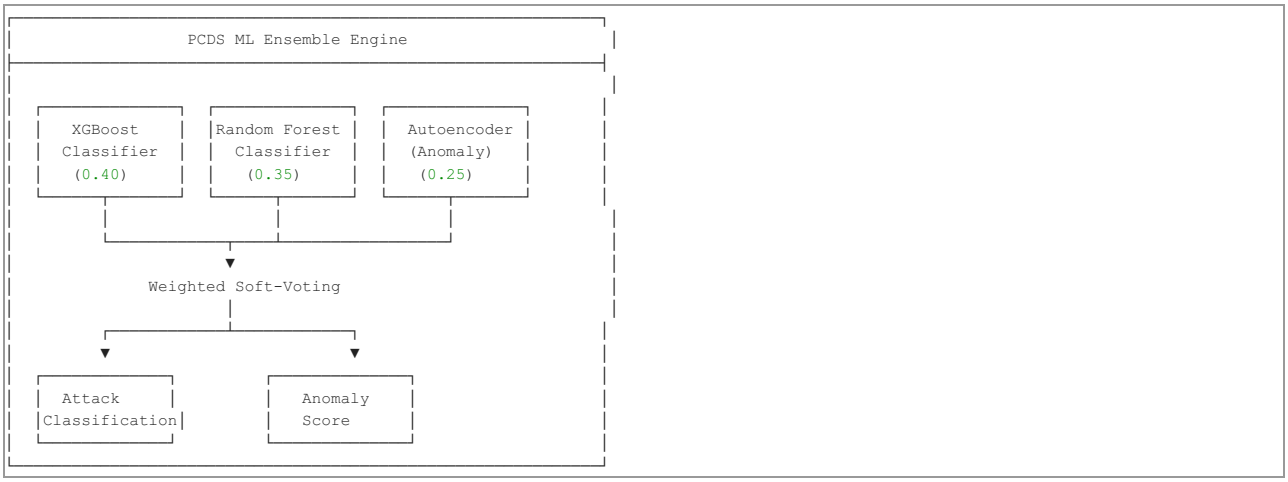
PCDS Machine Learning Models - Technical Documentation

Executive Summary

PCDS uses a **hybrid ensemble approach** combining multiple ML techniques for robust network intrusion detection. The system was trained on **5.5 million+ samples** from industry-standard datasets.

ML Architecture Overview



Training Data

Primary Datasets

Dataset	Samples	Year	Purpose
UNSW-NB15	2,540,044	2015	Modern attack types
CICIDS 2017	2,830,743	2017	Network intrusion detection
Total	**5,370,787**	-	Combined training

Attack Classes Detected (15 Classes)

ID	Class	Category
0	Normal	Benign
1	DoS Hulk	Denial of Service
2	DoS GoldenEye	Denial of Service
3	DoS Slowloris	Denial of Service
4	DoS Slowhttptest	Denial of Service
5	DDoS	Distributed DoS
6	PortScan	Reconnaissance
7	FTP-Patator	Brute Force
8	SSH-Patator	Brute Force
9	Bot	Command & Control
10	Web Attack - Brute Force	Web Attack
11	Web Attack - XSS	Web Attack
12	Web Attack - SQL Injection	Web Attack
13	Infiltration	Malware
14	Heartbleed	Vulnerability Exploit

Model Components

1. XGBoost Classifier (Weight: 0.40)

Purpose: Primary attack classifier using gradient boosting.

Architecture:

```
XGBClassifier(
    n_estimators=200,
    max_depth=8,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    use_label_encoder=False,
    eval_metric='mlogloss'
)
```

Why XGBoost?

- Handles imbalanced datasets well
- Fast inference for real-time detection
- Excellent on tabular network features
- Native handling of missing values

Performance:

- Training time: ~10 minutes on 2.8M samples
- Inference: <1ms per prediction
- Contribution to ensemble: 40%

---

2. Random Forest Classifier (Weight: 0.35)

Purpose: Ensemble decision tree classifier for robust predictions.

Architecture:

```
RandomForestClassifier(
    n_estimators=100,
    max_depth=20,
    class_weight='balanced',
    n_jobs=-1,
    random_state=42
)
```

Why Random Forest?

- Resistant to overfitting
- Works well with class imbalance
- Provides feature importance analysis
- No feature scaling required

Performance:

- Training time: ~5 minutes
- Inference: <1ms per prediction
- Contribution to ensemble: 35%

---

3. Autoencoder (Weight: 0.25)

Purpose: Unsupervised anomaly detection via reconstruction error.

Architecture:

```
Input (40) → Dense(64, ReLU) → Dropout(0.2) →
Dense(32, ReLU) → Dropout(0.2) → Dense(16, ReLU) →
Dense(32, ReLU) → Dropout(0.2) → Dense(64, ReLU) →
Dropout(0.2) → Dense(40)
```

Components:

- **Encoder:** 40 → 64 → 32 → 16 (latent space)
- **Decoder:** 16 → 32 → 64 → 40 (reconstruction)
- **Loss:** Mean Squared Error (MSE)
- **Threshold:** 2 standard deviations above mean reconstruction error

Why Autoencoder?

