MKI T3 – Enhancing Intrusion Detection System using Machine Learning

- Patil, Raj Vijaykumar
- Degirmen Kagan
- Hussein, Mahmoud Fathy Muhammad
- Loliyekar, Viraj

Introduction

Intrusion Detection Systems (IDS) are critical Components that provide real time monitering and analysis of network traffic.

What is IDS
Sophisticated security tools that monitor traffic flows and detect suspicious or potentially malicious behavior across network infrastructure

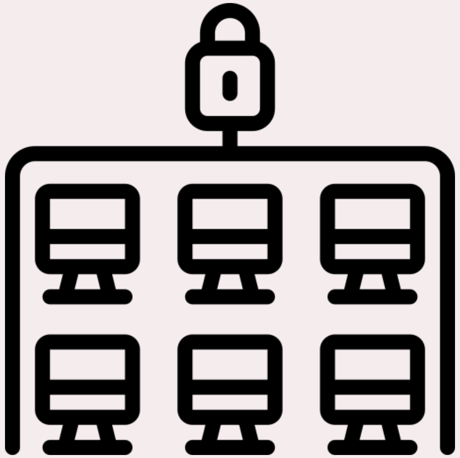
Types of IDS

- Host-Based IDS (HIDS): Monitor Individual Endpoints
- Network Based IDS: Monitors Entire Network

Beyond Firelwalls
Traditional firewalls only filter traffic based on predefined rules, whereas IDS can identify complex attack patterns that bypass perimeter defences

Host IDS

Network IDS



Problems

Traditional IDS works based on Signature (Attack Pattern), Policies, and Rules which are manually set.

Weakness	Consequence
Signature-based only	Fails to detect unknown (zero-day) attacks
Static thresholds	Mislabels unusual but safe behavior
Manual rules	Slow updates, hard to maintain
High false positives	Too many false alarms

