Complete AI Models Documentation for Software Engineers

Table of Contents

CNN Wi-Fi Vulnerability Detection Model

Model Overview

- **Model Type**: Convolutional Neural Network (CNN)
- **Primary Purpose**: Wi-Fi vulnerability detection and classification
- **Model Format**: `.h5` (Keras/TensorFlow)
- **Model File**: `wifi_vulnerability_cnn_final.h5`

Technical Specifications

- **Total Parameters**: ~2.3M parameters
- **Model Size**: ~9.2 MB
- **Training Time**: 30-45 minutes (GPU)
- **Inference Time**: <10ms per sample
- **Target Accuracy**: 94-97%
- **Confidence Threshold**: 0.85

Input Specifications

- **Input Shape**: `(32,)` - 32 network features
- **Reshaped for CNN**: `(32, 1)` for 1D convolution
- **Data Type**: `float32`
- **Feature Count**: 32 features total

Input Features Breakdown (32 Total)

Signal Strength Metrics (Index 0-7)

Index	Feature	Range	Unit	Description

0	RSSI	(-90, -20)	dBm	Received Signal Strength Indicator
1	SNR	(0, 40)	dB	Signal-to-Noise Ratio
2	Signal Quality	(0, 100)	%	Overall signal quality percentage
3	Noise Floor	(-100, -80)	dBm	Background noise level
4	Channel Utilization	(0, 100)	%	Channel usage percentage
5	Interference Level	(0, 100)	%	Interference detection level
6	Link Quality	(0, 100)	%	Connection quality metric
7	Signal Stability	(0, 100)	%	Signal consistency over time

Packet Header Analysis (Index 8-15)

Index	Feature	Range	Unit	Description
8	Packet Size Average	(64, 1500)	bytes	Average packet size
9	Packet Rate	(0, 1000)	pps	Packets per second
10	Fragmentation Rate	(0, 1)	ratio	Packet fragmentation frequency
11	Retransmission Rate	(0, 1)	ratio	Packet retransmission frequency
12	Header Anomalies	(0, 1)	score	Header structure anomaly score
13	Protocol Violations	(0, 1)	score	Protocol compliance violations
14	Timing Irregularities	(0, 1)	score	Timing pattern anomalies
15	Sequence Anomalies	(0, 1)	score	Sequence number irregularities

Encryption Protocol Indicators (Index 16-23)

Index	Feature	Range	Unit	Description
16	Encryption Strength	(0, 4)	level	0=None, 1=WEP, 2=WPA, 3=WPA2, 4=WPA3
17	Cipher Suite Score	(0, 100)	score	Encryption algorithm strength
18	Key Management Score	(0, 100)	score	Key exchange security

19	Authentication Method	(0, 5)	type	Authentication mechanism type
20	Certificate Validity	(0, 1)	binary	Certificate validation status
21	Handshake Integrity	(0, 1)	score	Handshake process integrity
22	Encryption Overhead	(0, 1)	ratio	Encryption processing overhead
23	Crypto Agility	(0, 1)	score	Cryptographic flexibility

Traffic Pattern Characteristics (Index 24-31)

Index	Feature	Range	Unit	Description
24	Bandwidth Utilization	(0, 100)	%	Bandwidth usage percentage
25	Connection Duration	(0, 86400)	seconds	Session duration
26	Data Volume	(0, 1000000)	bytes	Total data transferred
27	Session Count	(0, 1000)	count	Number of active sessions
28	Anomaly Score	(0, 1)	score	Traffic pattern anomaly
29	Behavioral Score	(0, 1)	score	User behavior analysis
30	Temporal Pattern	(0, 1)	score	Time-based pattern analysis
31	Geographic Mobility	(0, 1)	score	Location change frequency

Output Specifications

- **Output Shape**: `(12,)` - 12 vulnerability classes
- **Activation**: Softmax
- **Data Type**: `float32`
- **Output Classes**: 12 total classes

