DEVELOPMENT OF ENHANCED MACHINE LEARNING METHODS FOR THE AUTOMATIC MODULATION CLASSIFICATION OF QAM SIGNALS

by Andrii Savenkov

A Capstone Project

Presented in Partial Fulfillment of the Requirements for the Degree

Master

American University Kyiv

2025

APPROVED BY:

Vasyl Semenov, Ph.D., Dr. Habil.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This Capstone Project considers Automatic Modulation Classification (AMC) by evaluating multiple machine learning techniques for the classification of five common digital modulation schemes: Binary Phase Shift Keying (BPSK), Quaternary Phase Shift Keying (QPSK), 8PSK, 8-Quadrature Amplitude Modulation (8QAM), and 16QAM. Four distinct classification methods were applied: Gaussian Mixture Models (GMMs), multinomial regression with polynomial feature expansion, nearest neighbor approach and Convolutional Neural Networks (CNNs). The study also introduces a novel software architecture that integrates these methods into a web service developed using the Elixir and Phoenix framework, coupled with Python-based model implementations. This integrated architecture facilitates the deployment and interaction between the classification models and the user. Experimental results demonstrated high accuracy in classifying simulated signals, with CNNs achieving 99% accuracy and GMM and multinomial regression methods attaining 98% accuracy. Testing with real signals from a CDM-625 modem validated these findings, maintaining 99% accuracy across all approaches. This research offers a comparative analysis of different classification methods and provides a practical, integrated software solution for deployment in modern communication systems. The results contribute to improving spectrum utilization and offer a foundation for adaptive modulation strategies in military and commercial applications.

Keywords: Automatic Modulation Classification, Machine Learning, Digital Signal Processing, Convolutional Neural Networks, Gaussian Mixture Models, Higher Order Cumulants, Software-Defined Radio.

CHAPTER 1: EXPLORATION OF SOLUTIONS, LITERATURE, AND DEVELOPMENT TASKS

1.1 Review of Existing Solutions

Automatic Modulation Classification (AMC) involves identifying a received signal's modulation format (e.g., BPSK, QPSK, 8PSK, QAM) without prior knowledge of the underlying parameters. This capability is crucial for both military and commercial communication systems, as it enables dynamic resource allocation, interference detection, and adaptive modulation strategies. In recent years, significant research has focused on machine learning-based approaches that can address the increasing complexity and diversity of modulation schemes.

The capstone project specifically focuses on classifying BPSK, QPSK, 8PSK, 8QAM, and 16QAM signals. These modulations cover a range of spectral efficiencies and are widely used in modern communication systems. The goal is to explore and compare the performance of multinomial regression, convolutional neural networks (CNNs), Gaussian mixture models (GMMs), and nearest-neighbor classification in this AMC task.

Existing solutions include traditional approaches such as feature-based approaches (Gaussian mixture models, Nearest-Neighbor approaches (k-NN), or Multinomial (Softmax) Regression) and modern machine learning solutions such as Convolutional Neural Networks belonging to category of Deep Learning. Below is comparative analysis on each of these methods.

Feature-Based Methods

The feature-based methods in AMC relied on feature extraction from the in-phase (I) and quadrature (Q) components, or from their transformations (e.g., amplitude, phase, or frequency domains). Researchers would derive metrics such as higher-order moments or cyclostationary features, then feed these extracted features into traditional machine learning algorithms (e.g., SVM, k-NN, or decision trees).

- **Advantages**
  - Lower computational overhead compared to deep learning.
  - More interpretable, as the chosen features often directly reflect signal properties.
- **Limitations**
  - Feature engineering can be laborious and may require domain expertise.
  - Performance degrades if the chosen features are not robust to noise, fading, or hardware impairments.

- Scalability is challenging when adding more modulation types.

Gaussian Mixture Models (GMMs) are a classic probabilistic model often used for density estimation. In AMC, a GMM can model the distribution of the signal's features (e.g., amplitude or phase histograms) as a mixture of Gaussians.

- **Advantages**
  - Can capture multi-modal distributions inherent in phase or amplitude data (especially relevant for QAM).
  - Straightforward training procedure using the Expectation-Maximization (EM) algorithm.
- **Limitations**
  - May struggle with overlapping clusters if the features do not provide clear separations between modulations.
  - Scalability and convergence time can be issues when the dimensionality of feature space increases.

Nearest-neighbor classifiers are simple yet often effective for smaller sets of classes like BPSK, QPSK, 8PSK, and QAM variants, provided that the feature space is well-designed or learned.

- **Advantages**
  - Straightforward to implement.
  - Non-parametric, so no explicit model training phase.
- **Limitations**
  - Can suffer from high computational costs at inference time (distance calculations).
  - Performance heavily depends on feature representation.
  - Sensitive to the curse of dimensionality when many features are used.

Multinomial regression, also referred to as Softmax regression, is an extension of logistic regression that can handle multiple classes.

- **Advantages**
  - Interpretable weights indicate the importance of specific features.
  - Can serve as a baseline for classification tasks with relatively low computational overhead.
- **Limitations**
  - Often outperformed by more complex methods (e.g., CNNs) when dealing with high-dimensional data like raw I/Q samples.
  - Accuracy can degrade if the data is highly non-linear and not well separable.

Modern Machine Learning Solutions

Convolutional Neural Networks (CNNs) have emerged as a powerful tool for AMC. By treating I/Q samples or spectrograms as "images," CNNs can automatically learn spatial/temporal patterns that differentiate various modulation types.

- **Architecture**
  - Typical CNNs for AMC start with convolutional layers to detect local features in I/Q data, followed by pooling layers and fully connected layers for classification.
  - Some researchers augment the data using transformations (e.g., frequency shifting or time-domain jitter) to enhance robustness.
- **Advantages**
  - **High Accuracy**: CNNs often outperform traditional models, especially in lower Signal-to-Noise Ratio (SNR) regimes.
  - **Feature Learning**: Feature extraction is embedded in the training process, reducing the need for manual feature engineering.
- **Limitations**
  - **Data Requirements**: Require substantial training data to avoid overfitting.
  - **Computational Cost**: Training can be computationally expensive; deploying large CNN models on embedded systems might be challenging.

1.2 Literature Review

As discussed in the previous section, modern AMC research has evolved from traditional statistical and feature-based methods to more advanced machine learning (ML) and deep learning (DL) techniques. In this literature review, I focus on a selection of studies that have addressed the classification of common modulation schemes such as BPSK, QPSK, 8PSK, and various QAM constellations (e.g., 8QAM, 16QAM). Notably, the methods include multinomial (softmax) regression, Gaussian mixture models (GMMs), nearest-neighbor approaches (k-NN), and Convolutional Neural Networks (CNNs).

O'Shea, Corgan, and Clancy (2016)

Reference:

T. O'Shea, J. Corgan, and T. Clancy, "Convolutional Radio Modulation Recognition Networks," *Engineering Applications of Neural Networks*, vol. 629, pp. 213–226, 2016.

Context and Approach:

O'Shea et al. present some of the earliest work applying deep learning to AMC. Their study tackles classification tasks for several modulation types, including BPSK, QPSK, 8PSK, and QAM variants. They propose a CNN-based framework that directly processes in-phase (I) and quadrature (Q) samples as though they were image channels. This approach eliminates the need for extensive feature engineering.

