from GPU acceleration but includes Google Colab instructions for institutions without local GPU resources for students. The notebooks in each module contain specific library installation instructions.

2.3 Module 0: Setup

2.3.1 Session Configuration

This module sets up the necessary software for use in the rest of the lab series. To complete the activity, students need a computer with Visual Studio Code installed and internet access for downloading library dependencies. Additionally, students with Google accounts can use Google Colab for cloud-based execution, which is particularly useful for computationally intensive modules such as Module 7.

2.3.2 Description

The setup module establishes the software environment used throughout the lab series. The primary development environment combines Visual Studio Code with the Jupyter extension, providing an integrated experience where students can execute code cells individually while viewing explanatory text and visualizations. Students install the official Python and Jupyter extensions from the VS Code marketplace, create a Python virtual environment to isolate project dependencies, and verify that core libraries install correctly. The module walks through the process of selecting the correct kernel in VS Code, emphasizing that students should confirm the

virtual environment (displayed as `.venv` in the kernel selector) is active before installing packages or running notebooks. This becomes critical as sequential notebooks build on dependencies and do not attempt to reinstall existing libraries for every module. The complete list of dependencies for the entire lab series is provided in the documentation.

The setup process includes verification steps where students generate test plots and save data to confirm that the NumPy, Matplotlib, and Pandas libraries have been imported correctly. Students verify the Python version installed on their OS, import all required libraries for the module, and create a simple sine wave plot to confirm that matplotlib renders correctly within VS Code. File I/O operations are tested through exercises that save figures in multiple formats (PNG, PDF, SVG), write data to CSV files using Pandas DataFrames, and read data back to demonstrate round-trip integrity. These exercises ensure the environment can handle the data persistence operations used in subsequent labs and provide students a reference for saving their work.

Google Colab is presented as a cloud-based alternative to local code execution for ML model training and dataset generation. When GPUs are not available to students, Google Colab can be utilized to reduce the time and resources needed to synthesize data and train ML models. Colab notebooks execute on Google's servers with free GPU access, eliminating the need for local CUDA installation. The Jupyter notebooks in the lab modules can either be uploaded directly to Google Drive and run there with a hosted runtime utilizing GPUs, or a hosted runtime can be accessed through VS Code via an official extension. When using VS Code with the Colab extension, students must enable experimental settings for server mounting and file uploading, as standard Colab commands like `drive.mount()` require browser pop-ups that VS Code cannot display. After enabling these settings and connecting to a hosted Colab runtime, students can mount the remote server to their local workspace and upload files directly through VS Code's file explorer. This workflow is documented in the Module 7 where GPU acceleration significantly reduces training time. There are no differences in the activity or information learned when running in the Google Colab or via Visual Studio Code with Colab extension. However, local hardware cannot be run on hosted GPUs, so this option does not apply to the hardware-based labs.

2.3.3 Learning Outcomes

After completing this module, students will have a functional development environment configured for the remaining labs in the sequence. Students will be able to create and activate Python virtual environments, install libraries using `pip`, and execute Jupyter notebook cells within VS Code.

2.4 Module 1: RF Foundations

2.4.1 Description

This module introduces the fundamental concepts of Radio Frequency signals and the electromagnetic spectrum. A Radio Frequency signal is an electromagnetic wave that carries information wirelessly through the air. RF signals oscillate at frequencies ranging from 3 kHz to 300 GHz and form the foundation of wireless communication systems including radio, television, cellular networks, WiFi, and radar. These signals can be mathematically represented and analyzed

to understand their behavior, which is essential for designing and troubleshooting wireless systems[19, 20].

The basic sinusoidal signal is expressed as:

$$s(t) = A \cdot \sin(2\pi f t + \phi) \tag{1}$$

where A represents amplitude (signal strength), f represents frequency (cycles per second, measured in Hz), t represents time, and ϕ represents phase (offset in radians). Students are able guided through generating their first signals with this equation to see how individual parameters impact the shape of the signal.

Signals can be analyzed in two complementary ways. Time domain analysis shows how signal amplitude changes over time, representing what the signal looks like as it propagates. This is the natural way to observe signals on an oscilloscope. Frequency domain analysis shows which frequency components make up the signal and their respective strengths. This view is obtained through Fourier analysis using the Fast Fourier Transform (FFT) and is essential for understanding bandwidth, interference, and spectrum allocation. Both perspectives are necessary for complete signal analysis: the time domain reveals temporal behavior while the frequency domain reveals spectral content [19, 20].

Students begin by generating sinusoidal waves with varying parameters and observe how changes in frequency, amplitude, and phase affect the resulting waveforms. The module progresses to FFT analysis where students create composite signals containing multiple frequency components and convert them to frequency-domain representations, observing how the magnitude spectrum reveals the strength of each component. This exercise demonstrates the power of spectral analysis for decomposing complex signals into their constituent frequencies. Figure 1 shows a composite signal containing 10 Hz, 25 Hz, and 50 Hz components with amplitudes of 1.0, 0.5, and 0.3 respectively; the FFT spectrum clearly identifies each frequency and its relative magnitude.

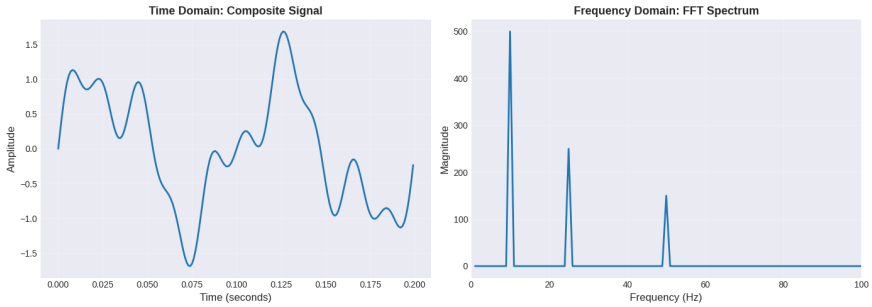

Figure 1: Side-by-side comparison of time domain and frequency domain representations. Left: A composite signal appears complex in the time domain. Right: The FFT spectrum reveals three distinct frequency components at 10 Hz, 25 Hz, and 50 Hz with their respective magnitudes.

Bandwidth concepts are introduced through comparison of narrowband and wideband signals. A narrowband signal contains a single frequency component and produces a simple waveform, while a wideband signal contains multiple frequency components and produces a complex waveform. Both signal types are visualized in time and frequency domains, observing how

bandwidth directly relates to the range of frequencies occupied. The module emphasizes that bandwidth is a limited and regulated resource in wireless communications, with wider bandwidth generally enabling higher data rates but requiring more spectrum allocation[19, 20].

The module introduces modulation as the process of varying properties of a carrier wave according to a message signal. Without modulation, practical wireless communication would be impossible: antennas would need to be impractically large, and multiple signals could not share the same medium without interference. We then further explore Amplitude Modulation (AM), where the carrier amplitude varies according to the message signal, and Frequency Modulation (FM), where the carrier frequency varies according to the message signal. Figure 2 demonstrates both modulation techniques, showing how the message signal information is encoded onto the carrier wave through amplitude variations (AM) or frequency variations (FM).

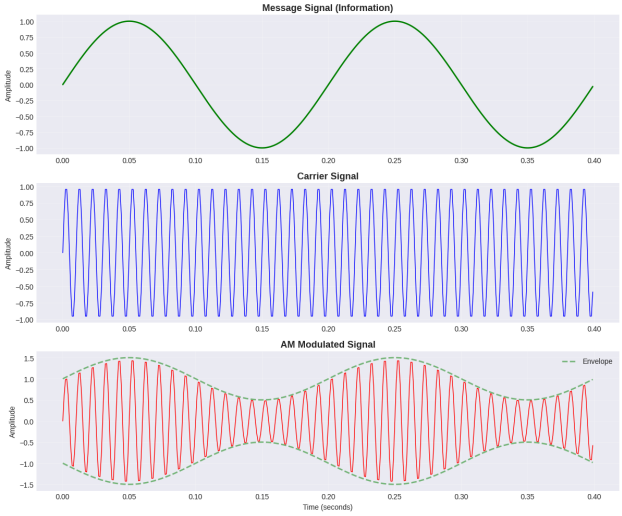

Figure 2: Amplitude Modulation demonstration showing the message signal (top), carrier signal (middle), and AM modulated signal (bottom). The dashed envelope traces how the carrier amplitude varies according to the message signal with a modulation index of 0.5.

The module culminates with a practical application exercise where students generate and analyze a complex modulated signal similar to those used in modern communication systems. Students create a complex signal using In-phase (I) and Quadrature (Q) components, which represent two orthogonal carriers that allow simultaneous transmission of two data streams. This I/Q representation preserves both amplitude and phase information as a stream of complex numbers, and serves as the standard format for SDR captures throughout the remaining modules. Figure 3 shows the time-domain waveform and frequency spectrum of a digitally modulated signal, demonstrating concepts that bridge to subsequent labs on SDR signal capture and classification.

2.4.2 Learning Outcomes

After completing this module, students will understand fundamental RF signal properties including frequency, amplitude, and phase. Students will gain practical skills in generating RF signals programmatically and visualizing signals in both time and frequency domains. The

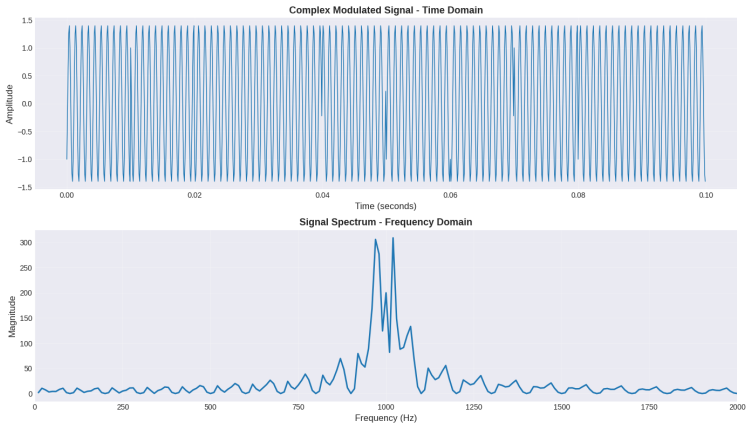

Figure 3: Complex I/Q modulated signal demonstrating digital modulation concepts. Top: Time domain showing the modulated waveform with symbol transitions. Bottom: Frequency spectrum showing the signal's bandwidth centered around the carrier frequency.

module establishes foundational understanding of bandwidth concepts, analog modulation techniques (AM and FM), and I/Q signal representation that supports all subsequent labs in the series.