Output Classes (12 Total)

Class ID	Class Name	Risk Level	Description
0	SECURE_NETWORK	LOW	Properly secured network
1	WEAK_ENCRYPTION	MEDIUM	Weak encryption protocols
2	OPEN_NETWORK	HIGH	No encryption/open access
3	WPS_VULNERABILITY	HIGH	WPS security weaknesses
4	ROGUE_AP	CRITICAL	Unauthorized access point
5	EVIL_TWIN	CRITICAL	Malicious AP mimicking legitimate
6	DEAUTH_ATTACK	CRITICAL	Deauthentication attack

7	HANDSHAKE_CAPTURE	CRITICAL	Authentication capture attempt
8	FIRMWARE_OUTDATED	MEDIUM	Outdated firmware vulnerabilities
9	DEFAULT_CREDENTIALS	HIGH	Default/weak credentials
10	SIGNAL_LEAKAGE	MEDIUM	Signal beyond intended range
11	UNKNOWN_THREAT	CRITICAL	Unidentified threat pattern

Required Files for Deployment

1. `wifi_vulnerability_cnn_final.h5` - Trained model
2. `wifi_vulnerability_scaler.pkl` - Feature scaler
3. `wifi_vulnerability_cnn_config.json` - Configuration

API Interface

```
# Prediction method signature
predict(features: np.ndarray) -> Dict

# Input: features - numpy array of shape (32,)
# Output: Dictionary with keys:
{
    'vulnerability_type': str,          # Predicted class name
    'confidence': float,                # Confidence score (0-1)
    'risk_level': str,                  # Risk level (LOW/MEDIUM/HIGH/CRITICAL)
    'high_confidence': bool,            # True if confidence >= 0.85
    'timestamp': str,                   # Prediction timestamp
    'model_version': str                # Model version
}
```

Risk Level Mapping

```
RISK_LEVELS = {
    'LOW': ['SECURE_NETWORK'],
    'MEDIUM': ['WEAK_ENCRYPTION', 'SIGNAL_LEAKAGE', 'FIRMWARE_OUTDATED'],
    'HIGH': ['OPEN_NETWORK', 'WPS_VULNERABILITY', 'DEFAULT_CREDENTIALS'],
    'CRITICAL': ['ROGUE_AP', 'EVIL_TWIN', 'DEAUTH_ATTACK', 'HANDSHAKE_CAPTURE',
'UNKNOWN_THREAT']
}
```

Training Configuration

- **Optimizer**: Adam (lr=0.001)
- **Loss Function**: Categorical Crossentropy
- **Batch Size**: 128
- **Max Epochs**: 50
- **Early Stopping**: Patience=10

LSTM Wi-Fi Vulnerability Detection Model

Model Overview

- **Model Type**: Bidirectional LSTM (Long Short-Term Memory)
- **Primary Purpose**: Wi-Fi vulnerability detection with temporal analysis
- **Model Format**: `.h5` (Keras/TensorFlow)
- **Model File**: `wifi_lstm_production.h5`

Technical Specifications

- **Total Parameters**: ~1.8M parameters
- **Model Size**: 45.2 MB
- **Inference Time**: <50ms per prediction
- **Target Accuracy**: 91-94%
- **Confidence Threshold**: 0.82

Input Specifications

- **Input Shape**: `(50, 48)` - 50 timesteps, 48 features
- **Data Type**: `float32`
- **Sequence Length**: 50 timesteps
- **Feature Count**: 48 features per timestep

Input Features Breakdown (48 Total)

Connection Patterns (Features 0-11)

- Connection frequency metrics
- Connection duration patterns
- Connection success/failure rates
- Port scanning indicators

Data Transfer Rates (Features 12-23)

- Upload/download volumes
- Transfer speed patterns
- Data flow directions
- Bandwidth utilization

Authentication Failures (Features 24-35)

- Failed login attempts
- Authentication timing patterns
- Credential stuffing indicators
- Access attempt frequencies