Key Findings:

- **Improved Accuracy**: CNNs exhibited superior performance relative to classical methods (e.g., decision trees or SVM) across a range of SNR conditions.
- **Data Requirements**: Larger labeled datasets significantly boosted performance, highlighting data-driven advantages of deep learning.
- **Computational Aspects**: Though CNN training was relatively resource-intensive, inference could be optimized for real-time applications with careful model compression.

Limitations and Future Work:

- The authors primarily focused on synthetic datasets. Real-world factors such as hardware impairments or multipath fading were less thoroughly addressed.
- The approach can be extended to more complex or newly emerging waveforms by retraining on updated datasets.

Amin et al. (2019)

Reference:

R. Amin, F. Bellili, S. Affes, and A. Ghrayeb, "On the Performance of Automatic Modulation Classification in Flat-Fading Channels: A Machine Learning Approach," *IEEE Transactions on Communications*, vol. 67, no. 10, pp. 6487–6500, 2019.

Context and Approach:

This paper focuses on flat-fading channels and examines a hybrid ML pipeline that combines **Gaussian mixture models (GMMs)** with **nearest-neighbor** classifiers. The authors first extract several time-domain and frequency-domain features—such as the instantaneous amplitude and phase characteristics—then apply GMM to model the probability distributions. Finally, they employ a nearest-neighbor decision rule to finalize the modulation class.

Key Findings:

- **Robustness to Fading**: The GMM-based modeling offered a more flexible representation of the signal distributions compared to single Gaussian or purely deterministic approaches.

- **SNR Variability**: The combined approach maintained respectable accuracy down to relatively low SNR levels, showcasing stability against channel-induced degradation.

Limitations and Observations:

- GMMs can be computationally heavier to train, especially for higher-dimensional feature sets.
- If two modulation schemes share similar statistical properties under fading, misclassification rates can rise unless additional distinguishing features (e.g., cyclostationary characteristics) are incorporated.

Liu, Yang, and El Gamal (2018)

Reference:

X. Liu, D. Yang, and A. El Gamal, "Deep Neural Network Architectures for Modulation Classification," arXiv preprint arXiv:1712.00443, 2018.

Context and Approach:

Liu et al. investigate the application of deep learning for Automatic Modulation Classification (AMC), building on prior work by extending convolutional neural network (CNN) architectures. They leverage the RadioML2016.10b dataset, which captures various wireless channel imperfections, to classify ten modulation types, including BPSK, QPSK, QAM16, and AM-DSB. The authors introduce multiple neural network architectures—CNN, ResNet, DenseNet, and CLDNN—and explore their performances across different Signal-to-Noise Ratios (SNRs). Each architecture is designed with increasing depth and innovative connections to enhance feature extraction.

Key Findings:

- **Improved Accuracy:** CLDNN achieves the best accuracy of 88.5% at high SNR, outperforming CNN, ResNet, and DenseNet.
- **Impact of Depth:** Increasing network depth improves accuracy up to a limit, after which performance degrades due to gradient vanishing.
- **Training Complexity:** DenseNet required the most training time (70 hours), followed by CLDNN (50 hours).

Limitations:

- The work focuses on synthetic datasets and does not consider real-world impairments like hardware mismatches or multipath fading.
- Enhancing feature preprocessing and deeper architectures could address significant misclassifications (e.g., QAM16 vs. QAM64).

Harper, Thornton, and Larson (2023)

Reference:

C. A. Harper, M. A. Thornton, and E. C. Larson, "Automatic Modulation Classification with Deep Neural Networks," arXiv preprint arXiv:2301.11773, 2023.

Context and Approach:

Harper et al. conduct a comprehensive analysis of CNN-based architectures for AMC, focusing on enhancing the RadioML2018.01A dataset. They explore various design elements through ablation studies, introducing novel architectural modifications such as dilated convolutions, squeeze-and-excitation (SE) blocks, and self-attention. Their approach highlights the impact of design choices on classification performance across varying SNRs and modulation types.

Key Findings:

- **New State-of-the-Art:** A combination of SE blocks, dilated convolutions, and optimized kernel configurations achieved an average accuracy of 63.7% and a peak accuracy of 98.9%.
- **Performance at Low SNR:** Dilated convolutions significantly improved classification in noisy environments (SNR < 0 dB).
- **Model Efficiency:** The X-vector-inspired architecture allowed the model to handle variable signal durations without retraining.

Limitations:

- Self-attention did not improve performance, possibly due to the increased model complexity.
- Investigating real-world deployment scenarios and low-SNR performance remains an open challenge.

1.2.5 Key Observations and Research Gaps

Data-Driven Methods Prevail

Deep learning—particularly CNNs—tends to outperform traditional models like multinomial regression or GMM + k-NN pipelines when sufficiently large datasets are available. Nonetheless, classical methods still hold value in situations with limited data or strict interpretability constraints.

Need for Realistic Datasets

Many studies rely on synthetic or lab-generated data, which may not encapsulate real-world channel fading, interference, or hardware non-idealities. More robust methods that generalize to real-world scenarios remain a pressing challenge.

Model Complexity vs. Resource Constraints

While complex CNNs can achieve high accuracy, they demand significant computational resources. For resource-limited or real-time applications, model compression or simpler algorithms like nearest-neighbor might be more practical—albeit with some sacrifice in accuracy.

Expanding Class Diversity

Most research focuses on a limited number of standard modulation types. As wireless communication evolves, new or proprietary modulation schemes arise, highlighting a need for either flexible (few-shot learning) or incremental learning approaches.

CHAPTER 2: DEVELOPMENT OF MACHINE LEARNING APPROACHES FOR AUTOMATIC MODULATION CLASSIFICATION OF QAM SIGNALS

2.1 Introduction

The rapid evolution of wireless communication technologies has made the automatic classification of modulation schemes an increasingly critical component in modern communication systems. Automatic Modulation Classification (AMC) plays a vital role in various applications, from non-cooperative communication to software-defined radio, where the characteristics of received signals are not known a priori. That is why the detection and classification of modulation types for unknown signals still remains an important technical and scientific problem.

This chapter presents an investigation into enhanced machine learning approaches for the classification of various digital modulation schemes, specifically focusing on Binary Phase Shift Keying (BPSK), Quaternary Phase Shift Keying (QPSK), 8PSK, 8QAM, and 16QAM signals. We explore and compare several classification methodologies, including feature-based approaches such as multinomial regression and nearest-neighbor classification, probabilistic methods like Gaussian mixture modeling (GMM), and deep learning techniques utilizing convolutional neural networks (CNN).

By developing robust classification methods, this work contributes to the advancement of adaptive communication systems and cognitive radio technologies. Our investigation encompasses both theoretical analysis and practical implementation, with performance evaluation conducted using both simulated data and real signals from commercial communication equipment.

2.2 Description of Signals

Signals in communication systems are the core medium for transmitting information. They are typically characterized by their modulation, frequency, amplitude, and phase properties. This section provides a detailed description of the signals considered in the context of this research, focusing on their mathematical modeling, characteristics, and relevance to Automatic Modulation Classification.