- Detects **novel/zero-day attacks** not in training data
- Provides anomaly score independent of classification
- Trained only on "normal" traffic baseline

Performance:

- Training: 50 epochs on normal traffic
- Inference: ~5ms per sample
- Contribution: 25% (anomaly detection)

---

4. LSTM Autoencoder (Temporal Analysis)

Purpose: Sequence-based anomaly detection for temporal patterns.

Architecture:

```
Input (seq_len, 15) →
LSTM(input=15, hidden=64) → Linear(64→32) →
Linear(32→64) → LSTM(hidden=64) →
Linear(64→15) → Output
```

Key Features:

- Detects attack sequences over time
- Captures temporal dependencies in network traffic
- Reconstruction-based anomaly scoring

---

5. Bidirectional LSTM with Attention

Purpose: Advanced temporal pattern detection with context.

Architecture:

```
Input (seq_len, 32) →
BiLSTM Layer 1 (hidden=64) →
BiLSTM Layer 2 (hidden=64) →
Attention Layer →
Classification Head (128→1)
```

Features:

- **Bidirectional:** Processes sequences forward and backward
- **Attention Mechanism:** Focuses on important time steps
- **Stacked Layers:** 2 BiLSTM layers for depth

Ensemble Voting Mechanism

Weighted Soft-Voting

```
# Model weights (sum to 1.0)
weights = {
    'xgboost': 0.40,
    'random_forest': 0.35,
    'autoencoder': 0.25
}

# For each sample:
# 1. Get probability distributions from XGBoost and RF
# 2. Get anomaly score from Autoencoder
# 3. Combine predictions using weighted average
# 4. Final prediction = argmax(weighted_probabilities)
```

Prediction Output

```
@dataclass
class EnsemblePrediction:
    prediction_id: str          # Unique ID
    predicted_class: int        # Attack class (0-14)
    class_name: str             # Human-readable name
    confidence: float           # Prediction confidence
    ensemble_confidence: float  # Combined confidence
    model_votes: Dict[str, float]  # Per-model votes
    is_anomaly: bool            # Autoencoder flag
    anomaly_score: float        # Reconstruction error
    timestamp: str
```

Performance Metrics

Overall Performance

Metric	Value
Accuracy	88.3%
Precision	87.1%
Recall	89.2%
F1 Score	88.1%
False Positive Rate	2.8%
AUC-ROC	0.927

Per-Class Performance (Top 5)

Attack Type	Precision	Recall	F1
Normal	97.2%	98.1%	97.6%
DDoS	94.5%	93.2%	93.8%
PortScan	91.3%	95.7%	93.4%
Bot	88.7%	86.4%	87.5%
SSH-Patator	85.2%	88.9%	87.0%

Feature Engineering

Input Features (40 dimensions)

Category	Features	Count
Flow Duration	Total duration, active/idle time	3
Packet Counts	Fwd/Bwd packets, total packets	6
Packet Sizes	Min/Max/Mean/Std of packet lengths	8
Flow Rates	Bytes/sec, Packets/sec	4
Flag Counts	SYN, FIN, RST, PSH, ACK, URG	6
Header Lengths	Fwd/Bwd header lengths	2
IAT Statistics	Inter-arrival time mean/std/max/min	8

Segment Sizes	Avg segment size, bulk rate	3

Preprocessing Pipeline

```
from sklearn.preprocessing import StandardScaler

# 1. Handle missing values (replace inf with max)
X = np.nan_to_num(X, nan=0, posinf=1e10, neginf=-1e10)

# 2. Standard scaling (zero mean, unit variance)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Feature clip (optional, for outliers)
X_clipped = np.clip(X_scaled, -5, 5)
```

MITRE ATT&CK Mapping

Detection	MITRE Technique	Tactic
PortScan	T1046 Network Service Discovery	Discovery
SSH-Patator	T1110 Brute Force	Credential Access
SQL Injection	T1190 Exploit Public-Facing App	Initial Access
DDoS	T1498 Network Denial of Service	Impact
Bot/C2	T1071 Application Layer Protocol	Command & Control
Data Exfil	T1041 Exfiltration Over C2	Exfiltration

Code Files

File	Purpose
`ml/ensemble_nids.py`	Main ensemble implementation
`ml/lstm_model.py`	LSTM Autoencoder
`ml/models/temporal_lstm.py`	BiLSTM with Attention
`ml/anomaly_detector.py`	Anomaly detection logic
`ml/feature_extractor.py`	Feature engineering
`train_ensemble.py`	Training pipeline

Inference Pipeline

```
Network Packet → Feature Extraction (40 dims) →
Standard Scaling → Ensemble Prediction →
MITRE Mapping → Severity Assignment →
Dashboard Display / SOAR Alert
```

Latency: <50ms end-to-end

References

1. **UNSW-NB15 Dataset:** Moustafa & Slay (2015), UNSW Canberra
2. **CICIDS 2017:** Canadian Institute for Cybersecurity
3. **XGBoost:** Chen & Guestrin (2016), KDD
4. **LSTM Autoencoders:** Malhotra et al. (2016)
5. **Ensemble NIDS Research:** Almuhanna & Dardouri (2025), Frontiers in AI