Device Behavior (Features 36-47)

- Device fingerprinting data
- Behavioral anomaly scores
- Usage pattern deviations

- Automation indicators

Output Specifications

- **Output Shape**: `(10,)` - 10 threat classes
- **Activation**: Softmax
- **Data Type**: `float32`

Output Classes (10 Total)

Class ID	Class Name	Description
0	NORMAL_BEHAVIOR	Baseline network activity
1	BRUTE_FORCE_ATTACK	Password cracking attempts
2	RECONNAISSANCE	Network scanning activities
3	DATA_EXFILTRATION	Unauthorized data transfer
4	BOTNET_ACTIVITY	Automated malicious activity
5	INSIDER_THREAT	Legitimate user malicious behavior
6	APT_BEHAVIOR	Advanced Persistent Threats
7	DDOS_PREPARATION	Distributed Denial of Service setup
8	LATERAL_MOVEMENT	Internal network exploration
9	COMMAND_CONTROL	C&C communication patterns

Required Files for Deployment

1. `wifi_lstm_production.h5` - Main model file
2. `wifi_lstm_preprocessor.pkl` - Data preprocessing pipeline
3. `wifi_lstm_metadata.json` - Model metadata and configuration

API Interface

```
# Prediction method signature
predict_threat(model, preprocessor, sequence_data, class_names) -> Dict

# Input: sequence_data - numpy array of shape (50, 48)
# Output: Dictionary with keys:
{
    'predicted_class': str,              # Predicted threat class
    'confidence': float,                 # Confidence score (0-1)
    'prediction_probabilities': dict     # All class probabilities
}
```

Confidence Threshold Guidelines

- **High Confidence**: ≥0.90 (Immediate action required)

- **Medium Confidence**: 0.70-0.89 (Monitor closely)
- **Low Confidence**: <0.70 (Additional validation needed)

Training Configuration

- **Optimizer**: Adam (learning_rate=0.001)
- **Loss**: Categorical Crossentropy
- **Batch Size**: 128
- **Max Epochs**: 100
- **Validation Split**: 0.15

Graph Neural Network (GNN) Model

Model Overview

- **Model Type**: Graph Neural Network with Graph Convolutional Layers
- **Primary Purpose**: Network topology-based vulnerability detection
- **Framework**: TensorFlow/Keras with Spektral
- **Model File**: `gnn_wifi_vulnerability_model.h5`

Technical Specifications

- **Total Parameters**: Custom (depends on graph size)
- **Target Accuracy**: 88-92%
- **Confidence Threshold**: 0.80
- **Dataset Size**: 16,000 samples (2,000 per class)

Input Specifications

- **Node Features**: 24 dimensions per node
- **Edge Features**: 16 dimensions per edge
- **Graph Size**: Variable (5-20 nodes per graph)
- **Input Format**: Graph structure with adjacency matrix

Node Features (24 Dimensions)

Device Characteristics (0-5)

- Device type indicators (Router, AP, Client, IoT, Server)
- Device capability score

Security Configuration (6-11)

- Encryption strength
- Authentication status
- Firewall configuration
- Update status
- Access control level
- Security protocol version

Trust Metrics (12-17)

- Device reputation score
- Historical reliability
- Communication patterns
- Anomaly detection score
- Network behavior metrics
- Trust relationship strength

Historical Data (18-23)

- Previous vulnerability incidents
- Patch compliance history
- Security audit results
- Incident response metrics
- Risk assessment scores
- Compliance status

Edge Features (16 Dimensions)

Connection Characteristics (0-3)

- Connection strength
- Link stability
- Bandwidth utilization
- Latency metrics

Communication Patterns (4-7)

- Traffic frequency
- Data volume
- Communication regularity
- Protocol usage

Data Flow (8-11)

- Flow direction
- Data sensitivity
- Encryption status
- Compression metrics

Security Protocols (12-15)