2.5 Module 2: SDR Toolchain

2.5.1 Description

A Software Defined Radio moves signal processing functions like filtering, modulation, and demodulation from dedicated analog circuits into software [13]. With this design, a general-purpose computer to perform signal processing on digitized samples from a relatively simple RF front-end. This architecture allows a single hardware platform to function as an AM/FM radio, WiFi transceiver, or perform other wireless operations simply by changing the software.

The SDR signal processing pipeline flows from the antenna through RF hardware to the computer. The antenna converts electromagnetic waves to electrical signals, which the SDR hardware digitizes using an Analog-to-Digital Converter (ADC). The USB interface transfers digital samples to the computer, where software processes, demodulates, and analyzes the data. The digitized output uses the I/Q representation introduced in Module 1, enabling software reconstruction of the original signal characteristics.

The module introduces three SDR platforms at different price and capability levels. The RTL-SDR provides receive-only capability at minimal cost ($40), suitable for signal capture and analysis exercises. The HackRF One ($350) adds transmission capability with half-duplex operation, enabling students to both generate and receive signals. The Ettus USRP B200/B210 series ($1,500-2,500) provides full-duplex operation with higher sample rates (up to 56 MHz bandwidth) and 12-bit ADC resolution, supporting the most demanding real-time applications. This range of hardware allows for flexibility across hobbyist and industry

The module begins with hardware verification including examples for all 3 SDRs. This module

then explores the RF spectrum using GQRX and GNU Radio Companion. GQRX is a free graphical SDR receiver application that provides immediate visualization through an FFT display showing real-time signal power versus frequency, and a waterfall display showing signal history over time. Students then scan the FM broadcast band (88-108 MHz) to identify active stations. Figure 4 shows the GQRX interface displaying an FM broadcast signal with both spectrum and waterfall views. GQRX also supports built-in demodulation for AM, FM, and SSB modes, allowing to listen to discovered signals.

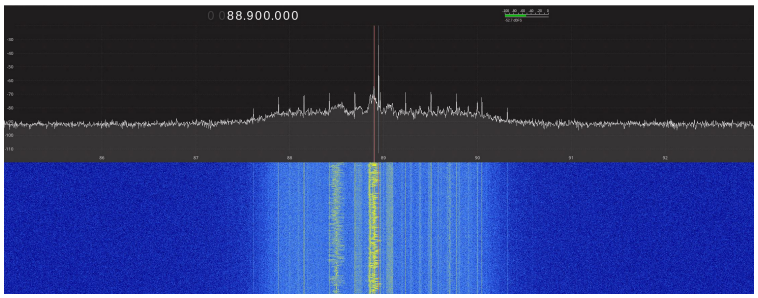

Figure 4: GQRX software defined radio receiver interface showing FM broadcast reception. The top panel displays the real-time FFT spectrum with signal peaks corresponding to active stations. The bottom panel shows the waterfall display revealing signal activity over time.

GNU Radio Companion provides the visual programming environment for constructing signal processing flowgraphs. Students build an FM receiver by following instructions and connecting specific flowgraph blocks. Figure 5 shows the complete FM receiver flowgraph. A `QT GUI Range` block creates a frequency slider allowing tuning across the FM band in real-time.

The module also introduces Python-based signal capture as an alternative to GNU Radio. Using device-specific libraries, students capture I/Q samples programmatically with configurable center frequency, sample rate, and duration. Captured data is visualized using the signal processing techniques from Module 1. The three tools presented in this module for capturing and interacting with signals ensure that students are exposed to a range of practical and professional resources that they can utilize beyond thie lab series.

2.5.2 Learning Outcomes

After completing this module, students will understand and be able to work with Software Defined Radios, and will learn how SDRs differ from traditional hardware radio implementations. They gain practical experience verifying SDR hardware installation, exploring the RF spectrum with GQRX, and constructing GNU Radio flowgraphs for FM reception. Additionally, students will be able to capture I/Q data using both GNU Radio and Python, visualize captured signals in multiple domains, and save data in formats suitable for later analysis. The module establishes the hardware foundation for signal capture and classification in subsequent exercises.

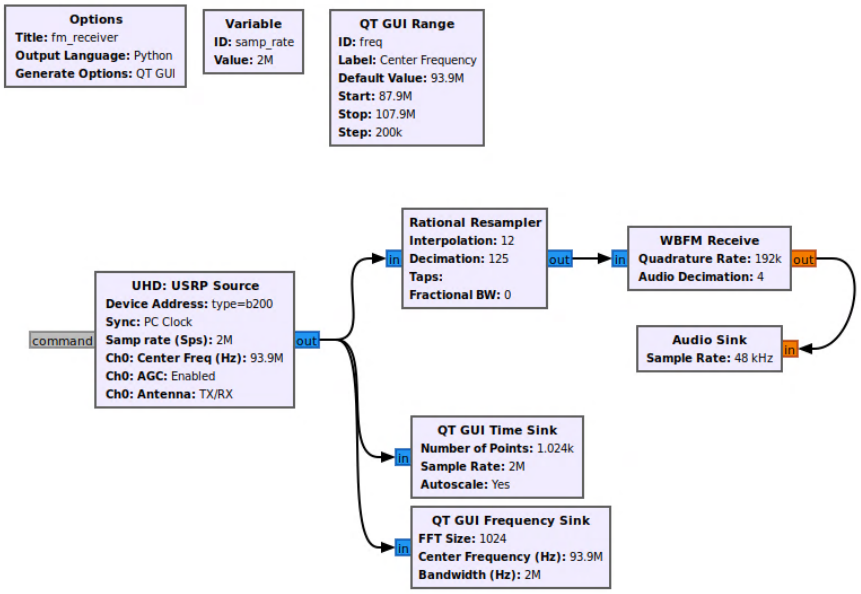

Figure 5: GNU Radio Companion flowgraph for FM broadcast reception. The signal flows from the SDR source block through a rational resampler that converts the sample rate for the WBFM demodulator, which outputs audio to the speakers. QT GUI blocks provide real-time spectrum and time-domain visualization.

2.6 Module 3: Modulation

2.6.1 Description

Modulation is the process of encoding information onto a carrier signal by varying one or more of its properties: amplitude, frequency, or phase. This module provides comprehensive coverage of both analog and digital modulation schemes commonly encountered in wireless systems.

Analog modulation techniques include Amplitude Modulation (AM), where the carrier amplitude varies proportionally to the message signal; Frequency Modulation (FM), where the instantaneous frequency deviates from the carrier based on the message; and Phase Modulation (PM), where the carrier phase shifts according to the message signal [19].

Digital modulation schemes encode discrete symbols rather than continuous waveforms. Phase Shift Keying (PSK) variants include Binary PSK (BPSK) using two phase states, Quadrature PSK (QPSK) using four states to encode two bits per symbol, and 8-PSK using eight states for three bits per symbol. Quadrature Amplitude Modulation (QAM) combines amplitude and phase variation, with 16-QAM encoding four bits per symbol through 16 distinct constellation points.

The lab implements signal generation functions for a number of modulation types that represent majority of the families of modulation signals, producing time-domain waveforms with configurable parameters. Each signal can be visualized in multiple representations: time-domain plots showing amplitude variation, frequency-domain spectra revealing bandwidth characteristics,

and constellation diagrams displaying the relationship between in-phase and quadrature components. Figure 6 presents twelve modulation types in the time domain, allowing students to observe how each scheme exhibits distinct visual characteristics that inform later classification approaches.

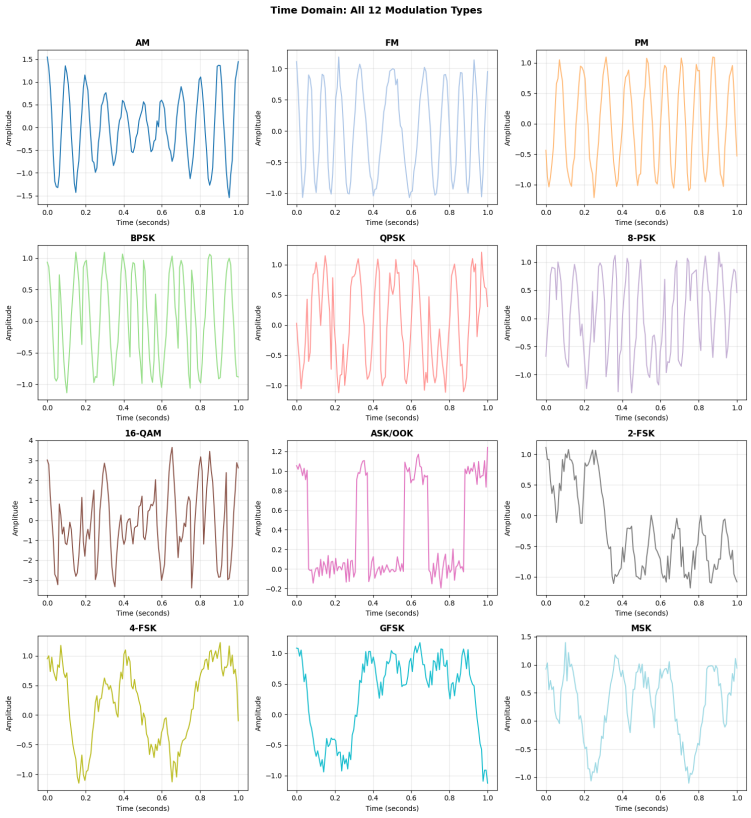

Figure 6: Time-domain visualization of all twelve modulation types covered in the module. Analog modulations (AM, FM, PM) show continuous variations, while digital schemes exhibit discrete symbol transitions. FSK family modulations maintain constant envelope while varying frequency.

Constellation diagrams provide a particularly powerful visualization for digital modulations. These plots display the In-phase (I) component on the x-axis and the Quadrature (Q) component on the y-axis, with each point representing a symbol state. BPSK produces two clusters along the I-axis corresponding to its two phase states. QPSK shows four clusters arranged in a square pattern, each representing a unique phase. 8-PSK displays eight clusters arranged in a circle, maintaining constant amplitude while varying only phase. 16-QAM produces a 4×4 grid pattern where both amplitude and phase vary to encode four bits per symbol. Figure 7 illustrates these constellation patterns, demonstrating how the number and arrangement of symbol points directly relates to the bits per symbol each scheme can transmit.