Solution

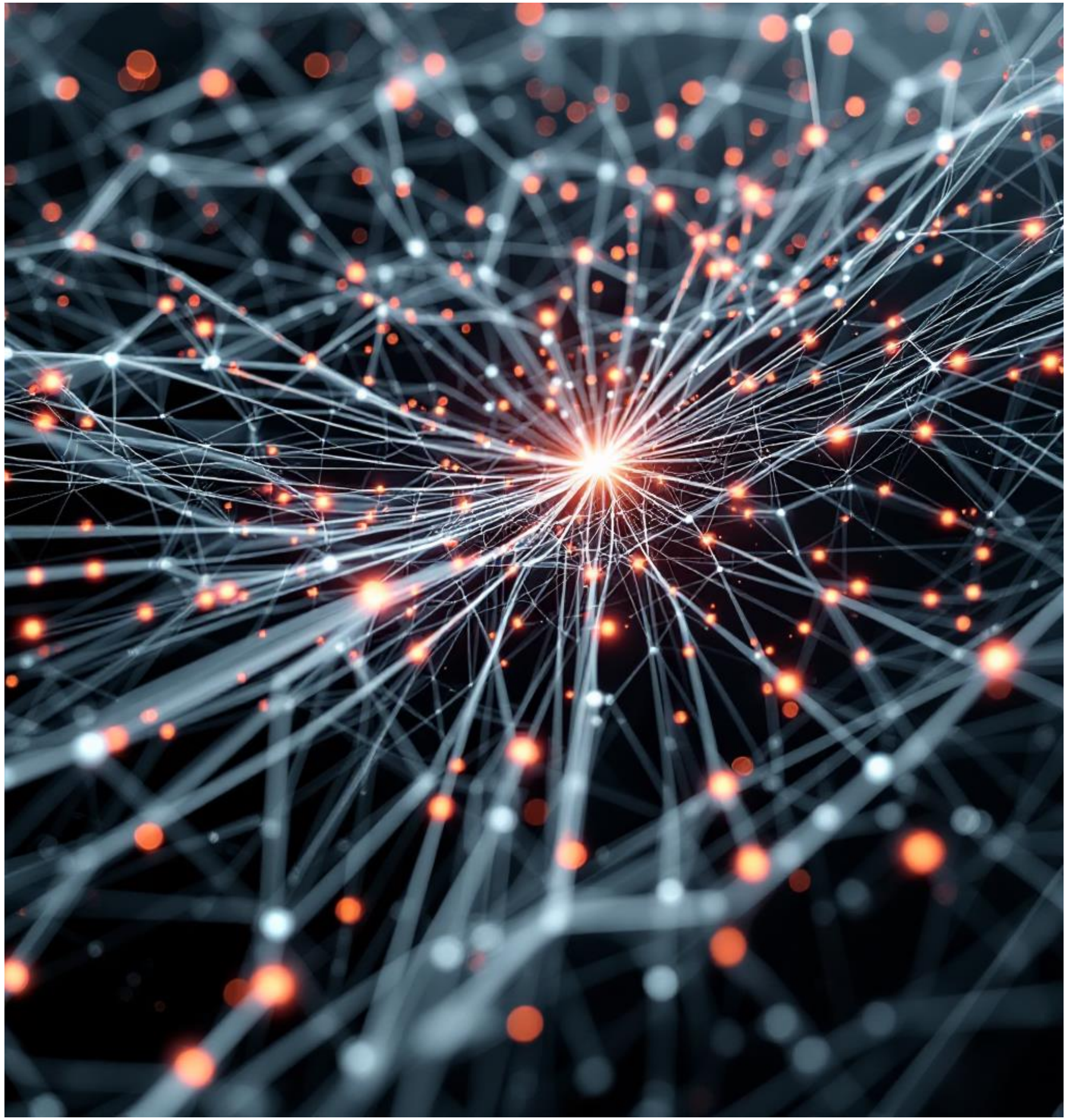
Smarter Detection with Machine learning

1. Proactive Threat Detection
Identifies new and unknown threats based on behavioural anomalies rather than predefined signatures

2. Self-Learning Systems
Continuously learns from past data, eliminating the need to manually hard-code detection rules

3. Improved Accuracy
Significantly reduces false positives over time through pattern recognition and contextual analysis

4. Real-Time Analysis
Enables instantaneous threat assessment and automated response capabilities



Data and Preprocessing

Data Selection
Used NSL-KDD 10% subset containing realistic network traffic patterns and attack vectors

Data Cleaning
Removed missing values and irrelevant features to ensure data quality and model efficiency

Feature Engineering
Encoded categorical text features like protocol and service into numerical representations

Normalization
Applied MinMaxScaler to standardize numeric features for optimal algorithm performance