2.2.1. Signal Types

The study involves five key modulation types commonly used in digital communication systems:

- **Binary Phase Shift Keying (BPSK):** A modulation scheme where each bit is represented by a distinct phase of the carrier signal, typically 0° and 180°.

- **Quadrature Phase Shift Keying (QPSK):** Uses four distinct phases (e.g., 0°, 90°, 180°, and 270°) to represent two bits per symbol.
- **8-Phase Shift Keying (8PSK):** Extends phase-shift keying to 8 distinct phases, enabling three bits per symbol.
- **8-Quadrature Amplitude Modulation (8QAM):** Combines amplitude and phase modulation with 8 points in the constellation.
- **16-Quadrature Amplitude Modulation (16QAM):** Uses 16 distinct amplitude-phase combinations, providing higher data rates.

These signals differ in their complexity and information-carrying capacity, which influences the classification challenge.

2.2.2 Signal Model

The general form of the transmitted signal is represented as:

$$z(t) = a \exp\big(j(\phi + \omega t)\big) \sum_{n=-\infty}^{\infty} s(n)h(t - nT - \tau) + w(t),$$

where:

- $s(n)$ represents the sequence of original symbols, varying by modulation type:
  - BPSK: 2 symbols
  - QPSK: 4 symbols
  - 8PSK: 8 symbols
  - 8QAM: 8 symbols
  - 16QAM: 16 symbols
- T represents the period between symbols
- a represents the signal amplitude
- ϕ represents the phase
- τ represents the time shift
- ω represents the carrier frequency
- $h(t)$ represents how the channel responds to the signal
- $w(t)$ represents background noise

This model accounts for real-world conditions, including noise and channel effects, making it representative of practical scenarios.

2.2.3 Signal Characteristics

- **Constellation Diagrams:** Each modulation type is represented by a constellation in the complex plane, with points corresponding to possible symbols. For example:
    - BPSK: Two points along the real axis.
    - QPSK: Four points evenly spaced around a circle.
    - QAM: Combines amplitude and phase variations to form grid-like constellations.
- **Noise and Interference:** Real-world signals are corrupted by noise, inter-symbol interference, and channel fading, increasing the difficulty of AMC tasks.

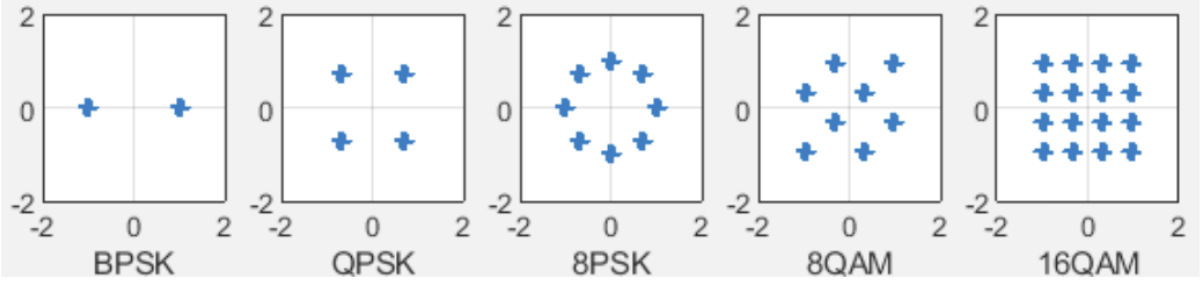

Figure 1: Constellations for different signal classes.

2.3 Feature Extraction and Analysis

The effectiveness of modulation classification significantly depends on the selection and processing of signal features. In this study, we focus on two primary types of features: Higher Order Cumulants (HOC) and their polynomial expansions.

2.3.1 Higher Order Cumulants

Higher Order Cumulants (HOC) serve as informative features for signal classification tasks. For a complex signal z, the (p, q) statistical moments are defined as:

$$M_{pq}(z) = E\{z^{p-q}(z^*)^q\},$$

where z^* represents the complex conjugate of the signal. These cumulants provide valuable information about the signal's characteristics and have proven effective in distinguishing between different modulation types.

2.3.2 Feature Normalization

A crucial aspect of feature processing is normalization. All features must be scaled to have approximately the same range to facilitate effective training of both regression models and neural networks. This normalization process ensures that no single feature dominates the classification decision due to its scale rather than its information content.

2.3.3 Feature Distribution Analysis

The distribution analysis of features C20 and C61 reveals interesting patterns:

- For the four-class problem (BPSK, QPSK, 8PSK, 8QAM), these features show reasonable separation between classes
- When extended to five classes (including 16QAM), these features alone become insufficient for complete class separation, indicating the need for additional cumulants to achieve effective classification

2.3.4 Polynomial Features

To address the non-linear nature of the classification boundaries, polynomial features are employed as an enhancement technique. While higher-order polynomial classifiers are possible, second-order polynomial features offer a practical balance between complexity and effectiveness. For an initial feature vector:

$$x = [x_1, x_2, \dots, x_p].$$

The expanded feature vector includes:

$$x' = [x_1, x_2, \dots, x_p, x_1x_2, \dots, x_{p-1}x_p, x_1^2, x_2^2, \dots, x_p^2].$$

This expansion results in a feature vector with $0.5p(p+3)$ total coordinates, where p is the number of original features. The polynomial expansion enables:

1. Adjustment of classification boundaries in higher-dimensional space
2. Proper linear distinction between features belonging to different classes
3. Enhanced separation capability while maintaining computational feasibility

The incorporation of polynomial features has proven particularly effective in improving classification accuracy, especially in cases where simple linear boundaries are insufficient for proper class separation.

2.4 Multinomial Classification

Multinomial classification represents an extension of binary classification methods to handle multiple classes simultaneously. In our context, this approach is particularly relevant for distinguishing between five different modulation types: BPSK, QPSK, 8PSK, 8QAM, and 16QAM.

2.4.1 From Binary to Multinomial Classification

To understand multinomial classification, it's helpful to first consider the binary case. In binary logistic regression, for a feature vector x, we aim to find parameters w and b such that:

$$\hat{y}_k = \sigma\big((w, x_k) + b\big),$$

where $\sigma(z)$ is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This function represents the probability of the feature vector x belonging to the positive class, with $1 - \hat{y}_k$ representing the probability for the negative class. The parameters w, b are found by minimizing the cost function:

$$J(w, b) = -\frac{1}{m}\sum_{k=1}^{m} \quad \Big(y_k log\sigma\big((w, x_k) + b\big) + (1 - y_k) log\left(1 - \sigma\big((w, x_k) + b\big)\right)\Big).$$