The module includes exercises where students add Additive White Gaussian Noise (AWGN) to signals at varying signal-to-noise ratios. Observing how noise affects constellation diagrams and time-domain waveforms builds intuition for the challenges that machine learning classifiers must address when identifying modulation schemes in realistic conditions. At low SNR, constellation

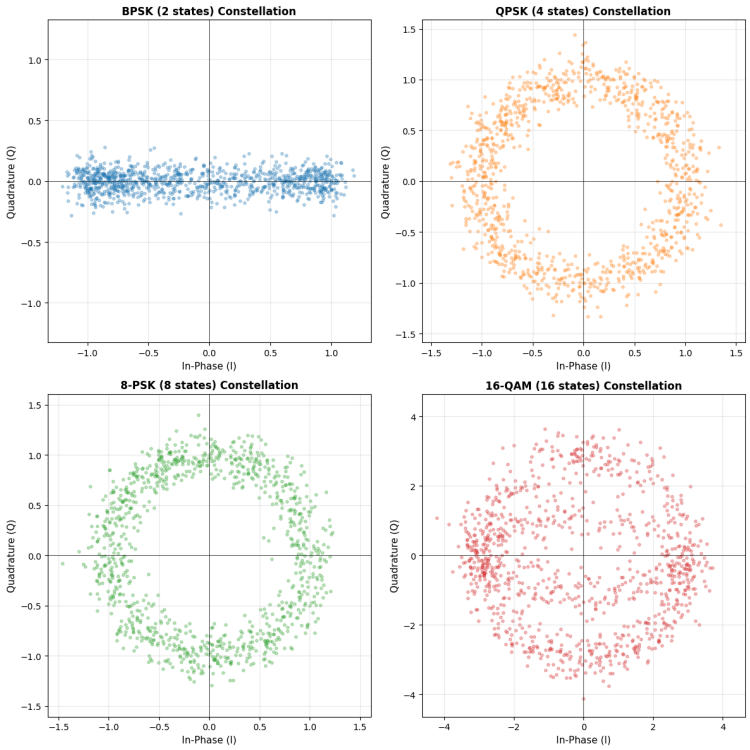

Figure 7: Constellation diagrams for BPSK (2 points), QPSK (4 points), 8-PSK (8 points arranged in a circle), and 16-QAM (16 points in a grid), demonstrating how digital modulation schemes map bits to symbol positions in the I/Q plane. The scatter around ideal symbol locations is caused by noise.

clusters overlap significantly, making symbol detection difficult. At high SNR, clusters become tightly grouped around their ideal positions.

For hands-on RF experimentation, this module includes pre-built GNU Radio Companion flowgraphs that enable students to generate and transmit modulated signals using SDR hardware. The provided flowgraphs cover AM modulation, FM modulation, ASK/OOK modulation, PSK modulation supporting BPSK, QPSK, and 8-PSK, 16-QAM modulation, and Frequency Shift Keying (FSK) family modulation including 2-FSK, 4-FSK, GFSK, and MSK. Each flowgraph includes hardware support for both USRP and HackRF platforms. Students can adjust modulation parameters using real-time QT GUI controls and observe how parameter changes affect the transmitted signal.

Modules 4, 6, and 8 require receiving a transmitted signal which requires minimum of two SDRs, so as an alternative to SDR-based transmission, the module provides ESP32 firmware for the CC1101 radio transceiver, which operates in the 433.92 MHz ISM band. This configuration costs approximately $30 and supports five modulation presets: 2-FSK, ASK/OOK, GFSK, 4-FSK, and MSK. The firmware we provide includes an interactive serial console where students can switch between modulation presets at runtime and transmit packets with configurable parameters. This setup is effective for courses where the RTL-SDR is the primary SDR, as it is only capable of receiving signals.

2.6.2 Learning Outcomes

After completing this module, students will generate all twelve modulation types programmatically and visualize them in time domain, frequency domain, and constellation representations. Completing the module exercises develops understanding of analog versus digital modulation distinctions, tradeoffs between spectral efficiency and noise immunity, and how modulation parameters affect signal characteristics.

2.7 Module 4: Spectrum Analysis

2.7.1 Description

Spectrum analysis bridges manual signal identification and automated machine learning classification [10]. This module develops feature extraction techniques [21] that students will later automate with neural networks, establishing understanding of what characteristics distinguish different modulation schemes.

Building on the FFT analysis from Module 1, this module extends frequency-domain techniques to Power Spectral Density (PSD) estimation. While direct FFT produces high-resolution but noisy spectral estimates, Welch's method reduces variance by dividing the signal into overlapping segments, windowing each segment, computing individual periodograms, and averaging the results. Side-by-side comparison reveals how Welch's averaging produces smoother spectral estimates that reveal signal characteristics buried in noise. Figure 8 demonstrates this variance reduction.

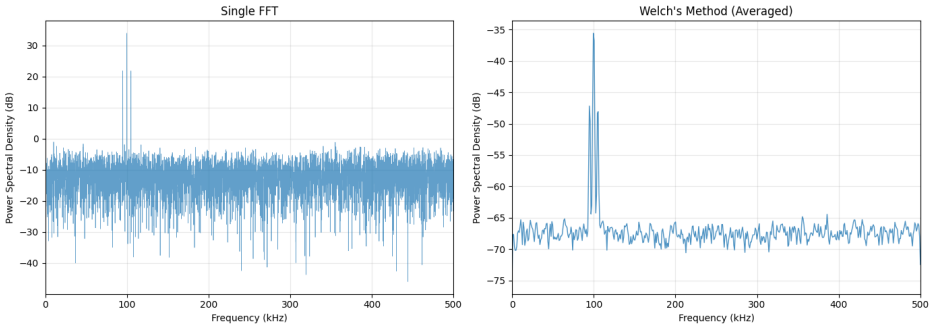

Figure 8: Comparison of single FFT (left) versus Welch's method (right) for an AM signal with noise. The single FFT shows high variance in the noise floor, while Welch's averaging produces a smoother spectrum that more clearly reveals the carrier and sidebands.

FFT size directly affects frequency resolution according to $\Delta f = f_s/N$, where f_s is the sample rate and N is the FFT length. Experimenting with different FFT sizes illustrates this tradeoff: smaller FFTs provide faster computation and better time resolution for spectrograms but coarser frequency detail, while larger FFTs resolve fine spectral features at the cost of requiring more samples. Window functions including Hamming, Hanning, and Blackman are applied to mitigate spectral leakage caused by truncating non-periodic signals. Students are able to adjust parameters and explore the impact of the FFT and the window functions on signals.

Spectrograms extend frequency analysis into the time dimension using the Short-Time Fourier Transform (STFT), which computes FFTs over sliding windows of the signal. In the examples for this module, the resulting time-frequency representation displays how spectral content evolves, with time on one axis, frequency on another, and power intensity shown as color. Exploring the time-frequency tradeoff inherent in STFT analysis reveals that larger FFT windows improve frequency resolution but blur rapid temporal changes, while smaller windows capture transients more precisely at the cost of frequency detail. This visualization proves essential for identifying frequency-hopping signals, burst transmissions, and interference patterns that would be invisible in time-averaged spectra.

The module introduces quantitative feature extraction methods that transform visual observations into measurable quantities. The -3 dB bandwidth measures the frequency span between half-power points relative to the peak, while occupied bandwidth (OBW) calculates the frequency range containing 99% of signal power. Signal-to-Noise Ratio (SNR) estimation requires identifying signal and noise regions in the spectrum and computing their power ratio. Spectral flatness, calculated as the ratio of geometric mean to arithmetic mean of the power spectrum, distinguishes noise-like (flat) spectra from tonal (peaked) spectra. These quantitative features provide the foundation for both manual classification and the machine learning approaches introduced in subsequent modules.

Modulation-specific feature extraction moves beyond general spectral characteristics to identify signatures unique to each modulation type. Envelope analysis extracts the signal magnitude over time, revealing amplitude variations that distinguish ASK/OOK and AM from constant-envelope modulations. Instantaneous frequency, computed as the derivative of the unwrapped phase, shows frequency deviation patterns that differentiate FM from FSK variants. Phase trajectory analysis reveals discrete phase clusters in PSK signals and distinguishes GFSK (S-curved transitions) from MSK (linear phase ramps) based on phase acceleration characteristics. Constellation diagram analysis maps the I/Q relationship to identify symbol states and their arrangement, distinguishing PSK (ring patterns) from QAM (grid patterns).

The module provides a GNU Radio Companion flowgraph (`signal_analyzer.grc`) that implements eight simultaneous visualization windows for comprehensive signal characterization. The flowgraph as shown in Figure 9 displays spectrum (FFT), waterfall (spectrogram), constellation, envelope time series, instantaneous frequency time series, phase trajectory, amplitude distribution histogram, and frequency distribution histogram. These views develop intuition for how different modulation types appear across multiple representations. The amplitude histogram reveals discrete levels in ASK while showing single peaks for constant-envelope modulations. The frequency histogram displays two peaks for 2-FSK, 2GFSK, and four peaks for 4-FSK.

A rule-based classifier in Python combines these features into a decision tree for modulation identification. This type of classifier is a simple entry point for students to understand how manually filtering signals works, and the priory of characteristics used to identify a signal. While a rules-based classifier is not broadly applicable across problems, it is a computationally low cost method that mirrors the intuition students will work to develop in these modules. The classifier first examines envelope variance to identify amplitude-based modulations (ASK/OOK, AM, QAM). For constant-envelope signals, it analyzes the frequency histogram to count discrete

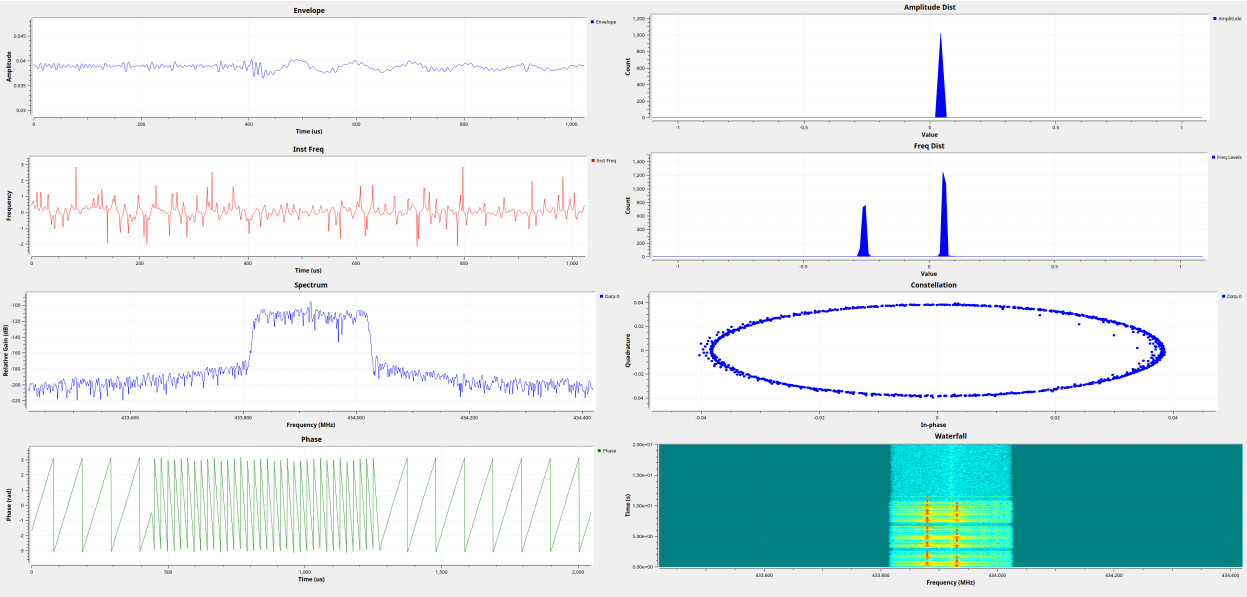

Figure 9: GNU Radio flowgraph output used for signal analysis displays envelope time series, amplitude distribution histogram, instantaneous frequency time series, frequency distribution histogram, spectrum, constellation, phase trajectory, and waterfall.

frequency levels, distinguishing 2-FSK, 4-FSK, and analog FM. Phase trajectory characteristics separate GFSK from MSK based on the linearity of phase ramps. Constellation clustering identifies PSK order (BPSK, QPSK, 8-PSK) based on the number and arrangement of symbol clusters. Figure 10 shows the six-panel diagnostic view generated during classification, illustrating how each feature contributes to the modulation identification decision.