- Protocol compatibility
- Security level
- Authentication methods
- Encryption algorithms

Output Specifications

- **Output Shape**: `(8,)` - 8 vulnerability classes
- **Activation**: Softmax
- **Data Type**: `float32`

Output Classes (8 Total)

Class ID	Vulnerability Type	Description
0	ISOLATED_VULNERABILITY	Single device with security weakness
1	CASCADING_RISK	Vulnerability that can spread across network
2	CRITICAL_NODE	High-value target with poor security
3	BRIDGE_VULNERABILITY	Weakness in network bridge/gateway
4	CLUSTER_WEAKNESS	Multiple related devices with similar weaknesses
5	PERIMETER_BREACH	External access point compromise
6	PRIVILEGE_ESCALATION	Unauthorized access level increase
7	NETWORK_PARTITION	Risk of network segmentation

Training Configuration

- **Batch Size**: 32
- **Epochs**: 100
- **Learning Rate**: 0.001
- **Validation Split**: 0.2
- **Early Stopping Patience**: 10

Enhanced Crypto-BERT Model

Model Overview

- **Model Type**: Transformer-based Language Model (BERT variant)
- **Primary Purpose**: Cryptographic vulnerability detection in protocol sequences
- **Framework**: TensorFlow/Keras with Transformers
- **Model File**: `crypto_bert_enhanced.h5`

Technical Specifications

- **Total Parameters**: ~4.2M parameters
- **Model Size**: 85-120 MB
- **Memory Usage**: 280-350 MB runtime
- **Inference Time**: <0.01 seconds per sequence
- **Target Accuracy**: 95-98%
- **Confidence Threshold**: 0.88

Input Specifications

- **Input Shape**: `(batch_size, 256)` - 256 tokens per sequence

- **Token Type**: int32 token IDs
- **Attention Mask**: `(batch_size, 256)` - int32 attention mask
- **Vocabulary Size**: 30,000 tokens
- **Max Sequence Length**: 256 tokens

Input Processing

```
# Input format for single sample
input_ids = [101, 23434, 1035, 12809, ...]  # Length: 256
attention_mask = [1, 1, 1, 1, ...]          # Length: 256

# Input tensor shapes
input_ids: (batch_size, 256) int32
attention_mask: (batch_size, 256) int32
```

Output Specifications

- **Output Shape**: `(batch_size, 15)` - 15 vulnerability classes
- **Activation**: Softmax
- **Data Type**: `float32`

Output Classes (15 Total)

Class ID	Vulnerability Type	Severity	Description
0	STRONG_ENCRYPTION	LOW	Robust cryptographic implementation
1	WEAK_CIPHER_SUITE	HIGH	Deprecated encryption methods
2	CERTIFICATE_INVALID	HIGH	SSL/TLS certificate issues
3	KEY_REUSE	MEDIUM	Cryptographic key reuse detected
4	DOWNGRADE_ATTACK	HIGH	Protocol downgrade attempt
5	MAN_IN_MIDDLE	HIGH	MITM attack indicators
6	REPLAY_ATTACK	MEDIUM	Message replay vulnerability
7	TIMING_ATTACK	MEDIUM	Side-channel attack potential
8	QUANTUM_VULNERABLE	MEDIUM	Post-quantum cryptography needed
9	ENTROPY_WEAKNESS	HIGH	Poor random number generation
10	HASH_COLLISION	MEDIUM	Hash function vulnerability
11	PADDING_ORACLE	HIGH	Padding oracle attack possible
12	LENGTH_EXTENSION	MEDIUM	Hash length extension vulnerability
13	PROTOCOL_CONFUSION	MEDIUM	Protocol implementation flaw
14	CRYPTO_AGILITY_LACK	LOW	Limited cryptographic flexibility

Required Files for Deployment

1. `crypto_bert_enhanced.h5` - Model weights and architecture
2. `crypto_bert_enhanced_tokenizer/` - Tokenizer files
3. `crypto_bert_enhanced_metadata.json` - Model specifications