Note that sigmoid function is an inverse of logit function:

$$logit(p) = log\frac{p}{1-p}.$$

2.4.2 Multinomial Model Structure

In the multinomial case with K classes, the task expands to finding K pairs of parameters (ω_1, b_1), ..., (ω_K, b_K) The probability of a feature vector x belonging to class K is given by:

$$P_k = \frac{exp\big((\omega_k, x) + b_k\big)}{\sum_{l=1}^{K} exp\big((\omega_l, x) + b_l\big)}$$

where the summation in the denominator runs from $l = 1$ to K. This formulation employs the softmax function instead of the sigmoid:

$$softmax(k, z_1, \dots, z_K) = \frac{expz_k}{\sum_{l=1}^{K} expz_l}$$

An important characteristic of the multinomial model is that its parameters are not uniquely identifiable. For instance, shifting all parameters by constant values:

$$\omega_k := \omega_k + \omega'$$
$$b_k := b_k + b'$$

results in the same probability distributions. This property leads to a practical simplification where one class (typically the last class K) can be used as a reference, resulting in:

$$P\{y = k\} = \frac{exp\big((\omega_k, x) + b_k\big)}{1 + \sum_{l=1}^{K-1} exp\big((\omega_l, x) + b_l\big)}, k \leq K - 1,$$
$$P\{y = K\} = \frac{1}{1 + \sum_{l=1}^{K-1} exp\big((\omega_l, x) + b_l\big)}.$$

2.4.3 Loss Function and Training

The model parameters are optimized using a cross-entropy loss function:

$$L(y, \hat{y}) = -\sum_{k=1}^{K} y_k log\hat{y}_k,$$

where y_k equals 1 if k is the correct label for the prediction $\hat{y}_k$ and 0 otherwise. This function generalizes the binary classification cost function to the multi-class case.

2.5 Gaussian Mixture Modeling (GMM)

Gaussian Mixture Modeling represents a probabilistic approach to signal classification that models the distribution of feature vectors as a weighted combination of Gaussian probability densities. This method is particularly effective for modeling complex data distributions that cannot be adequately represented by a single Gaussian distribution.

2.5.1 Mathematical Foundation

The fundamental principle of GMM is to represent the probability density of a d-dimensional feature vector x as:

$$p(x) = \sum_{m=1}^{M} a_m b(x/\mu_m, D_m),$$

where:

- M is the number of Gaussian components
- a_m represents the mixture weights
- $b(x/\mu, D)$ is a Gaussian density function with mean μ and covariance matrix D
- The sum runs from $m = 1$ to M

The Gaussian density function is defined as:

$$b(x/\mu, D) = \frac{1}{\sqrt{2\pi\ detD}} \times exp\ [-0.5(x - \mu)^T D^{-1}(x - \mu)]$$

2.5.2 Model Characteristics

The GMM approach offers several key advantages:

- It divides the feature space into M subclasses
- Provides flexibility similar to vector quantization but with enhanced adaptability
- Each class requires its own separate GMM training
- Parameters to be determined include a_i, μ_i, D_i for $i = 1, \dots, M$ based on training data $X = [x_1, x_2, \dots, x_T]$.

2.5.3 Parameter Estimation

The parameter estimation process involves maximizing the likelihood function:

$$p(X/\lambda) = \prod_{t=1}^{T} p(x_t/\lambda),$$

where:

- $\lambda = \{ a_i, \mu_i, D_i, i = 1, \dots, M\}$
- The product runs from $t = 1$ to T, where T represents the number of training samples.

2.5.4 Training Algorithm

Sincedirect analytical maximization of the likelihood function is not possible, the Expectation-Maximization (EM) algorithm is employed. The process involves:

1. Initial approximation using K-means algorithm with uneven dichotomy method
2. Iterative refinement through EM steps
3. Convergence to optimal parameter values

2.5.5 Classification Process

For test data $X = [x_1, x_2, \dots, x_N]$, the classification involves comparing densities:

$$p(X/a_k, \mu_k, D_k)$$

This can be performed in either:

- Linear scale
- Logarithmic scale: $(\sum_{t=1}^{N}(log\ p(x_t/\alpha_k, \mu_k, D_k)))$

The class k that maximizes the log-likelihood function is selected as the classification result.

Note that Nearest Neighbour approach is a private case of GMM when covariance matrices D are identity matrices and all weights a_k are equal: $a_k = 1/M$.

2.6 Convolutional Neural Network

For comparative analysis, we implemented the CNN-based approach to automatic modulation classification of non-mixed signals as proposed in [2]. This approach differs from traditional methods by directly processing signal samples rather than using extracted features.

2.6.1 Network Architecture

The implemented CNN architecture follows a specific structure designed for optimal signal classification. Figure 2 presents the detailed summary of the network architecture:

The network consists of the following key components:

1. Input Layer ('Input Layer')
   - Processes 2-D input images (1×1024×2)
2. Multiple Convolutional Blocks, each containing:
   - 2-D Convolution layer
   - Batch Normalization
   - 2-D Max Pooling
   - ReLU activation
3. Sequential Structure:
4. First block: 16 1×3×2 convolutions with stride [1 1] and padding 'same'
   - Subsequent blocks increase in channel depth (24, 32, 48, 64, 96)
   - Each max pooling layer uses stride [1 2]
   - Final layers include fully connected and softmax classification

1	'Input Layer'	Image Input	1×1024×2 images
2	'CNN1'	2-D Convolution	16 1×1×2 convolutions with stride [1 1] and padding 'same'
3	'BN1'	Batch Normalization	Batch normalization with 16 channels
4	'ReLU1'	ReLU	ReLU
5	'MaxPool1'	2-D Max Pooling	1×2 max pooling with stride [1 2] and padding [0 0 0 0]
6	'CNN2'	2-D Convolution	24 1×1×16 convolutions with stride [1 1] and padding 'same'
7	'BN2'	Batch Normalization	Batch normalization with 24 channels
8	'ReLU2'	ReLU	ReLU
9	'MaxPool2'	2-D Max Pooling	1×2 max pooling with stride [1 2] and padding [0 0 0 0]
10	'CNN3'	2-D Convolution	32 1×1×24 convolutions with stride [1 1] and padding 'same'
11	'BN3'	Batch Normalization	Batch normalization with 32 channels
12	'ReLU3'	ReLU	ReLU
13	'MaxPool3'	2-D Max Pooling	1×2 max pooling with stride [1 2] and padding [0 0 0 0]
14	'CNN4'	2-D Convolution	48 1×1×32 convolutions with stride [1 1] and padding 'same'
15	'BN4'	Batch Normalization	Batch normalization with 48 channels
16	'ReLU4'	ReLU	ReLU
17	'MaxPool4'	2-D Max Pooling	1×2 max pooling with stride [1 2] and padding [0 0 0 0]
18	'CNN5'	2-D Convolution	64 1×1×48 convolutions with stride [1 1] and padding 'same'
19	'BN5'	Batch Normalization	Batch normalization with 64 channels
20	'ReLU5'	ReLU	ReLU
21	'MaxPool5'	2-D Max Pooling	1×2 max pooling with stride [1 2] and padding [0 0 0 0]
22	'CNN6'	2-D Convolution	96 1×1×64 convolutions with stride [1 1] and padding 'same'
23	'BN6'	Batch Normalization	Batch normalization with 96 channels
24	'ReLU6'	ReLU	ReLU
25	'AP1'	2-D Average Pooling	1×32 average pooling with stride [1 1] and padding [0 0 0 0]
26	'FC1'	Fully Connected	5 fully connected layer
27	'SoftMax'	Softmax	softmax
28	'Output'	Classification Output	crossentropyex with '0' and 4 other classes

Figure 2: Summary of implemented CNN

2.6.2 Training Process and Performance

The training progression is visualized in Figure 3, which shows:

1. Training Parameters:
   - 50% data split for training and validation
   - Batch size of 1024
   - Training continued for 25 epochs
2. Performance Metrics:
   - Achievement of approximately 90% accuracy on validation data
   - Convergence of cross-entropy loss
   - Stable training progression

The confusion matrix presented in Figure 4 demonstrates the classification performance across different modulation types:

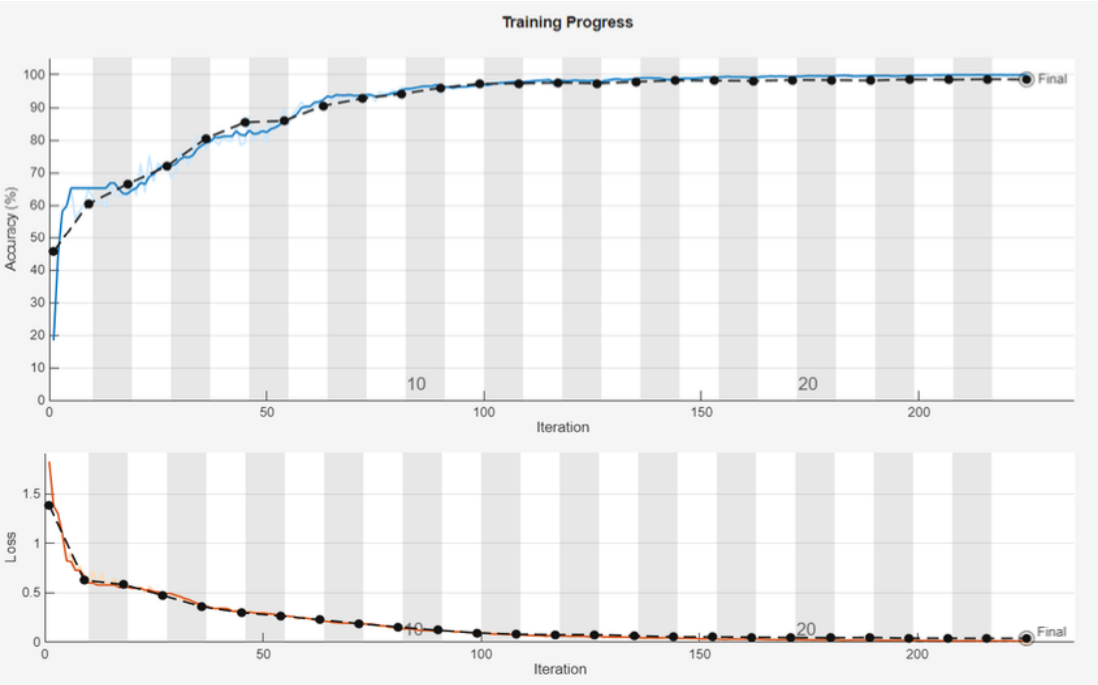

Figure 3: The graphs of decision accuracy and cross-entropy loss for the validation while training the CNN.

Key classification results include:

- BPSK: 2008 correct classifications
- QPSK: 1964 correct with 14 misclassifications
- 8PSK: 2032 correct with 13 misclassifications
- 8QAM: 1936 correct with 37 misclassifications
- 16QAM: 1977 correct with 18 misclassifications

2.7 Experimental Results and Analysis

To evaluate the effectiveness of the proposed classification approaches, we conducted extensive testing using both simulated and real-world signals. The experimental framework was designed to provide a thorough assessment of each method's performance under various conditions.

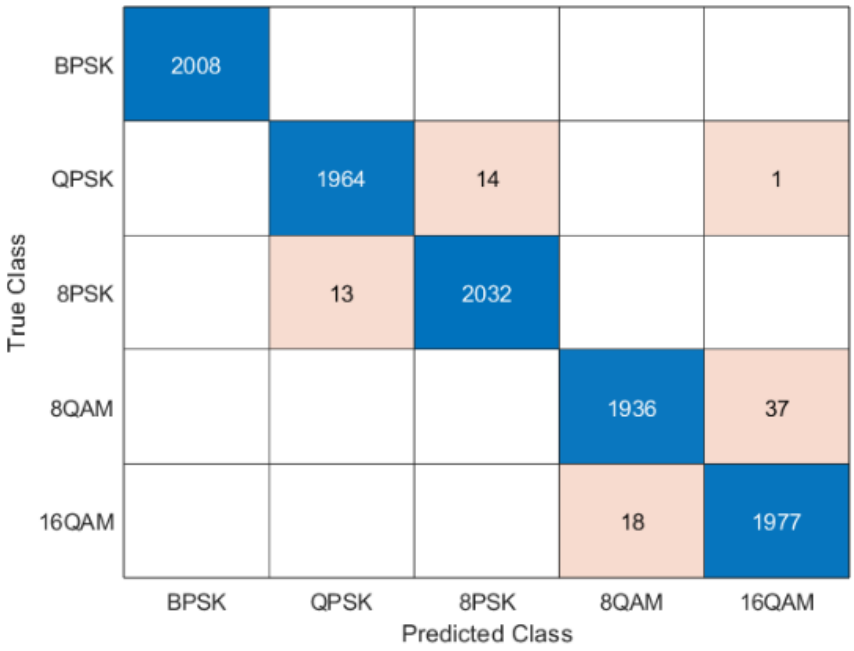

Figure 4: Confusion matrix for the CNN approach

2.7.1 Experimental Setup

The evaluation was conducted using a comprehensive dataset consisting of:

- 20,000 total signals
- 4,000 signals for each modulation type:
    - BPSK
    - QPSK
    - 8PSK
    - 8QAM
    - 16QAM

Signal generation parameters included:

- Random variations in a, φ, ω, τ parameters
- Signal-to-noise ratios (SNR) ranging from 0 to 15 dB
- Step size of 2.5 dB for SNR variations

Feature vectors were processed differently for each method:

- Multinomial regression: 54-dimensional extended feature vectors
- Nearest neighbor and GMM: Original 9-dimensional feature vectors
- CNN: Direct processing of 1024 signal samples

2.7.2 Results on Simulated Signals

Table 1 presents the decision accuracies for different classification methods for non-mixed signals.

Table 1. Programs Decision accuracies for different classification methods

	BPSK	QPSK	8PSK	8QAM	16QAM	Average
Nearest Neighbour	100%	93%	97%	96%	91%	96%
GMM	100%	97%	99%	98%	99%	98%
Multinomial regression	100%	97%	99%	98%	98%	98%
CNN	100%	99%	99%	98%	99%	99%

Key observations from Table 1:

- All methods achieved 100% accuracy for BPSK signals
- CNN demonstrated the highest overall average accuracy at 99%
- GMM and Multinomial regression showed identical average performance at 98%
- Nearest Neighbour approach achieved 96% average accuracy

2.7.3 Results on Real Signals

The methods were further validated using real signals from a CDM-625 modem:

- 10 BPSK signals
- 20 QPSK signals
- 25 8PSK signals
- 20 8QAM signals
- 14 16QAM signals

Table 2 presents the decision accuracies for different classification methods when applied to real signals.