Data and Preprocessing

1. Raw File

2. Assign Columns

3. Map Attacks to 4 Categories

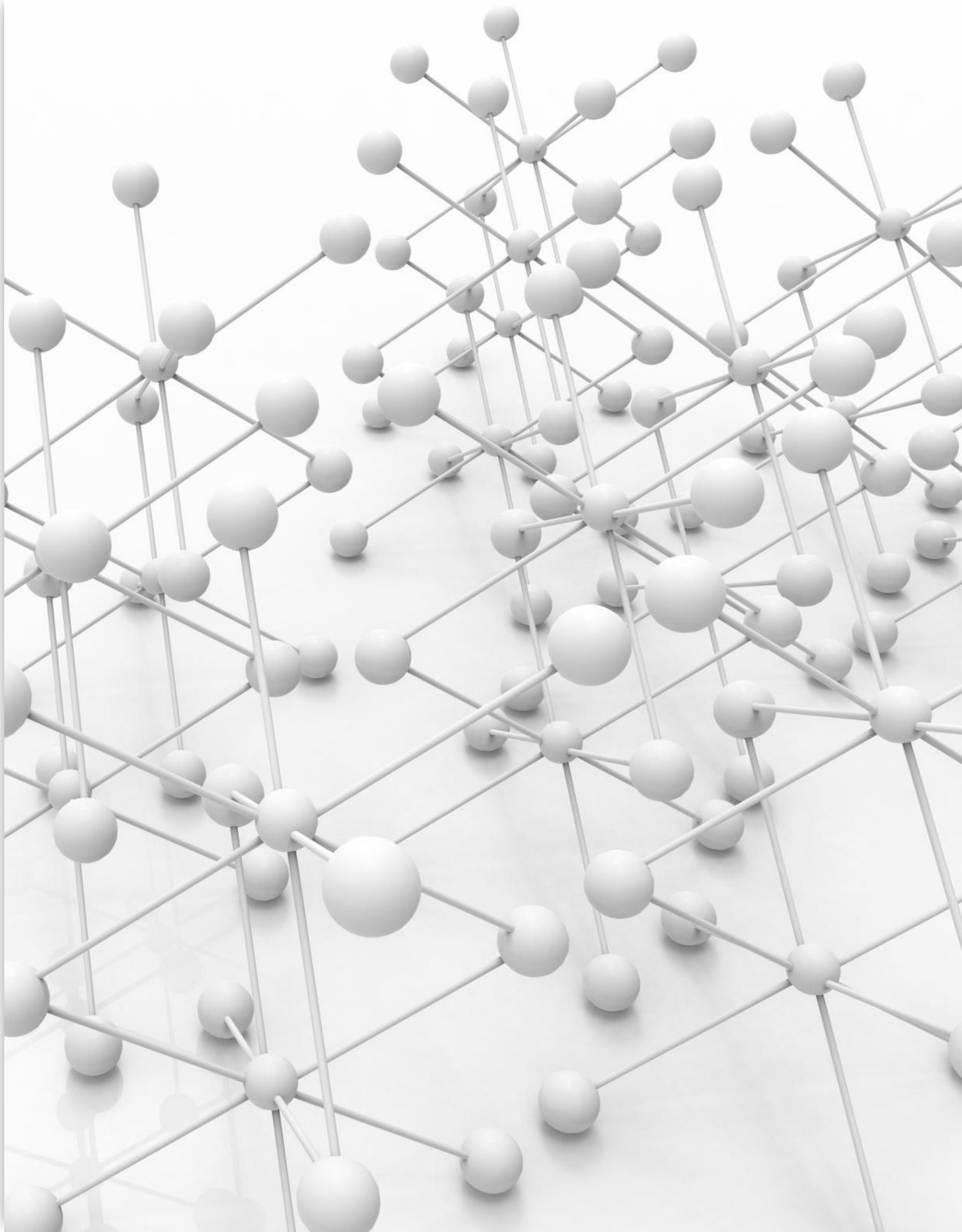
4. Encode Protocol/Service/Flag

5. Scale Numerical Values(0-1)

6. Separate Features (X) and Labels (Y)

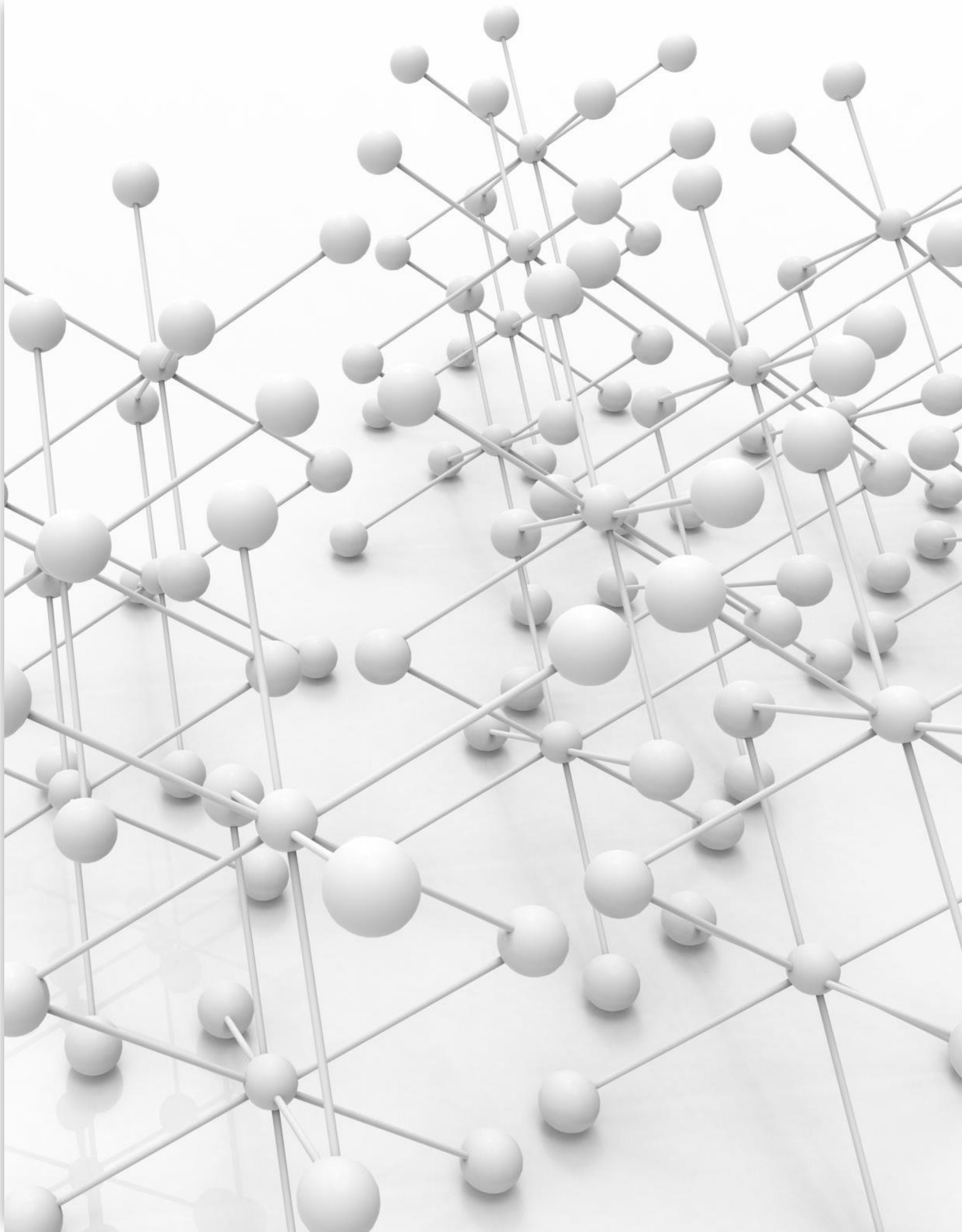
Shallow Neural Network

- Neural network with **only one hidden layer**.
- **Architecture:**
    - **Input Layer:** Raw data (e.g., network features from your .ipynb file)
    - **Single Hidden Layer:** Performs core computations.
    - **Output Layer:** Final classification (e.g., 'Normal' vs. 'Attack Type')
- **Characteristics:**
    - Simpler, easier to interpret.
    - Faster to train, less computational power needed.
    - Limited capacity: Best for less complex problems.