The module concludes by examining the limitations of rule-based classification. For example, fixed thresholds tuned for specific SNR and symbol rates may not generalize to different signal conditions. Also, binary decision points cannot express uncertainty, and edge cases between similar modulations (GFSK versus MSK, QPSK versus 8-PSK) challenge handcrafted rules. The classifier does not improve with additional examples, and performance degrades rapidly at low SNR where noise corrupts the extracted features. These limitations motivate the machine learning approaches introduced in Module 5, where algorithms learn optimal decision boundaries from training data rather than relying on manually specified thresholds.

2.7.2 Learning Outcomes

After completing this module, participants will compute PSD using both direct FFT and Welch's method, understanding the tradeoff between frequency resolution and variance reduction. The exercises cover spectrogram generation and interpretation, window function application to reduce spectral leakage, and quantitative feature extraction including bandwidth, SNR, and spectral flatness. Modulation-specific feature extraction addresses envelope, instantaneous frequency, phase trajectory, and constellation analysis. The decision tree methodology classifies various modulation types from Module 3, illustrating both the capabilities and limitations of rule-based approaches that motivate the machine learning methods in subsequent modules.

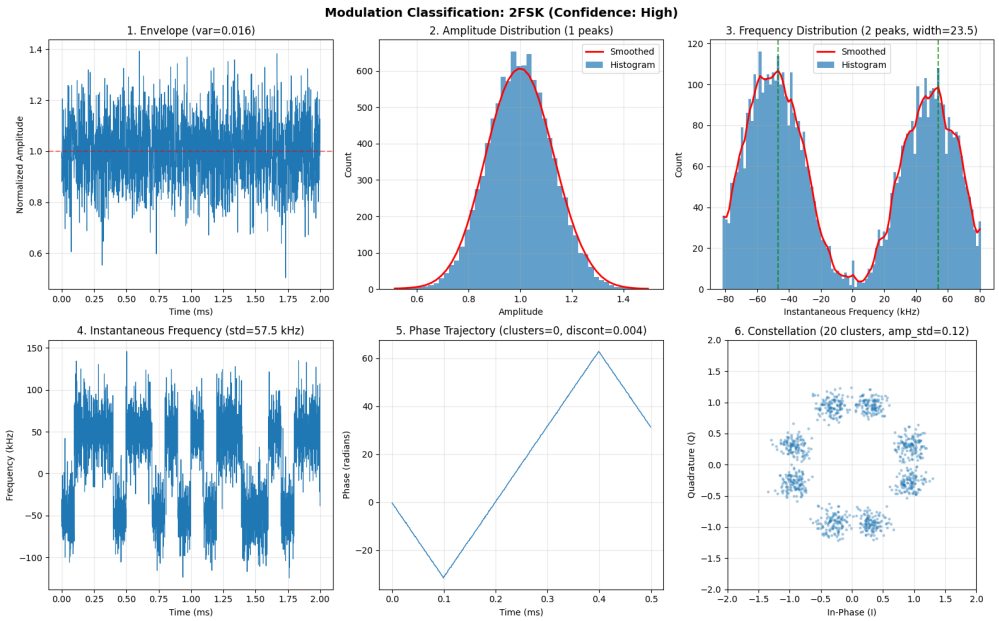

Figure 10: Six-panel diagnostic view from the rule-based classifier showing envelope time series, amplitude distribution, frequency distribution, instantaneous frequency, phase trajectory, and constellation diagram. Each panel provides complementary information for modulation identification.

2.8 Module 5: Narrowband Classifier

2.8.1 Description

The limitations of rule-based classification identified in Module 4 motivate machine learning approaches that learn optimal decision boundaries from data featured in this module. This module introduces ML for RF signal classification, progressing from classical methods through neural network implementations. A narrowband classifier focuses on signals occupying a relatively small frequency band, making it suitable for identifying modulation schemes within a single channel. Applications include spectrum monitoring, signal detection in crowded RF environments, and automatic modulation recognition.

The primer introduces four machine learning approaches applied to synthetic signal data Support Vector Machines (SVM), Neural Networks (NN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) [22, 23].

SVMs find the optimal boundary (hyperplane) separating different classes in feature space. SVMs work well with moderate-sized datasets, are computationally efficient, and use kernel functions to handle non-linear problems. Key parameters include the kernel type (linear, RBF, or polynomial), the regularization parameter C that controls the tradeoff between decision boundary smoothness and classification accuracy, and the gamma parameter that defines how far the influence of a single training example reaches.

NNs consist of layers of interconnected nodes that learn patterns through forward propagation and backpropagation. The hidden layer configuration determines network capacity, with deeper networks capable of learning more complex patterns at the cost of increased training time and

overfitting risk. The activation function introduces non-linearity, with ReLU being the most common choice for hidden layers. Training parameters include the solver algorithm (Adam is generally effective without extensive tuning), learning rate, and maximum iterations.

CNNs use convolutional layers to automatically learn spatial or temporal features. Originally developed for images, CNNs work excellently for time-series signal data because they recognize patterns regardless of position in the signal. The architecture consists of convolutional layers with configurable kernel sizes that determine how many adjacent samples are analyzed together, pooling layers that downsample the signal to focus on important features, and fully-connected layers that map learned features to class predictions. Dropout regularization randomly disables neurons during training to prevent overfitting.

RNNs process sequences step-by-step while maintaining internal state (memory). Long Short-Term Memory (LSTM) variants handle long-range temporal dependencies effectively through separate hidden and cell states that allow the network to remember information from earlier in the sequence. Unlike CNNs that identify local patterns, LSTMs capture dependencies across the entire signal sequence, making them effective when temporal evolution matters for classification.

During the model training and evaluation phase, splitting the dataset into training and test sets enables evaluation of how well a model generalizes to new, unseen data. The training set teaches the model patterns, often using small subsets (batches) to iteratively improve the model over many iterations within an epoch. This training data is only ever used to train the model. The test data is used to evaluate the performance of the model at certain check points, typically at the end of an epoch. The standard 80/20 train/test split provides sufficient training data while reserving enough samples for reliable testing. In some examples, the data is split into three categories to allow for a subset to be allocated as validation data. Validation data is never used in the training or test period. After the model has been trained, the previously unseen validation is data is used to evaluate how well a model performs on new data.

The primer demonstrates all four ML architectures and compares their performance on the same synthetic dataset. All methods can achieve perfect accuracy on clean synthetic data, which occurs because the signals were generated algorithmically with clear, distinct characteristics, low noise levels, and strong discriminative features. The primer emphasizes that these results are specific to the controlled environment. In real-world RF applications, expect to encounter environmental noise and interference, signal fading and multipath effects, hardware imperfections, and limited or imbalanced training data. Accuracies in the 70-95% range are typical depending on signal-to-noise ratio and environmental conditions. For professional applications, the prediction accuracy of well-tuned models trained on datasets magnitudes of samples larger than what is used in this lab series can achieve results with greater than 99% accuracy.

The primer in this module also covers validation techniques beyond simple accuracy. The confusion matrix visualizes which classes are correctly classified and which are confused with each other, revealing systematic misclassifications that may indicate insufficient distinguishing features. The classification report provides per-class precision (of all predictions for a class, what percentage were correct), recall (of all actual instances, what percentage did the model find), and F1-score (harmonic mean balancing both metrics). ROC curves plot the true positive rate against

the false positive rate at various classification thresholds, with the Area Under the Curve (AUC) providing a threshold-independent performance metric.

The core lab implements a 1D CNN for classifying AM, FM, and PSK signals. The signal generation includes configurable signal-to-noise ratio, bringing synthetic data closer to real-world conditions. Each signal uses the two-channel I/Q representation from Module 1, preserving the complex baseband structure captured by SDR hardware.

The CNN architecture used in the lab example processes these two-channel inputs (I/Q data) through convolutional layers that scan across the signal to detect modulation-specific patterns. The first layer of the CNN architecture applies filters designed to capture broad temporal features; 32 filters and a kernel size of 7 are used. The second layer uses 64 filters with a kernel size of 5, which extracts finer details. The kernel size (7 vs 5) determines how many samples are considered at once, with a smaller kernel able to detect 'higher resolution' features. The larger number of filters in the second layer (64 vs 32) can detect more patterns, or connections, between the layers. Max pooling between layers reduces dimensionality while retaining the most significant features. Adaptive average pooling produces a fixed-size output regardless of input signal length, enabling the network to handle variable-length captures. Fully-connected layers map the extracted features to class probabilities, with dropout preventing overfitting. Students are encouraged to explore the impact of parameters such as differing kernel size, filter amounts, and layers.

Model training uses cross-entropy loss with the Adam optimizer. The training loop iterates through batched data, computing forward passes through the network, calculating loss against true labels, backpropagating gradients, and updating weights. Training progress is observable through loss values printed at regular intervals, with final performance evaluated using classification reports and confusion matrices. Figure 11 shows a representative confusion matrix from the core lab, demonstrating classification accuracy across the three modulation types. This is an ideal result due to the distinct characteristics between the three modulation classes in synthetic data. Students will be able to replicate and experiment with other parameters to see how they impact classification accuracy.

The module includes a challenge exercise to implement an RNN-based classifier using LSTM layers and compare performance against the CNN approach. The RNN processes the time-series sequentially, using the final hidden state (which contains information about the entire sequence) for classification. Comparing training time, accuracy, and performance across different SNR levels reveals how the two architectures handle the same classification task differently.

2.8.2 Learning Outcomes

After completing this module, students will understand when to apply SVM, neural networks, CNN, or RNN approaches based on data characteristics and problem requirements. The exercises provide practical experience implementing 1D CNNs for signal classification using complex I/Q representation, training models with PyTorch, and evaluating performance using confusion matrices, classification reports, and ROC curves.