API Interface

```
# Prediction method signature
predict(protocol_sequences, return_confidence=True) -> Dict

# Input: protocol_sequences - List of strings (variable length)
# Output: Dictionary with keys:
{
    'vulnerabilities': List[str],              # Predicted class names
    'confidence_scores': List[float],          # Confidence scores (0-1)
    'high_confidence_predictions': List[bool], # True if confidence >= 0.88
    'raw_predictions': np.ndarray,             # Raw probability distributions
    'inference_time': float,                   # Processing time in seconds
    'meets_confidence_threshold': float        # Percentage meeting threshold
}
```

Training Configuration

- **Optimizer**: Adam (lr=2e-5)
- **Loss Function**: Categorical Crossentropy
- **Batch Size**: 8-32
- **Max Epochs**: 5
- **Validation Split**: 0.2

WiFi LSTM Ensemble Fusion Model

Model Overview

- **Model Type**: Ensemble of Deep Learning and Traditional ML models
- **Primary Purpose**: WiFi network threat detection with superior accuracy
- **Architecture**: 5 models combined using weighted voting
- **Model Files**: Multiple models with ensemble weights

Technical Specifications

- **Component Models**: 5 total models
- **Target Accuracy**: 91-94%
- **Inference Time**: ~0.01 seconds per sample
- **Throughput**: ~1000 samples/second

Input Specifications

- **Input Shape**: `(50, 48)` - 50 timesteps, 48 features
- **Data Type**: `float32`

- **Sequence Length**: 50 timesteps
- **Feature Count**: 48 features per timestep

Component Models

1. **LSTM Model**: `wifi_lstm_model.h5`
2. **CNN-LSTM Hybrid**: `wifi_cnn_lstm_model.h5`
3. **Attention Model**: `wifi_attention_model.h5`
4. **Random Forest**: `wifi_random_forest_model.pkl`
5. **Gradient Boosting**: `wifi_gradient_boosting_model.pkl`

Output Specifications

- **Output Shape**: `(10,)` - 10 threat classes
- **Activation**: Weighted ensemble voting
- **Data Type**: `float32`

Output Classes (10 Total)

Class ID	Class Name	Description
0	NORMAL_BEHAVIOR	Regular network usage
1	BRUTE_FORCE_ATTACK	Password cracking attempts
2	RECONNAISSANCE	Network scanning and information gathering
3	DATA_EXFILTRATION	Unauthorized data theft
4	BOTNET_ACTIVITY	Malicious bot network communication
5	INSIDER_THREAT	Threats from within the organization
6	APT_BEHAVIOR	Advanced Persistent Threat activities
7	DDOS_PREPARATION	Distributed Denial of Service setup
8	LATERAL_MOVEMENT	Spreading across network systems
9	COMMAND_CONTROL	Remote command execution

Required Files for Deployment

1. `wifi_lstm_model.h5` - Main LSTM model
2. `wifi_cnn_lstm_model.h5` - CNN-LSTM hybrid
3. `wifi_attention_model.h5` - Attention-based model
4. `wifi_random_forest_model.pkl` - Random Forest classifier
5. `wifi_gradient_boosting_model.pkl` - Gradient Boosting classifier
6. `wifi_preprocessor.pkl` - Data scaler
7. `wifi_ensemble_weights.pkl` - Model weights
8. `wifi_ensemble_metadata.json` - Model metadata

API Interface

```
# Prediction method signature
predict_threat(ensemble_model, scaler, test_sample, class_names) -> Dict

# Input: test_sample - numpy array of shape (50, 48)
# Output: Dictionary with keys:
{
    'predicted_class': str,        # Predicted threat class
    'confidence': float,           # Ensemble confidence score
    'is_threat': bool,             # True if threat detected
    'take_action': bool,           # True if action required
    'all_probabilities': List[float], # All class probabilities
    'processing_time': float       # Processing time in seconds
}
```