Key findings from Table 2:

- Nearest Neighbour method achieved perfect classification with 100% accuracy across all modulation types
- Other methods (GMM, Multinomial regression, CNN) maintained consistently high performance with 99% accuracy
- Slight variations were observed in 8PSK classification for some methods

While the CNN showed slight advantages in handling simulated signals, it's worth noting that its performance on real signals matched other methods. This could be partially explained by the overfitting effect.

Table 2. Programs Decision accuracies for different classification methods (real signals)

	BPSK	QPSK	8PSK	8QAM	16QAM	Average
Nearest Neighbour	100%	100%	100%	100%	100%	100%
GMM	100%	100%	96%	100%	100%	99%
Multinomial regression	100%	100%	96%	100%	100%	99%
CNN	100%	100%	96%	100%	100%	99%

2.7.4 Comparative Analysis

The experimental results revealed several key findings:

- All methods provided robust classification performance
- CNN showed superior performance for simulated signals
- Nearest Neighbor method excelled with real signals
- The high accuracy across all methods validates the effectiveness of the chosen approaches

CHAPTER 3: SOFTWARE DEVELOPMENT TECHNICAL TASKS

The application architecture comprises two distinct components:

- **Web Service**: Developed using Elixir and the Phoenix web framework, ensuring robust, high-performance backend functionality.
- **Model Methods**: Implemented as independent modules in Python, leveraging state-of-the-art techniques, including Convolutional Neural Networks (CNNs), Gaussian Mixture Models (GMMs), Neural Networks (NNs), and Multinomial Regression (MNR).

Each component operates independently, ensuring modularity, while integration facilitates seamless interaction between the web service and the Python-based models.

3.1 Web Service (Elixir and Phoenix Framework)

Features and Responsibilities

- **API Development:**
  - RESTful API endpoints to enable interaction with Python-based model methods through erlport.
  - JSON-based communication for efficient data exchange.
- **Task Management:**
  - Asynchronous job handling for executing model-related processes.
  - Utilization of Oban for advanced job scheduling and background processing.
- **Web Interface:**
  - provide a user interface or serve static assets for front-end applications.

Technical Requirements

- **Framework: Phoenix (version 1.8 or later).**
- **Database: in-memory databased**
- **Integration with Python Models**:
  - erlport ensures seamless communication between Elixir and Python components.
- **Testing**:
  - Unit testing using ExUnit.
  - Stress testing of API endpoints to ensure scalability.

Implementation Tasks

- Configure the Phoenix framework.

- Design and develop RESTful API endpoints.
- Implement GenServer-based job scheduling for task management.
- Develop and execute comprehensive unit and integration tests.

3.2 Model Methods (Python)

Features and Responsibilitiesx

- **Advanced Machine Learning Models**:
  - **CNN**: Applied for classification tasks signal data.
  - **GMM**: Utilized for density estimation and clustering purposes.
  - **NN**: Custom or pre-trained models tailored for specific predictions.
  - **MNR**: Designed for multi-class classification problems.
- **Independent Execution**:
  - Each model operates as a standalone module.
  - Models can be invoked from the Elixir web service using erlport.
- **Training Workflow**:
  - Users upload feature files to train models.
  - Post-training, each model generates a dedicated training module for testing with new data.

3.3 Lifecycle Management

Training modules are allocated per user and are automatically removed after one hour to optimize resource usage.

3.3.1 Technical Requirements

- **Libraries and Frameworks**:
  - PyTorch, TensorFlow, or Keras for deep learning implementations.
  - Scikit-learn for statistical models like GMM and MNR.
  - Flask or FastAPI for exposing APIs.
- **Integration**:
  - API endpoints for training and testing models.
  - Seamless invocation of Python methods from Elixir using erlport.
- **Testing**:
  - Unit tests using PyTest for individual methods.

- End-to-end integration tests with the Elixir web service.

3.3.2 Implementation Tasks

- Develop and validate each model method (CNN, GMM, NN, MNR).
- Enable integration with the Elixir web service.
- Implement logging, error handling, and optimization techniques.
- Perform model compression or quantization to enhance deployment efficiency.

3.3.3 System Integration

Communication Between Components

- REST APIs and erlport enable efficient communication between the web service and model methods.
- Clearly define API contracts with detailed input/output formats and error handling protocols.

Deployment Strategy

- Utilize Docker for containerization of both components.
- Set up CI/CD pipelines (GitHub Actions) for automated testing and deployment.

Security Measures

- Enforce HTTPS for secure communications.
- Use environment variables for sensitive data like API keys and database credentials.

Conduct comprehensive security testing, including penetration tests on all endpoints.

CHAPTER 4: SOFTWARE IMPLEMENTATION

4.1 System Components

The system architecture overview is presented in Figure 5. The main components of it are:

1. Phoenix Application Layer:

- Phoenix LiveView for reactive UI
- Genserver to run Python code
- ErlPort integration for Python communication
- Model management and storage

2. Python Processing Layer:

- Independent Python modules for classification
- Feature extraction and processing
- Model training and inference
- Serialization/deserialization of models

3. Storage Layer:

- Model storage for trained classifiers
- Temporary Database for metadata

4.2 Innovative Features

Hybrid Elixir-Python Architecture is presented in Figure 6.

Key innovations:

- Dynamic Python process pool for parallel processing
- Automatic process health monitoring and recovery
- Load balancing across multiple Python instances
- Zero-downtime model updates