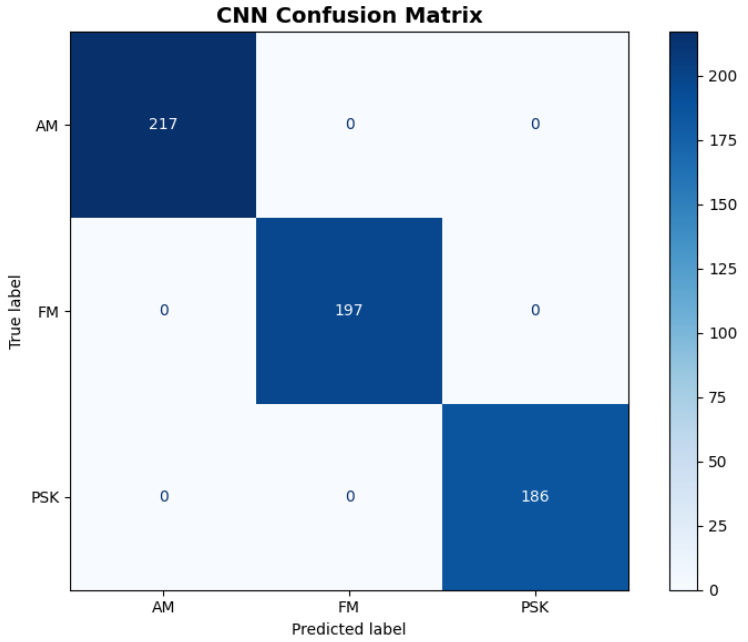

Figure 11: Confusion matrix for AM/FM/PSK classification using the 1D CNN classifier. The diagonal shows correct classifications while off-diagonal elements reveal misclassifications between modulation types.

2.9 Module 6: Dataset Creation

2.9.1 Description

The high accuracy performance for the ML models trained on synthetic data observed in Module 5 does not transfer directly to real-world signals. Models trained exclusively on synthetic data often under perform when used to classify captured signals due to channel impairments, noise characteristics, and hardware effects absent from idealized simulations. This module addresses the sim-to-real gap by creating labeled datasets from actual RF transmissions, formatted for compatibility with TorchSig's 'Bring Your Own Data'(BYOD) [11] workflow used in Module 7.

The module begins with dataset design principles that determine classifier performance. Class balance ensures the model does not become biased toward overrepresented modulation types, with a target of 200-500 samples per class providing sufficient diversity without excessive capture time. SNR variation occurs naturally when capturing at different transmitter-receiver distances, providing the model with examples across a range of signal quality conditions. Sample length must match the input dimensions expected by downstream classifiers; the module uses 4096 I/Q samples per window, corresponding to approximately 4 ms of signal at 1 MHz sample rate and capturing multiple symbol periods per training example.

The dataset targets five modulation types: 2-FSK, 4-FSK, Gaussian FSK (GFSK), Minimum Shift Keying (MSK), and On-Off Keying (OOK). Class labels must follow TorchSig naming conventions exactly (`2fsk`, `4fsk`, `2gfsk`, `2msk`, `ook`) to ensure compatibility with pre-trained

models and the BYOD workflow.

The capture workflow uses an SDR to capture signals, each with 30 seconds of capture, at 1 MHz sample rate with 7,300 potential training windows. The transmit workflow can use either another SDR or provided CC1101 presets, all operating in the unlicensed 433 MHz ISM band.

Signal processing functions transform raw captures into training-ready windows. DC offset removal subtracts the mean from captured samples to eliminate the zero-frequency spike common in direct-conversion receivers. The segmentation function divides continuous I/Q streams into fixed-length windows with configurable overlap. Power-based filtering removes windows containing only noise by calculating mean power for each window and applying a threshold relative to the maximum observed power. This active signal detection ensures training data contains valid signal examples rather than noise-only segments that would confuse the classifier. A typical 30-second capture yields 200-500 usable windows after filtering, depending on transmission duty cycle and distance.

The module implements TorchSig's BYOD format, which consists of three files that together enable seamless integration with TorchSig's data loading infrastructure. The `data.npy` file stores all I/Q windows as a complex64 NumPy array with shape (n_samples, 4096). The `metadata.csv` file provides per-sample information including index, label string, class index, and sample rate. The `info.json` file contains global dataset information including total size, sample dimensions, class labels, class counts, and creation timestamp. This standardized format allows TorchSig to load custom datasets without modification to its core code.

A custom file handler class implements the interface required by TorchSig's `ExternalTorchSigDataset`. The handler provides three essential methods: `size()` returns the number of samples, `load_dataset_metadata()` returns dataset-level configuration, and `load(idx)` retrieves individual samples with their metadata. The metadata returned for each sample must include `class_name` and `class_index` keys as required by TorchSig's internal processing. Testing the file handler independently before proceeding to Module 7 verifies that samples load correctly and all required fields are present.

Dataset validation confirms quality before training. Time-domain visualization displays I and Q components for multiple samples from each class, revealing characteristic amplitude patterns that distinguish OOK (discrete on/off levels) from constant-envelope modulations (continuous oscillations). Figure 12 shows representative samples from each modulation type, illustrating the distinct temporal signatures that classifiers learn to recognize. Spectrogram visualization using STFT reveals frequency content evolution over time, with FSK variants showing discrete frequency levels, GFSK displaying smoother transitions, and OOK exhibiting broadband spectral content during keying transitions. Figure 13 presents spectrograms for each class, demonstrating the time-frequency signatures that complement time-domain features.

Class distribution analysis ensures balanced representation across modulation types. The module generates bar charts showing sample counts per class and computes balance metrics including mean samples per class, minimum and maximum counts, and imbalance ratio. Significant imbalance (ratios exceeding 2:1) may require resampling or weighted loss functions during training. The verification function performs comprehensive format checking, confirming file existence, data types, required columns, consistency between files, and successful file handler

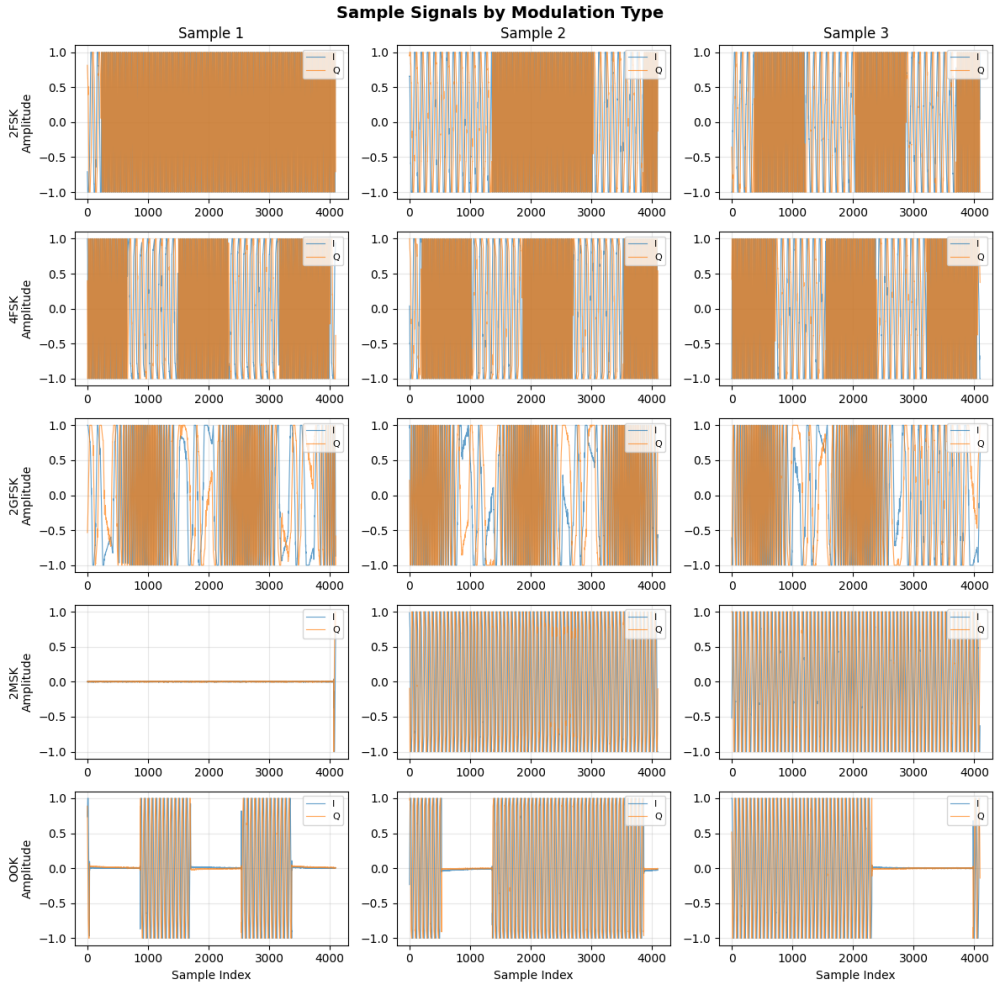

Figure 12: Time-domain visualization of captured I/Q samples for each modulation type. Each row shows multiple samples from one class, with in-phase (I) and quadrature (Q) components plotted against sample index. OOK exhibits discrete amplitude levels while FSK variants maintain constant envelope with frequency-encoded information.

initialization. While data processing and cleaning is beyond the scope of these labs, it is a valuable skill that can be covered by an instructor in preparation for working with corrupted data or non-ideal collection methods.

The module concludes by discussing the relationship between dataset characteristics and classifier performance. Natural SNR variation from distance changes provides robustness to varying signal conditions. Hardware effects including phase noise, frequency offset, and nonlinearities are captured in real data but absent from synthetic generation. Multipath and fading in the capture environment provide realistic channel impairments. These factors collectively address the sim-to-real gap that limits models trained exclusively on synthetic data. A pre-captured reference dataset is provided for students without hardware access, enabling completion of subsequent modules while acknowledging that hands-on capture experience is pedagogically valuable.

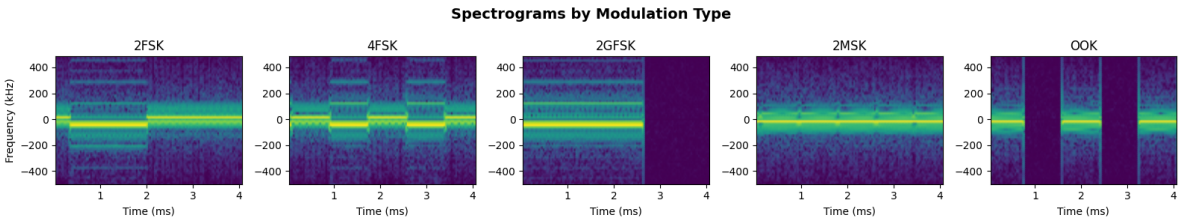

Figure 13: Spectrograms showing time-frequency content for each modulation type. 2-FSK and 4-FSK display discrete frequency levels, GFSK shows Gaussian-smoothed transitions between frequencies, MSK exhibits linear phase ramps, and OOK produces broadband spectral splatter during amplitude transitions.