Deployment Guidelines

System Requirements

- **Python**: 3.7+
- **TensorFlow**: 2.8.0+
- **Memory**: 4-8 GB RAM minimum
- **Storage**: 2-5 GB for all models
- **GPU**: Optional but recommended for batch processing

Framework Dependencies

```
# Core Dependencies
tensorflow>=2.8.0
scikit-learn>=1.1.0
numpy>=1.21.0
pandas>=1.3.0
transformers>=4.20.0  # For Crypto-BERT
spektral>=1.0.0       # For GNN
```

Model Loading Pattern

```
# Standard model loading for TensorFlow models
model = tf.keras.models.load_model('model_file.h5')

# Preprocessing pipeline loading
with open('preprocessor.pkl', 'rb') as f:
    preprocessor = pickle.load(f)

# Configuration loading
with open('config.json', 'r') as f:
    config = json.load(f)
```

Performance Monitoring

- **Accuracy Tracking**: Monitor per-class accuracy
- **Confidence Analysis**: Track confidence score distributions
- **Inference Time**: Monitor processing speed
- **Memory Usage**: Track memory consumption

Integration Specifications

Flask API Integration

```
# Required endpoints
POST /predict          # Single sample prediction
POST /batch_predict    # Multiple samples prediction
GET /health            # Model health check
GET /metrics           # Model performance metrics
```

Input Validation

- **Data Type**: Ensure correct numpy array types
- **Shape Validation**: Verify input dimensions
- **Range Checking**: Validate feature value ranges
- **Missing Values**: Handle NaN/null values

Error Handling

- **Model Loading Errors**: Graceful degradation
- **Prediction Errors**: Fallback mechanisms
- **Memory Errors**: Batch size adjustment
- **Timeout Handling**: Processing time limits

Logging and Monitoring

- **Prediction Logs**: Log all predictions with metadata
- **Performance Metrics**: Track model performance
- **Error Logs**: Capture and log errors
- **Audit Trail**: Maintain prediction history

Security Considerations

- **Input Sanitization**: Validate all inputs
- **Rate Limiting**: Prevent abuse
- **Authentication**: Secure API access
- **Data Privacy**: Protect sensitive information

Model-Specific Implementation Notes

CNN Model

- Preprocess input to shape `(32, 1)` before prediction
- Apply StandardScaler normalization
- Use confidence threshold of 0.85
- Handle categorical output with argmax

LSTM Model

- Ensure sequence length is exactly 50 timesteps
- Pad or truncate sequences as needed
- Apply feature scaling per timestep
- Use confidence threshold of 0.82

GNN Model

- Construct graph adjacency matrix
- Normalize node and edge features
- Handle variable graph sizes
- Use confidence threshold of 0.80

Crypto-BERT Model

- Tokenize input sequences to 256 tokens
- Apply attention masks for padding
- Handle batch processing efficiently
- Use confidence threshold of 0.88

Ensemble Model

- Load all 5 component models
- Apply weighted voting based on saved weights
- Aggregate predictions using ensemble logic
- Provide comprehensive confidence scoring

Troubleshooting Guide

Common Issues

1. **Shape Mismatch**: Verify input dimensions match model expectations
2. **Memory Errors**: Reduce batch size or use model optimization
3. **Slow Inference**: Enable GPU acceleration or model quantization
4. **Low Accuracy**: Check preprocessing pipeline and feature scaling
5. **Model Loading**: Ensure all required files are present

Performance Optimization

- **Batch Processing**: Process multiple samples together
- **Model Quantization**: Reduce model size for faster inference

- **GPU Acceleration**: Use CUDA for TensorFlow models
- **Caching**: Cache frequently used preprocessors and models

This documentation provides complete technical specifications for all AI models. Each model has specific input/output requirements, file dependencies, and integration guidelines that must be followed for successful implementation.