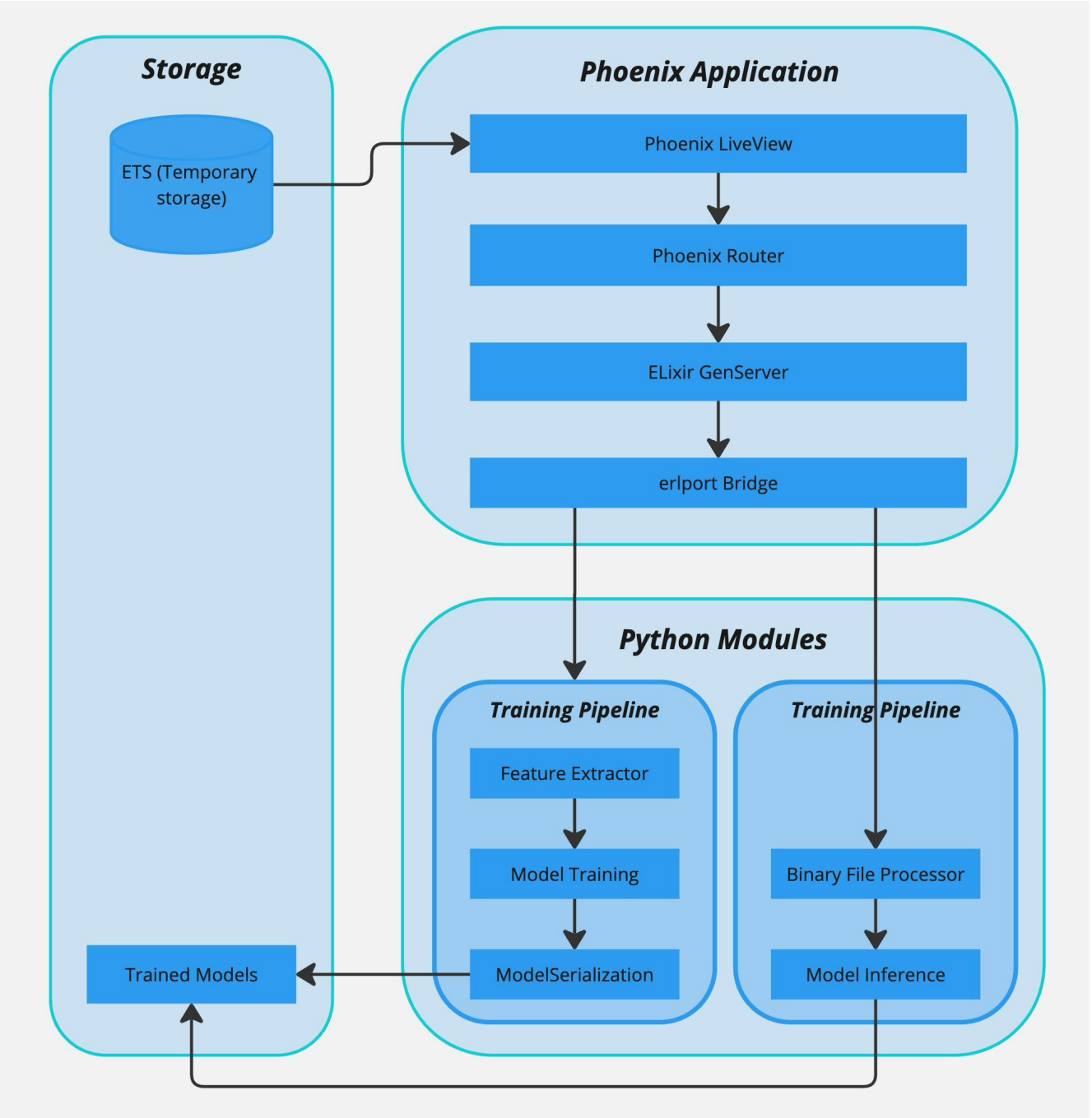

Figure 5: system architecture overview

Key innovative aspects:

1. Dynamic Processing Architecture:

- Automatic scaling of Python processes based on load
- Smart resource allocation based on signal complexity
- Real-time performance optimization
- Fault tolerance with automatic recovery

2. Advanced Model Management:

- Hot-swapping of models without service interruption
- Automatic model versioning and rollback
- Performance monitoring and automatic optimization
- Model metadata tracking and analytics

3. Intelligent Resource Allocation:

- Dynamic resource allocation based on task complexity
- Automatic load balancing across available resources
- Priority-based task scheduling
- Resource usage optimization

4. Monitoring and Analytics:

- Real-time performance tracking
- Automatic anomaly detection
- Performance trend analysis
- Resource utilization optimization

5. Fault Tolerance:

- Automatic process recovery
- Graceful degradation under heavy load
- Data consistency guarantees
- Zero-downtime updates

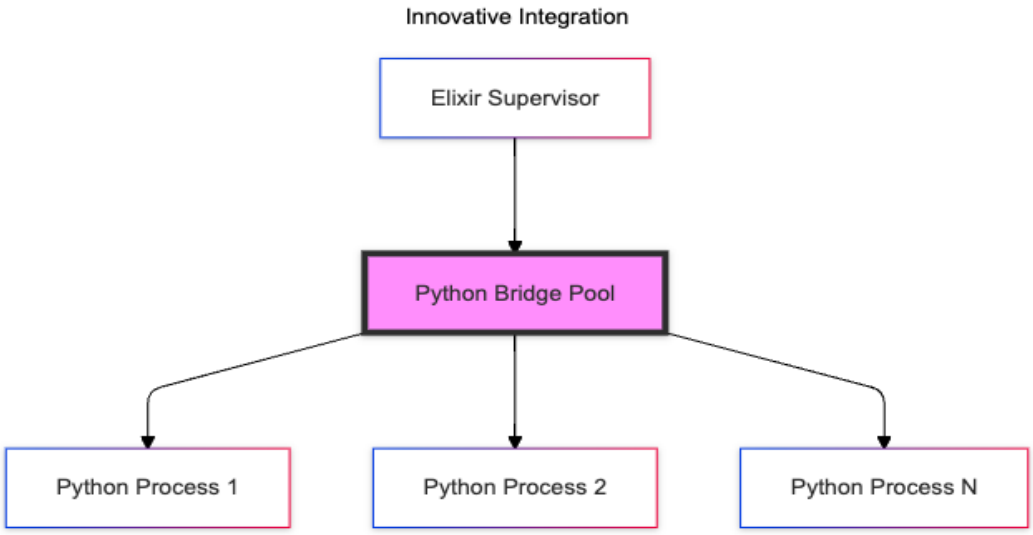

Figure 6: Integration between Elixir and Python

4.3 Technical stack rationale

4.3.1 Core Technologies

1. Elixir/Phoenix Framework

- Rationale:
    - High concurrency through BEAM VM
    - Excellent fault tolerance with supervisor trees
    - Real-time capabilities with Phoenix LiveView
    - Built-in distribution capabilities
    - Efficient handling of multiple simultaneous connections
- Benefits for the project:
    - Reliable handling of multiple signal processing requests
    - Real-time status updates during model training
    - Robust error handling for long-running processes

2. Python for Signal Processing

- Rationale:
    - Rich ecosystem of scientific computing libraries (NumPy, SciPy)
    - Extensive machine learning frameworks (TensorFlow, PyTorch)
    - Efficient numerical computations
    - Strong community support for signal processing
- Benefits for the project:
    - Efficient implementation of feature extraction
    - Access to optimized machine learning libraries
    - Easy integration of different classification algorithms

3. ErlPort for Elixir-Python Integration

- Rationale:
    - Stable bidirectional communication
    - Efficient process isolation
    - Built-in error handling
    - Maintained by the community
- Benefits for the project:
    - Safe execution of Python code
    - Resource isolation

- Easy error recovery

4.3.2 Storage Solutions

File System for Model Storage

- Rationale:
  - Direct access to model files
  - Simple versioning
  - Easy backup and restore
  - Efficient handling of large files
- Use cases:
  - Storing trained models
  - Keeping feature files
  - Managing binary signal files

4.3.3 Development Tools

1. Git for Version Control

- Rationale:
  - Distributed development
  - Branch-based workflow
  - Code review capabilities
- Benefits:
  - Team collaboration
  - Feature tracking
  - Code quality maintenance

2. Docker for Containerization

- Rationale:
  - Consistent development environment
  - Easy deployment
  - Scalable architecture
- Benefits:
  - Reproducible builds
  - Simple dependency management
  - Environment isolation

4.3.4 Alternative Technologies Considered

1. Alternative to Elixir/Phoenix:

- Node.js/Express
    - Pros: Large ecosystem, familiar to many developers
    - Cons: Less suitable for long-running processes, more complex concurrency
- Ruby on Rails
    - Pros: Rich ecosystem, convention over configuration
    - Cons: Limited concurrency, higher resource usage

2. Alternative to Python:

- Julia
    - Pros: Better performance for numerical computations
    - Cons: Smaller ecosystem, fewer machine learning libraries
- R
    - Pros: Strong statistical capabilities
    - Cons: Less suitable for production systems