2.9.2 Learning Outcomes

After completing this module, students will understand the design principles underlying effective RF training datasets, including class balance, SNR diversity, and parameter alignment with downstream classifiers. The exercises provide practical experience configuring SDR, implementing signal processing pipelines for DC removal and power-based filtering, and formatting datasets in TorchSig's BYOD format.

2.10 Module 7: Advanced Classification

2.10.1 Description

With labeled datasets prepared in Module 6, students advance to sophisticated classification using TorchSig, a specialized library for RF signal processing built on PyTorch. The module spans three notebooks that progress from foundational concepts through production-ready classification: a primer introducing TorchSig capabilities, a core lab implementing state-of-the-art architectures, and an optional lab with a transfer learning exercise demonstrating model adaptation to new classes.

In Module 5, students generated synthetic RF signals manually to learn RF fundamentals and explore how noise affects different modulation techniques. While useful for understanding signal processing basics, manual generation cannot easily replicate the complexity of real-world signals. TorchSig provides pre-built professional-quality modulation generators, standardized dataset formats using I/Q samples consistent with the RF machine learning community, realistic impairments including channel fading, interference, extended noise models, and hardware effects that make data less idealized than previous synthetic signals, and reproducibility through standardized processes for generating noise and complex signal features that enable comparison across research efforts[11].

The primer notebook begins with TorchSig installation and signal generation for eight modulation types: OOK, 2-FSK, 8-PSK, BPSK, QPSK, AM, FM, and 4-FSK. Each signal is visualized in three representations: constellation diagrams showing I/Q relationships, time-domain plots of both components, and frequency-domain spectra. Figure 14 shows this multi-view visualization for representative modulation types. Comparing TorchSig-generated signals against the manual synthetic signals from earlier modules reveals increased complexity from realistic pulse shaping

and channel effects.

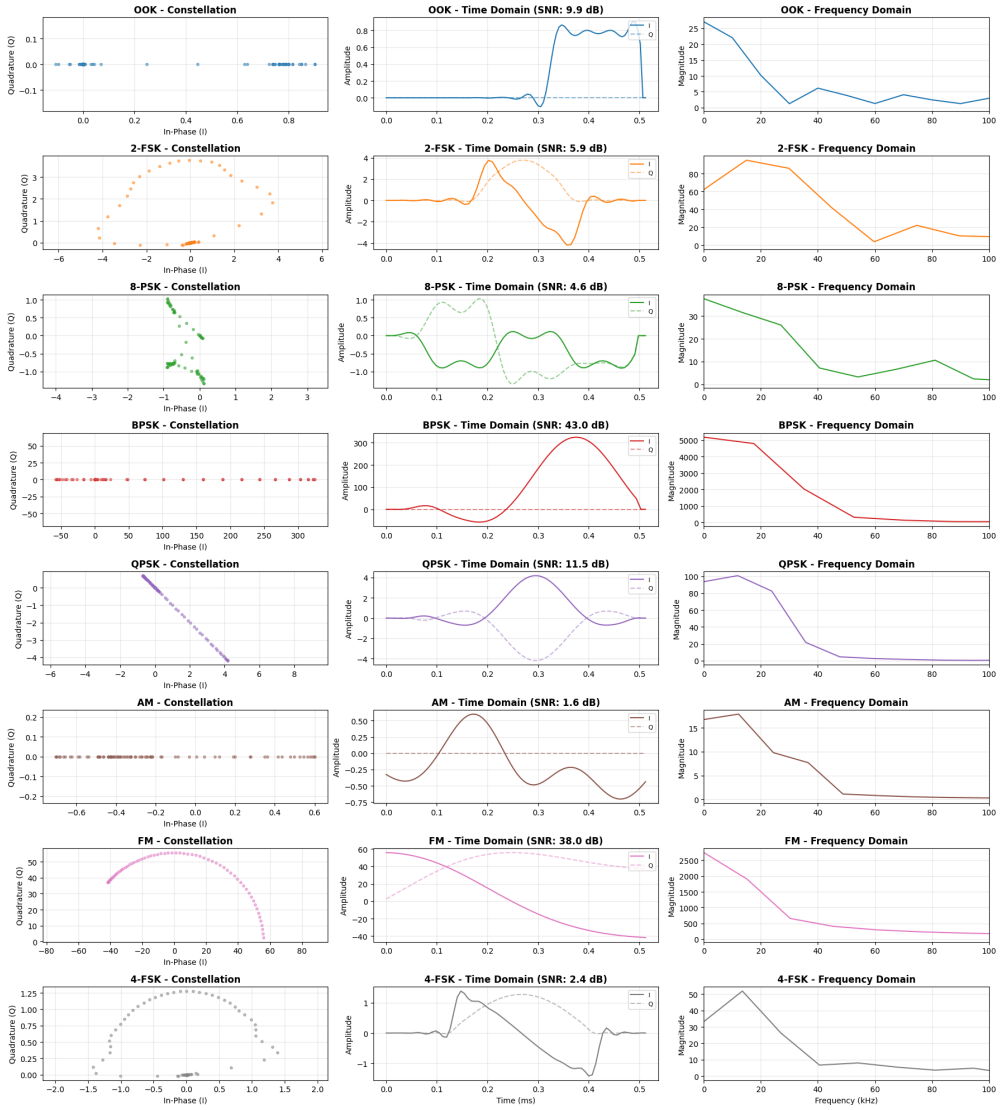

Figure 14: TorchSig signal visualization showing constellation diagram, time domain, and frequency domain representations for OOK, 2-FSK, 8-PSK, BPSK, QPSK, AM, FM, and 4-FSK modulation types. Each row corresponds to one modulation scheme, illustrating distinctive patterns across all three representations.

Two machine learning architectures are implemented in the primer using pytorch for baseline comparison. A 1D CNN architecture processes raw I/Q samples through three convolutional layers with batch normalization, ReLU activation, max pooling, and dropout regularization. Adaptive average pooling produces fixed-size output regardless of input signal length, and fully-connected layers map extracted features to class predictions. A bidirectional LSTM with attention mechanism provides an alternative approach, processing the signal sequentially while weighting important timesteps through learned attention weights. Both architectures include early stopping based on validation accuracy to prevent overfitting. Figure 15 shows training curves for

both models, and Figure 16 presents confusion matrices comparing their classification performance.

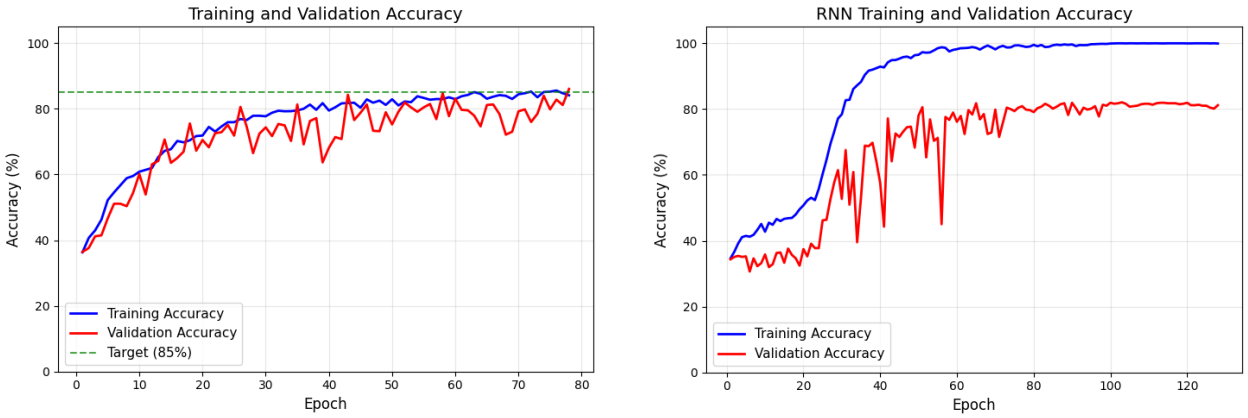

Figure 15: Training and validation accuracy curves for CNN (left) and RNN (right) classifiers in the primer notebook. Both architectures show convergence, with the CNN typically training faster while the RNN captures longer-range temporal dependencies.

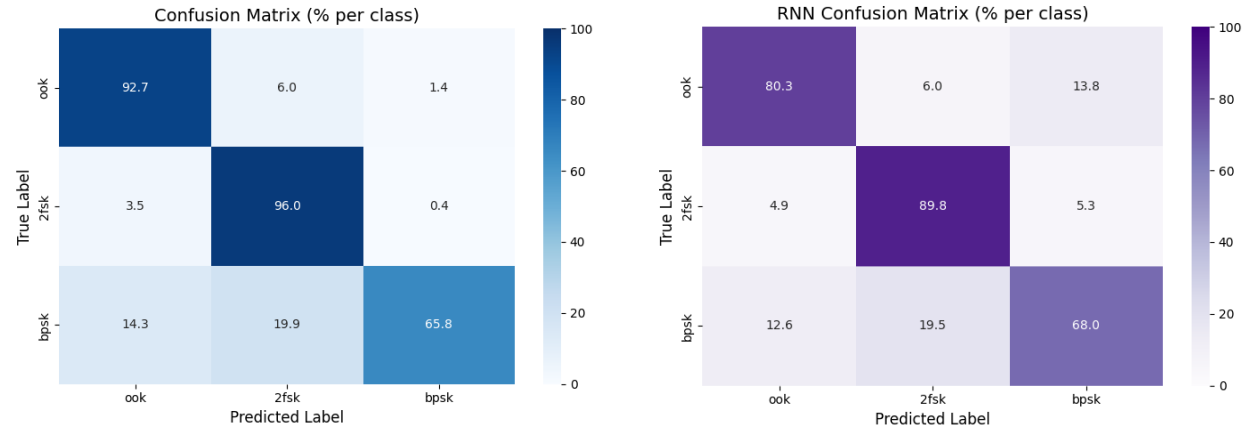

Figure 16: Confusion matrices comparing CNN (left) and RNN (right) classifier performance on OOK, 2-FSK, and BPSK test data. Diagonal elements show correct classifications while off-diagonal elements reveal which modulation pairs are most commonly confused.