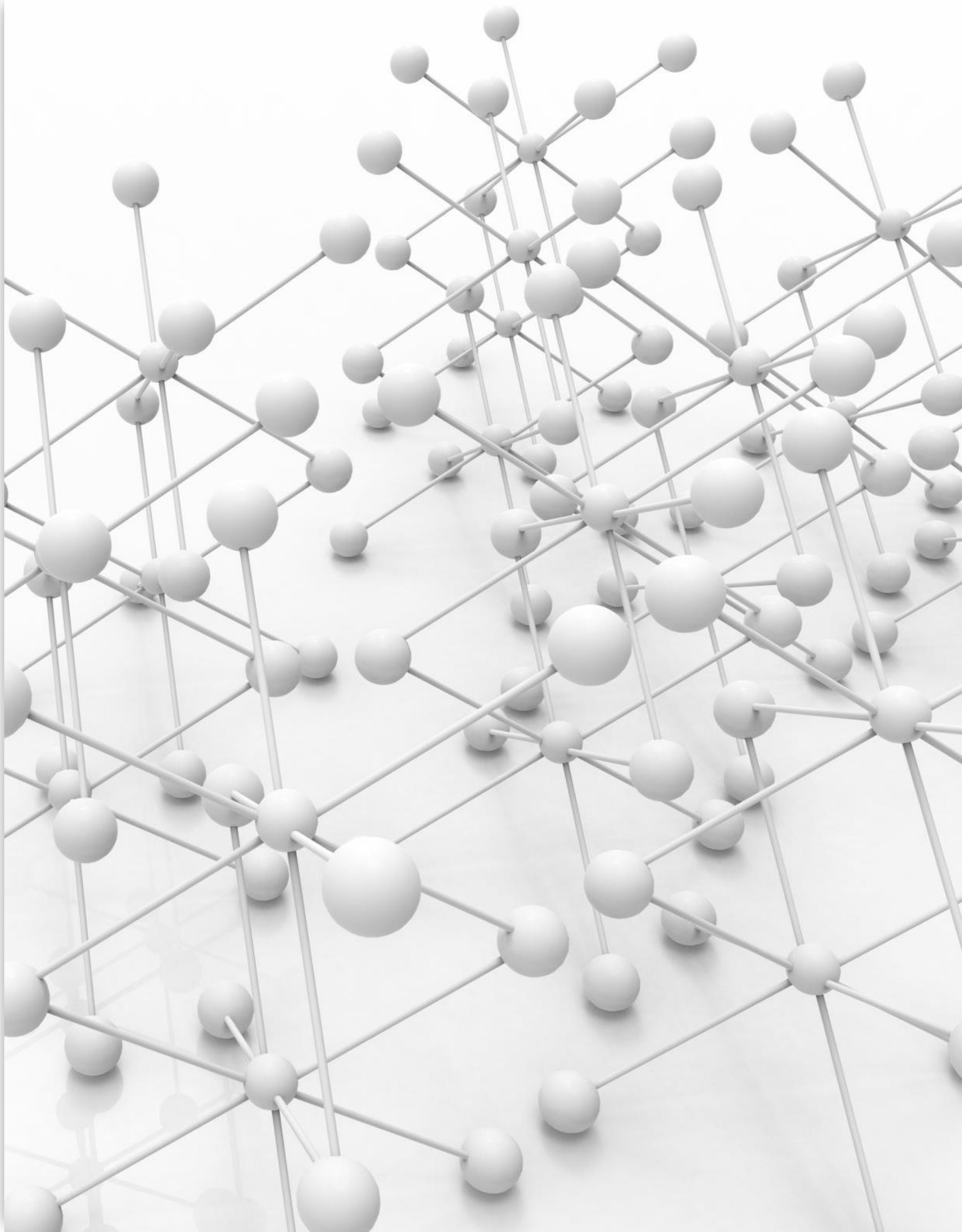
Shallow Neural Network

- **Building & Training (Shallow):**
  - Define basic layer structure (neurons, activation functions).
  - Prepare data (clean, scale, split).
  - Train efficiently: Adjust weights to minimize error.
- **Intrusion Detection Relevance:**
- Shallow networks could be employed for **simpler intrusion detection tasks** where the attack patterns are distinct and can be identified based on a few prominent features.
- For instance, detecting very obvious port scans or simple denial-of-service (DoS) attacks based on basic traffic statistics. However, for more sophisticated or novel attacks, their ability to learn complex representations might be limited.