3. Alternative Integration Methods:

- REST API
    - Pros: Simple, standard approach
    - Cons: Higher latency, more complex error handling
- gRPC
    - Pros: Efficient communication
    - Cons: More complex setup, less flexible

4.3.5 Justification of Final Choices

The combination of Elixir/Phoenix and Python provides:

- Robust web interface with real-time capabilities
- Efficient signal processing and machine learning
- Scalable architecture for future growth
- Good balance between development speed and performance

4.4 Deployment documentation

4.4.1 System Requirements

- Hardware Requirements:
    - CPU: 4+ cores recommended for parallel processing
    - RAM: Minimum 8GB, 16GB recommended

- Storage: 20GB+ for system and model storage
- GPU: Optional, recommended for CNN models

- Software Requirements:
  - Erlang/OTP 27 or later
  - Elixir 1.18 or later
  - Python 3.10
  - Docker 20.10+
  - Git

4.4.2 Development Environment Setup

Using bash commands

- Clone repository

```
git clone https://github.com/savik13/amc_ml_methods
cd amc_ml_methods
```

- Setup Python environment

```
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

- Setup Elixir dependencies

```
mix deps.get
mix compile
```

Docker Deployment

- Clone repository

```
git clone https://github.com/savik13/amc_ml_methods
cd amc_ml_methods
```

- Run docker-compose file

```
Docker-compose up -d
```

4.5 Software documentation

The Modulation Classification System is an Elixir/Phoenix-based web application integrated with Python modules for signal processing and classification. The system allows users to:

- Upload training data for different modulation types
- Train classification models

- Upload and classify new signals using trained models

Key Components:

1. Web Interface (Phoenix)
2. Signal Processing Engine (Python)
3. Model Management System
4. Storage Management

4.6 User guide

This application consists of two main parts:

1. Model Training
2. Signal Classification Testing

4.6.1 Model Training

Prerequisites

- Prepare your feature files in .mat format for each of the following modulation types:
  - BPSK
  - QPSK
  - 8PSK
  - 8QAM
  - 16QAM

Training Process

1. In the "Model training" section, locate the file upload buttons for each modulation type
2. Click "Choose file" for each modulation type and select the corresponding .mat feature file
3. Once all files are uploaded, click the blue "Train" button
4. The training process will run asynchronously - please wait for completion
5. Results will automatically populate in the results table below, showing:
   - Method name
   - Accuracy scores for each modulation type (BPSK, QPSK, 8PSK, 8QAM, 16QAM)
   - Average accuracy across all modulations

Understanding the Results Table

1. The table displays accuracy percentages for different classification methods:
   - Nearest Neighbour
   - GMM (Gaussian Mixture Model)

- Multinomial regression
- CNN (Convolutional Neural Network)

4.6.2 Signal Classification Testing

Testing Process

1. In the "Signal classification" section, locate:
    - "Path to bin directory" - Click "Choose file" to select your binary file containing real data
    - "Classifier method" - Use the dropdown menu to select your preferred classification method (currently showing MNR)
2. Click the blue "Classifier" button to start the analysis
3. Wait for the results to be displayed

Best Practices

- Ensure all your feature files are in the correct .mat format before uploading
- Wait for the training process to complete before starting classification
- Keep track of which models perform best for your specific use case
- Note that different classification methods may perform differently depending on your data

Troubleshooting

- If a file is not selected, you'll see "No file selected”
- If the training process seems stuck, check your file formats
- Ensure all required files are uploaded before starting the training process

Technical Notes

- The application supports multiple classification methods including Nearest Neighbour, GMM, Multinomial regression, and CNN
- Results are displayed with high precision (up to 15 decimal places for some measurements)
- The interface supports both training and testing phases of model development

CONCLUSIONS

The results of this study validate the effectiveness of multiple machine learning approaches in Automatic Modulation Classification (AMC). Among the methods tested, Convolutional Neural Networks (CNNs) demonstrated superior performance, achieving 99% classification accuracy on both simulated and real signals. Gaussian Mixture Models (GMMs) and multinomial regression methods also yielded strong results, with accuracies of 98%, confirming their viability for high-performance classification tasks.

The proposed software architecture, integrating these classification methods into a unified system through a web service, offers a practical solution for real-world implementation. The accuracy demonstrated in both controlled and real-world testing highlights the robustness and potential of this approach for use in various communication environments.

While the study achieved strong results, it is important to note some limitations. For instance, the classification accuracy may vary in more complex or noisy real-world conditions, suggesting that future work should focus on further testing under diverse environmental conditions. Additionally, exploring the scalability of the system to handle larger datasets and more modulation schemes is a key area for future improvement.

In terms of practical applications, the findings of this study have significant implications for non-cooperative communication systems, software-defined radio, and cognitive radio technologies. The ability to accurately classify modulation schemes under varying conditions could greatly enhance the efficiency of spectrum usage and improve the adaptability of communication systems in dynamic environments. Future research should investigate more advanced machine learning techniques, such as deep reinforcement learning, and explore ways to optimize the system for real-time processing in commercial and military applications.

REFERENCES

[1] C. Agne , B. Cornell, M. Dale, R. Keams, F. Lee, Shared-spectrum band-width efficient satellite communications. – Proc. IEEE MILCOM, 2010. – P.341-346.

[2] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," IEEE Journal of Selected Topics in Signal Processing, vol. 12, 2018, pp. 168–179.

[3] A. Abdelmutalab, Kh. Assaleh, and M. El-Tarhuni, "Automatic modulation classification based on high order cumulants and hierarchical polynomial classifiers," Physical communication, vol. 21, 2016, pp. 10–18.

[4] V. Semenov, "Method for the automatic modulation classification based on linear regression and feature selection," Physico-mathematical modelling and informational technologies, Vol. 36, 2023, pp. 22-26.

[5] V. Semenov, "Method for the Automatic Modulation Classification for the Sums of Phase-Shift-Keying Signals Based on Polynomial Features", Proceedings of 2024 IEEE ELNANO conference, pp. 587-590.

[6] V.K. Zadiraka and V. Semenov, "Methods for the solution of systems of nonlinear algebraic equations and functions' minimization tasks: elements of theory and applications", Naukova Dumka, Kyiv, 2023.

[7] T. O'Shea, "An Introduction to Deep Learning for the Physical Layer,"IEEE Transactions on Cognitive Communications and Networking, Vol. 3, pp. 563-575.

[8] T. O'Shea, J. Corgan, and T. Clancy, "Convolutional Radio Modulation Recognition Networks," Engineering Applications of Neural Networks, vol. 629, pp. 213–226, 2016.

[9] R. Amin, F. Bellili, S. Affes, and A. Ghrayeb, "On the Performance of Automatic Modulation Classification in Flat-Fading Channels: A Machine Learning Approach," IEEE Transactions on Communications, vol. 67, no. 10, pp. 6487–6500, 2019.

[10] X. Liu, D. Yang, and A. El Gamal, "Deep Neural Network Architectures for Modulation Classification," arXiv preprint arXiv:1712.00443, 2018.

[11] C. A. Harper, M. A. Thornton, and E. C. Larson, "Automatic Modulation Classification with Deep Neural Networks," arXiv preprint arXiv:2301.11773, 2023.