The primer also explores spectrogram-based 2D CNNs, which convert I/Q signals to time-frequency representations using TorchSig's built-in `Spectrogram` transform. The Short-Time Fourier Transform (STFT) creates 2D images where the vertical axis represents frequency bins, the horizontal axis represents time bins, and color intensity indicates power at each time-frequency point. Different modulation types create distinctive visual patterns: OOK produces on-off spectral signatures, FSK variants show discrete frequency levels, and PSK creates complex spectral patterns from phase transitions. 2D CNNs with `Conv2d` layers process these spectrogram images similarly to image classification, often achieving strong performance because they can detect patterns in both frequency and time dimensions simultaneously.

The primer concludes with an impairment robustness study comparing models trained on clean

versus impaired data. TorchSig provides three impairment levels: level 0 produces clean signals ideal for baseline testing, level 1 simulates cabled/laboratory conditions with transmitter hardware impairments, and level 2 introduces full wireless channel effects including fading. Identical architectures are trained on clean and impaired datasets, then both models are evaluated on both test sets [11]. Models trained exclusively on clean data typically show significant performance degradation when tested on impaired signals, while models trained on impaired data maintain reasonable accuracy across both conditions. This experiment demonstrates the importance of training with realistic impairments for robust real-world deployment.

The core lab notebook builds production-ready classifiers using TorchSig's pre-built architectures. The lab begins by exploring how SNR and impairment levels affect classification difficulty. Signals generated across SNR ranges (low: 0-20 dB, medium: 20-40 dB, high: 40-60 dB) demonstrate how constellation diagrams degrade as noise increases. At low SNR, constellation points spread and merge, making visual identification difficult. Impairment effects are similarly visualized, showing how frequency offset rotates constellations, IQ imbalance distorts symmetry, and phase noise causes point spreading.

The core lab generates a 5-class production dataset containing 2-FSK, 4-FSK, GFSK, MSK, and OOK signals. Dataset parameters are carefully configured: signal length of 4096 I/Q samples provides sufficient temporal context for classification, 1 MHz sample rate matches the configuration used in Module 6 dataset creation, SNR range of 10-30 dB represents realistic operational conditions.

Data preparation follows standard machine learning practices. The dataset is split into training (70%), validation (15%), and test (15%) sets using stratified sampling to maintain class balance across splits. Complex I/Q data is converted to two-channel tensors with separate in-phase and quadrature components.

The core lab implements TorchSig's XCiT-1D (Cross-Covariance Image Transformer for 1D signals) [11], an architecture specifically designed for RF signal classification. Unlike traditional self-attention transformers that scale quadratically with sequence length, XCiT uses cross-covariance attention that scales linearly, making it efficient for long signal sequences [24]. The model processes raw I/Q samples directly without requiring spectrogram conversion. Model training in this example uses cross-entropy loss with the Adam optimizer, learning rate scheduling that reduces the rate when validation loss plateaus, and early stopping based on validation accuracy thresholds to prevent overfitting.

The training loop monitors both training and validation metrics each epoch, saving the best model weights based on validation loss. Figure 17 presents the confusion matrix on held-out test data. The classification report provides per-class precision, recall, and F1-score, revealing which modulation pairs are most challenging to distinguish. Trained models are saved as checkpoint files containing model weights, architecture configuration, class labels, and training history for later deployment.

The core lab includes a real-time classification demonstration that loads saved model checkpoints and classifies freshly generated signals. For each test signal, the system displays the ground truth label, predicted class, confidence score, and whether the prediction was correct. Visualization shows constellation diagram, frequency spectrum, and time-domain waveform alongside the

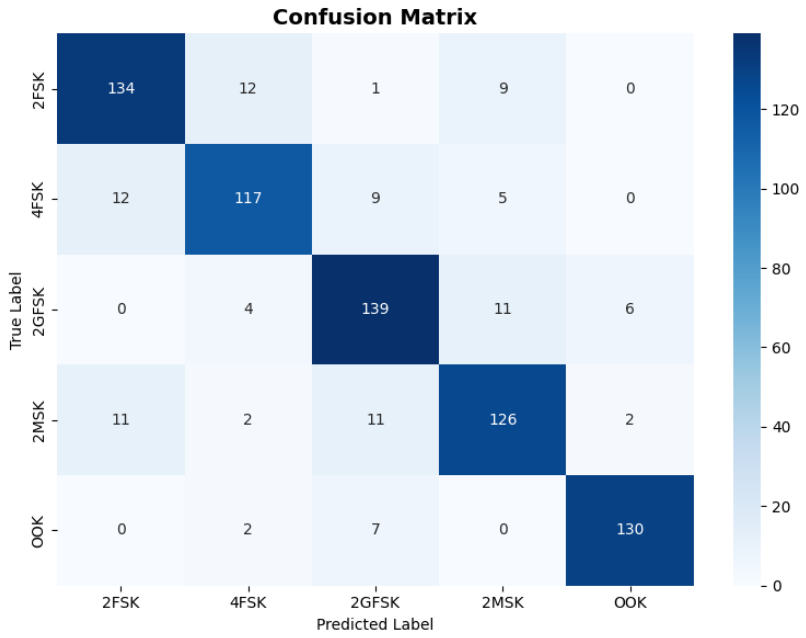

Figure 17: Confusion matrix for XCiT-1D classifier on 5-class test data (2-FSK, 4-FSK, GFSK, MSK, OOK). Strong diagonal dominance indicates accurate classification, with occasional confusion between similar FSK variants.

classification result. This demonstration previews the real-time SDR classification workflow implemented in Module 8.

The core lab also introduces fine-tuning with captured data from Module 6. Models trained exclusively on synthetic data often struggle with real-world signals due to domain shift: the subtle differences between simulated and actual RF captures including hardware-specific frequency response, environmental multipath, and transmitter imperfections. Fine-tuning adapts the pre-trained model to recognize real signal characteristics by continuing training on captured data with a reduced learning rate. Power normalization scales each captured sample to unit power, matching the distribution of synthetic training data. This training workflow significantly improved real signal classification as seen in Figure 18.

The transfer learning notebook demonstrates model adaptation for expanding classification capabilities. Transfer learning leverages pre-trained models as starting points rather than training from random weights, offering reduced training time because models converge much faster.

The transfer learning workflow begins by loading a pre-trained 5-class model checkpoint and reviewing its configuration including architecture, class labels, and training performance. New training data is generated that includes the original five classes plus an additional modulation type, 8-PSK (8-Phase Shift Keying) in the example exercise. The model's final classification layer must be modified to accommodate six output classes rather than five. This involves 'unlocking' the existing model and then replacing the final `Conv1d` grouper layer with a new layer having six output channels.

This workflow freezes all layers except the new classification head, preserving the learned feature

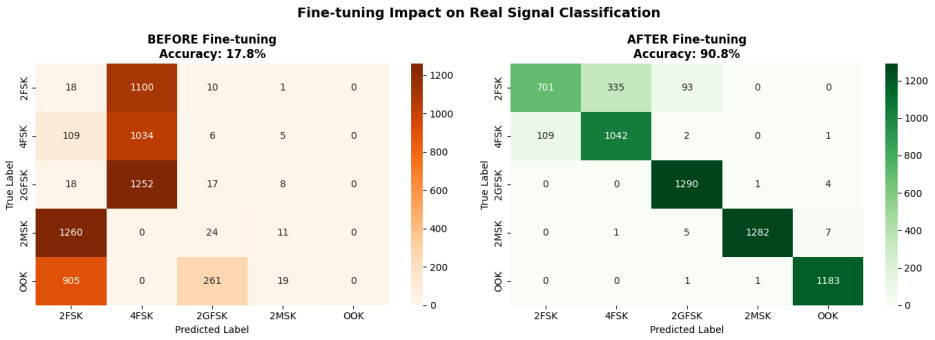

Figure 18: Confusion matrix for synthetic XCiT-1D data classifier (left) and fine-tuned XCiT-1D classifier with data from Module 6 (right). Strong diagonal dominance indicates accurate classification.

representations while training only the new output mapping. This approach requires far fewer training epochs (typically 10-15 versus 50+ for training from scratch) and a lower learning rate to avoid destroying learned features. Training monitors validation accuracy with early stopping, and the fine-tuned model is saved as a new checkpoint. Figure 19 shows performance after transfer learning with the expanded 6-class task.

2.10.2 Learning Outcomes

After completing this module, participants will understand the advantages of TorchSig for realistic RF signal generation including configurable SNR, impairments, and standardized dataset formats. The exercises implement and compare multiple neural network architectures including 1D CNNs, bidirectional LSTMs with attention, spectrogram-based 2D CNNs, and transformer-based XCiT-1D, highlighting their respective strengths for different signal characteristics. Practical experience covers production ML workflows including static dataset generation, stratified train/validation/test splits, early stopping, model checkpointing, and performance evaluation through confusion matrices and classification reports. The transfer learning exercise provides critical skills for adapting pre-trained models to new classification tasks, demonstrating efficient model evolution when requirements change over time.

2.11 Module 8: Real-time SDR Classification

2.11.1 Description

With trained models ready for deployment, this capstone module demonstrates real-time signal classification using concepts from all previous labs. The module covers the complete deployment workflow: loading trained model checkpoints, validating configuration alignment between training and inference parameters, implementing preprocessing pipelines for live I/Q data, running inference on freshly captured signals, and calculating classification accuracy against known ground truth.

Model deployment begins with checkpoint loading and configuration validation to prevent subtle errors during real-time operation. The saved checkpoint contains trained weights and

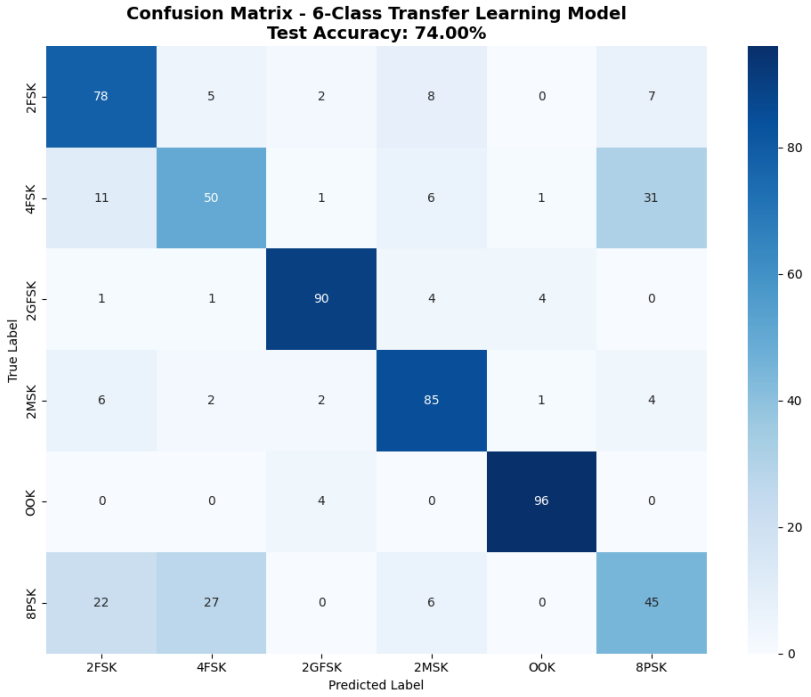

Figure 19: Confusion matrix after transfer learning showing successful adaptation to 6-class classification including 8-PSK. The model maintains accuracy on original classes while correctly classifying the new modulation type.

comprehensive metadata including the model architecture type, input channel configuration (2 channels for I/Q), signal length (typically 4096 samples), number of output classes, and the complete class label list. These parameters must align exactly between training and inference configurations. A signal length mismatch would cause dimension errors during forward pass, while a sample rate mismatch would shift frequency features in ways the model cannot recognize. The validation routine displays all checkpoint parameters alongside the inference configuration for verification before proceeding to live capture.