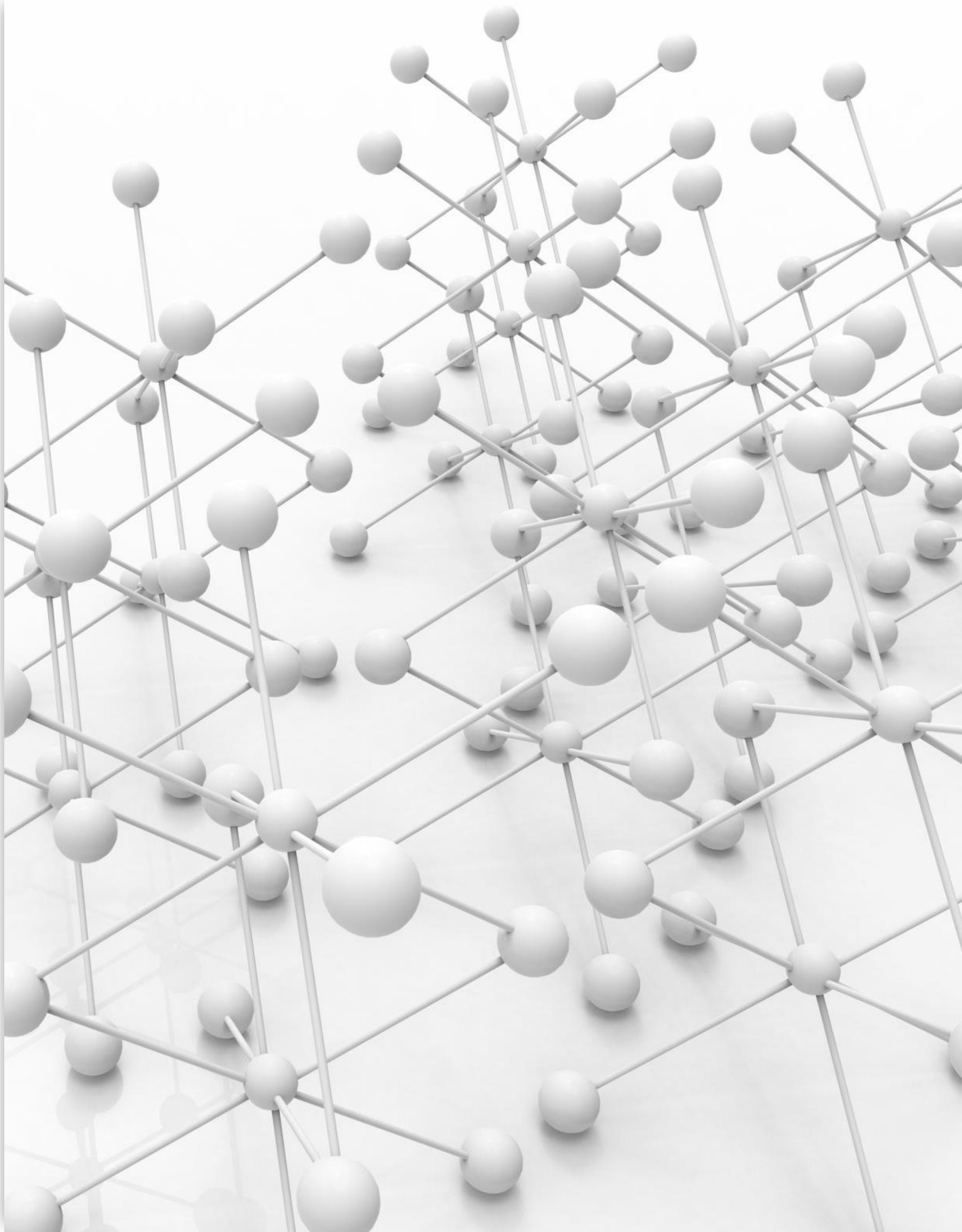
Deep Neural Network

- Neural network with **multiple hidden layers**.
- **Architecture:**
  - **Input Layer:** Raw data.
  - **Multiple Hidden Layers:** Learn hierarchical, increasingly abstract features from data.
  - **Output Layer:** Final classification (e.g., 'DoS', 'R2L', 'U2R', 'Probe').
- **Characteristics:**
  - High capacity: Learns intricate patterns from high-dimensional data.
  - Automates feature learning: Reduces manual engineering.
  - Computationally intensive: Requires more data & powerful hardware (GPUs).
  - More complex: Can be challenging to design & interpret.



Deep Neural Network

- **Building & Training (Deep):**
  - Design complex multi-layer architecture.
  - Extensive data preprocessing is crucial.
  - Demanding training: Iterative process (forward/backward propagation, optimization over many epochs) to refine weights.
- **Intrusion Detection Relevance: Ideal for complex, sophisticated attacks** (like those in your KDD Cup dataset). Learns subtle anomalies, handles vast features, and generalizes well to evolving threats.
  - **Identify Subtle Anomalies:** Learn to recognize complex, multi-stage attack patterns that might be too subtle for shallow networks to detect.
  - **Handle High-Dimensional Data:** Process the many features present in network traffic data and learn relevant hierarchical representations for accurate classification.
  - **Generalize Better:** With sufficient data, deep networks can generalize well to new, unseen attack variations, which is crucial in the ever-evolving landscape of cyber threats.