The preprocessing pipeline converts raw SDR captures to model-ready tensors through standardized transformations. DC offset removal subtracts the complex mean from captured samples, eliminating hardware-induced bias present in many SDR receivers including the B200/B205mini. Continuous capture streams are segmented into fixed-length windows matching the training signal length and I/Q samples are separated into real and imaginary components and converted to PyTorch tensors with shape (batch, 2, signal_length), matching the input format expected by the trained model.

In our example, we use Ettus B205mini SDR capture with the native UHD Python API, providing direct programmatic control over receiver parameters. The script initializes the SDR with specified center frequency, sample rate, and gain, returning a stream command object for capture control. Sample rate alignment is critical: the inference configuration must match the rate used during dataset creation and model training. Center frequency should align with the transmitter being classified; the CC1101 example operates at 433.92 MHz in the ISM band. Receiver gain

requires empirical adjustment: starting at 40 dB provides a reasonable baseline, with increases needed for weak signals and decreases if samples show clipping.

The real-time classification loop implements continuous capture-and-classify operation with operator feedback. Each iteration captures a configurable duration of samples, preprocesses the capture, runs batch inference on all valid windows, and reports results. Majority voting across windows within each capture provides robust classification: rather than reporting the prediction for each individual window, the system tallies votes and reports the class with the highest count along with the vote fraction. This aggregation strategy significantly improves reliability because individual windows may contain partial transmissions, interference, or other anomalies, while the majority across many windows reflects the dominant signal present. Figure 20 shows the live capture visualization.

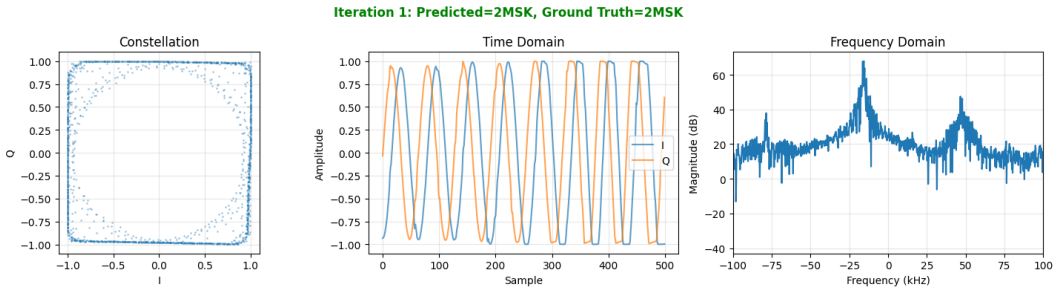

Figure 20: Live SDR capture visualization showing time-domain I/Q samples, frequency spectrum, and prediction distribution across windows. Majority voting aggregates per-window predictions for robust classification of each capture.

The module demonstrates the performance gap between synthetic validation accuracy and real-world over-the-air accuracy discussed in Module 6. Models achieving 95%+ validation accuracy on TorchSig-generated synthetic data may show reduced performance on captured signals due to hardware-specific frequency response, environmental multipath, interference from other devices, and transmitter imperfections such as frequency offset and phase noise. Typically, models fine-tuned on captured data show improved real-time accuracy compared to models trained exclusively on synthetic signals.

2.11.2 Learning Outcomes

After completing this module, participants will understand the complete pipeline from trained model checkpoint to deployed real-time classifier. The exercises provide practical experience loading checkpoints and validating configuration alignment, implementing preprocessing pipelines, performing batch inference with majority voting, and calculating accuracy metrics.

3 Solutions

Educators can request access to solution materials by contacting the authors. Verification of educational affiliation is required. The open-source codebase remains publicly available, ensuring

students can access all lab materials while solution materials support instructor preparation and assessment activities.

4 Conclusion

This work addresses an important gap in engineering education by providing a comprehensive, hands-on approach to integrating machine learning techniques into radio frequency curricula. The eight laboratory modules presented offer educators a flexible, modular framework for introducing ML concepts into existing RF courses without requiring major curriculum restructuring. By progressing through RF fundamental concepts to advanced machine learning architectures, these labs provide students with the practical experience needed to bridge the traditionally separate domains of RF engineering and machine learning.

The intentional focus on the gap between ideal, simulated signals and actual over-the-air transmissions ensures that students develop ML models that generalize to real world deployment scenarios. The materials provided, including detailed learning outcomes and configurable complexity parameters enable instructors to confidently adopt and adapt this curriculum to their specific educational contexts. Ultimately, this work equips the next generation of wireless engineers with the interdisciplinary skills necessary to design, implement, and optimize intelligent systems that will define future wireless network communications.

References

[1] A. Al-Jumaily, A. Sali, M. Riyadh, S. Q. Wali, L. Li, and A. F. Osman, "Machine learning modeling for radiofrequency electromagnetic fields (rf-emf) signals from mmwave 5g signals," *IEEE Access*, vol. 11, pp. 79 648–79 658, 2023.

[2] B. Clerckx, K. Huang, L. R. Varshney, S. Ulukus, and M.-S. Alouini, "Wireless power transfer for future networks: Signal processing, machine learning, computing, and sensing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 5, pp. 1060–1094, 2021.

[3] A. Santra, P. Wang, G. Shaker, B. S. Mysore, G. Dolmans, Y. Chen, N. Shariati, and A. Pandharipande, "Machine learning-powered radio frequency sensing: A review," *IEEE Sensors Journal*, vol. 25, no. 13, pp. 23 164–23 183, 2025.

[4] K. T. Selvan and K. F. Warnick, *Teaching electromagnetics: Innovative Approaches and Pedagogical Strategies*, 1st ed. CRC Press, 2021.

[5] H. G. Espinosa, T. Fickenscher, N. Littman, and D. V. Thiel, "Teaching wireless communications courses: An experiential learning approach," in *2020 14th European Conference on Antennas and Propagation (EuCAP)*, 2020.

[6] H. Kim, N. Rogers, G. York, and P. Leiffer, "Enhanced learning by visualization applying embedded hands-on in electromagnetics class," *2024 ASEE Annual Conference and Exposition Proceedings*, 2024.

[7] A. Morales Bezeira and A. Silva Sprock, "Experiential teaching paradigms in the learning of antenna theory for undergraduate students," in *Proceedings of the 19th Latin American*

Conference on Learning Technologies (LACLO 2024). Berlin, Heidelberg: Springer-Verlag, 2025, p. 153–168. [Online]. Available: https://doi.org/10.1007/978-981-96-3698-3_12

[8] B. M. Notaroš, "Electromagnetics education and its future and challenges," in *2019 13th European Conference on Antennas and Propagation (EuCAP)*, 2019, pp. 1–4.

[9] L. J. Wong and A. J. Michaels, "Transfer learning for radio frequency machine learning: A taxonomy and survey," *Sensors*, vol. 22, no. 4, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/4/1416

[10] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *Engineering Applications of Neural Networks*, C. Jayne and L. Iliadis, Eds. Cham: Springer International Publishing, 2016, pp. 213–226.

[11] E. Oh, J. Mullins, M. Carrick, M. Vondal, J. Hoffman, F. Leonardo, P. Toliver, and R. Miller, "Torchsig 2.0: Dataset customization, new transforms and future plans," in *Proceedings of the 15th GNU Radio Conference (GRCon25)*, Everett, Washington, USA, September 2025. [Online]. Available: https://events.gnuradio.org/event/26/contributions/752/attachments/220/586/TorchSig_GRCon2025.pdf

[12] E. Karincic, L. Linkous, and E. Topsakal, "Interactive educational platform for digital modulation recognition and signal analysis," in *2026 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, 2026.

[13] K. VonEhr, W. Neuson, and B. Dunne, "Software defined radio: Choosing the right system for your communications course," *2016 ASEE Annual Conference and Exposition Proceedings*, 2016.

[14] T. F. Collins, R. Getz, D. Pu, and A. M. Wyglinski, *Software-Defined Radio for Engineers*, ser. Artech House Mobile Communications Series. Norwood, MA: Artech House, 2018, illustrated Edition.

[15] R. G. Ekanthappa, R. Escobar, A. Matevossian, and D. Akopian, "A remote laboratory for usrp-based software defined radio," *SPIE Proceedings*, vol. 9030, p. 903003, 2014.

[16] Texas Instruments, *CC1101 Low-Power Sub-1 GHz RF Transceiver*, Texas Instruments, Dallas, TX, 2018, rev. SWRS061I. [Online]. Available: https://www.ti.com/lit/ds/symlink/cc1101.pdf

[17] GNU Radio Project, "Gnu radio: The free & open-source toolkit for software radio." [Online]. Available: https://github.com/gnuradio/gnuradio

[18] E. Karincic (dollarhyde), "Radio frequency machine learning signal processing labs." [Online]. Available: https://github.com/Dollarhyde/radio-frequency-machine-learning

[19] S. Haykin and M. Moher, *Communication Systems*, 5th ed. Hoboken, NJ: Wiley, 2014.

[20] A. Goldsmith, *Wireless Communications*. Cambridge, UK: Cambridge University Press, 2005.

[21] O. A. Dobre, A. A. Abdi, Y. Bar-Ness, and W. Su, "Survey of automatic modulation classification techniques: classical approaches and new trends," *IET Communications*, vol. 1, pp. 137–156, 2007.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[23] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," *IEEE Transactions on Cybernetics*, vol. 50, no. 8, pp. 3668–3681, 2020.

[24] A. Ali, H. Touvron, M. Caron, P. Bojanowski, M. Douze, A. Joulin, I. Laptev, N. Neverova, G. Synnaeve, J. Verbeek, and H. Jegou, "Xcit: Cross-covariance image transformers," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 20 014–20 027. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/a655fbe4b8d7439994aa37ddad80de56-Paper.pdf