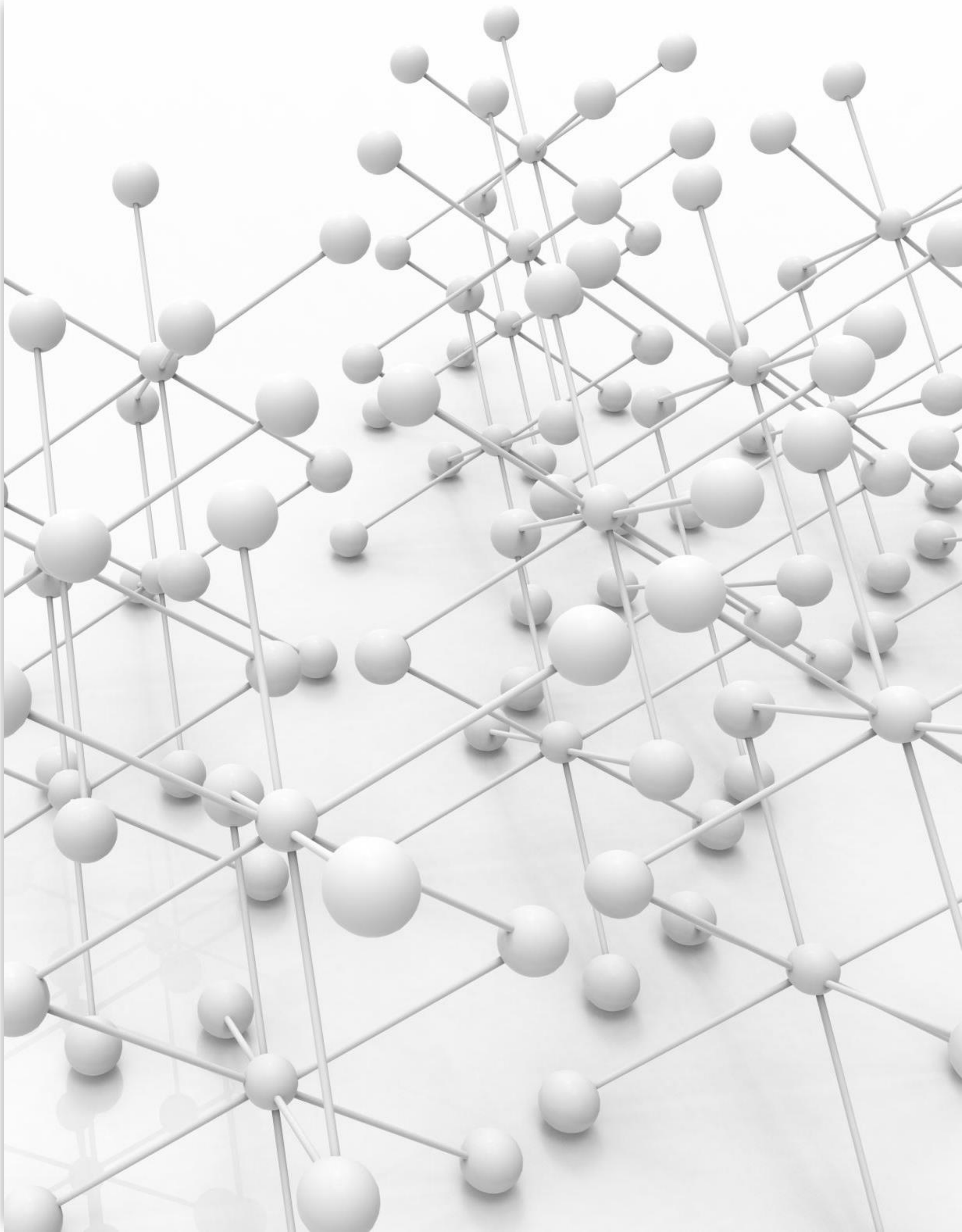


Convolutional NN

Problem Context
Classify “Attack Type” from 30-dimensional feature vectors
Input reshaped to (30, 1) for 1D convolutions

Data Pipeline

- Drop unused columns, split out target
- Min–Max scale features to [0, 1]
- 67/33 train/test split

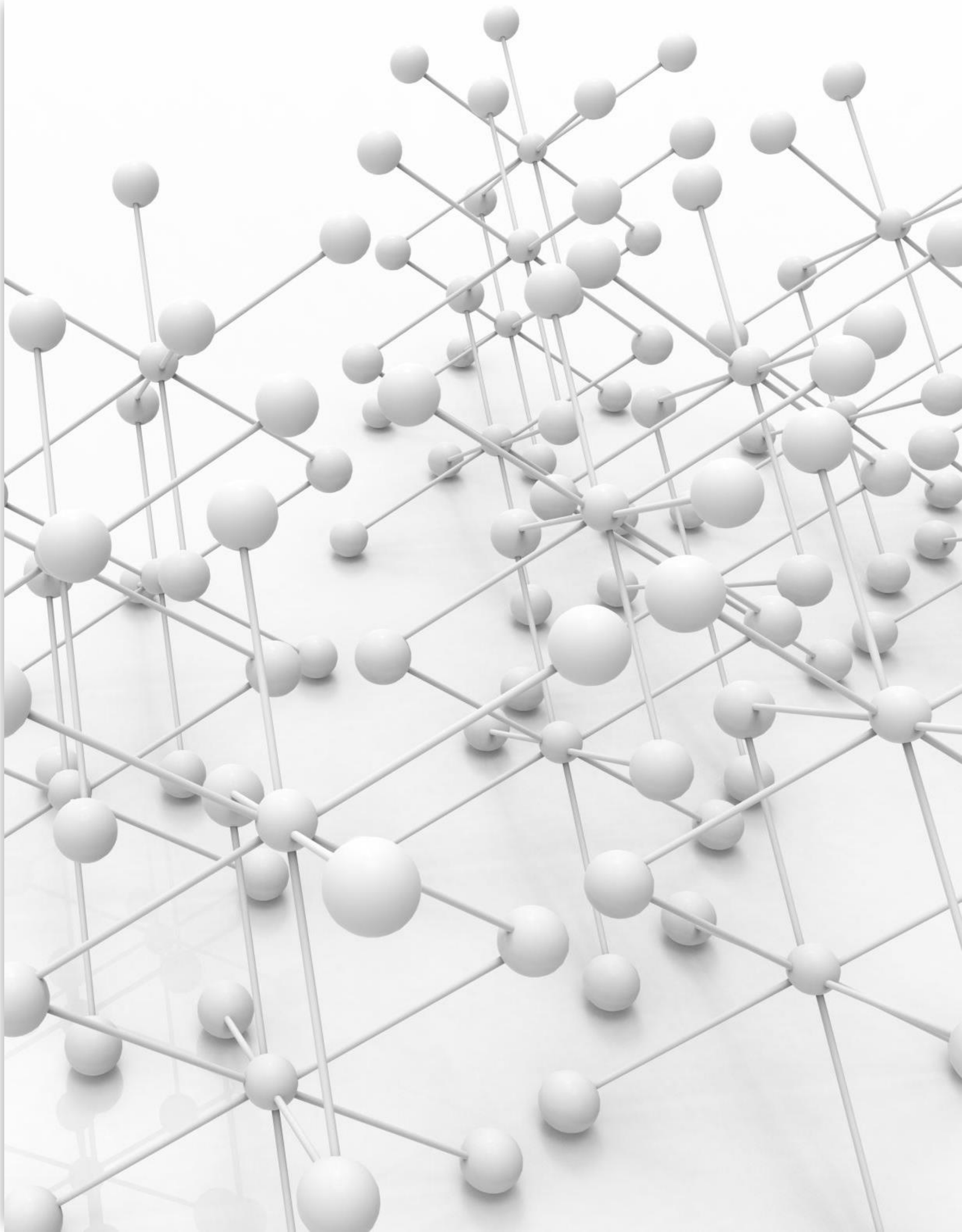


"Straightforward" 1D CNN

Architecture Highlights

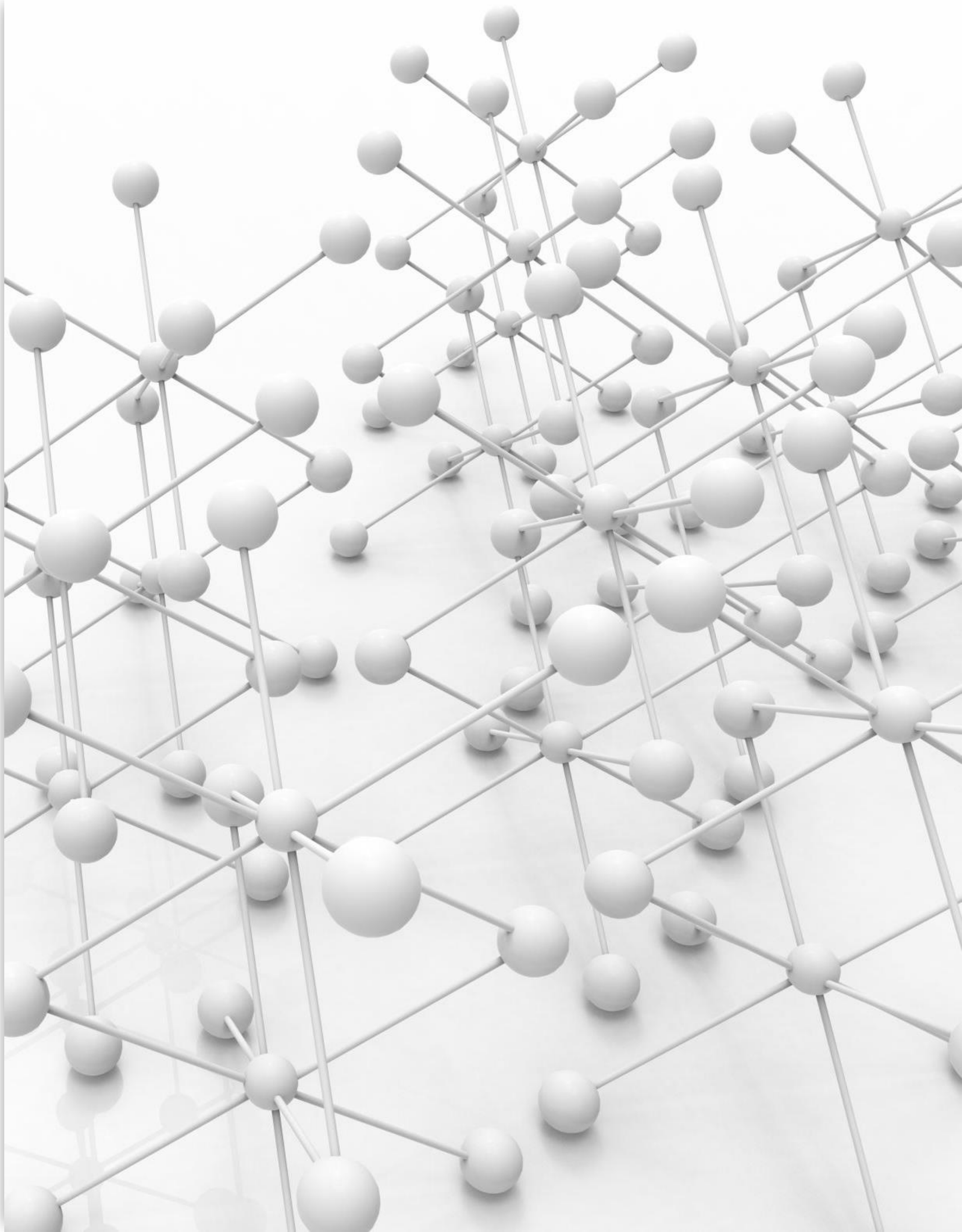
- Conv1D (24 filters, kernel=3) → ReLU
- Dropout(0.5)
- Conv1D (24 filters, kernel=3) → ReLU
- Dropout(0.5)
- Conv1D (48 filters, kernel=3) → ReLU
- MaxPool1D(pool=2) → Flatten → Dense(256) → Dropout → Dense(5, softmax)

- **Characteristics**
  – Sequential, deepening feature extraction
  – Moderate parameter count (~100 K)
- **Performance**
  – Near-perfect train & test accuracy in only 2 epochs
  – No underfitting due to dropout



Hybrid (CNN + Dense)

- **Architecture Highlights**
  - **Branch 1:** Conv1Ds
  - **Branch 2:** Conv1D → Dense
  - **Branch 3:** Conv1Ds → Dense
  - **Merge:** Concatenate([Branch1, Branch2, Branch3])
- **Why Hybrid?**
  – Captures features at multiple filter sizes and depths
  – Ensembles shallow vs. deep representations
- **Performance**
  – Also achieves perfect accuracy
  – Slightly larger model, more expressive



Random Forest

Random Forest is an ensemble learning algorithm used for classification and regression. It works by building multiple **decision trees** and combining their outputs using **majority voting** (for classification) or averaging (for regression).

Key Concepts:

- **Ensemble Model**: Uses multiple decision trees instead of one.
- **Bagging (Bootstrap Aggregation)**: Each tree is trained on a different random subset of the data.
- **Random Feature Selection**: At each split, only a random subset of features is considered.
- Helps reduce **overfitting** and improves **generalization**.

Why Use Random Forest in Our Project?
In our Intrusion Detection System (IDS) project using neural networks (Shallow NN, Deep NN, CNN), we added Random Forest to:

- **Compare performance** of classical ML with deep learning.
- **Evaluate speed vs. accuracy** tradeoff.
- **Show a simpler model** that requires less training time and tuning.

How Random Forest Works ?

- Builds multiple decision trees on random subsets of data.
- Each tree gets random rows (samples) and columns (features).
- Sampling is done with replacement (bootstrapping).
- Each tree learns differently, reducing overfitting.
- For classification, it uses majority voting.
- Aggregation of diverse trees leads to better accuracy and robustness.

Classical ML Model: Random Forest Class

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import time

# start the training
start = time.time()

# training the model
rf = RandomForestClassifier(n_estimators=300, random_state=42)
rf.fit(X_train, Y_train.values.ravel())

# End time of training
end = time.time()
print(f" Training completed in {end - start:.2f} seconds.")

# Predict and print accuracy
y_pred_rf = rf.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred_rf)
print(f" Random Forest Accuracy: {accuracy:.4f}")
```

```
Training completed in 194.42 seconds.
Random Forest Accuracy: 0.9998
```

Random Forest