Deployment Logic - mmWave Radar AI System

Assignment: Part 4 - Deployment Logic Documentation
Student Name: Arnav Deshmukh
Email: arnavdeshmukh2004@gmail.com
GitHub Repository: https://github.com/Arnav-0/GURUJI-AIR-ASSIGNMENT
Date: November 2025
Institution: Guruji Air

Executive Summary

This document describes the deployment architecture for a 77 GHz FMCW radar-based metal detection system using machine learning. The system achieves **82.5% classification accuracy** using Support Vector Machines (SVM) with real-time processing capability of **14.9 FPS** on CPU. This report details the complete pipeline from signal acquisition to decision output, preprocessing techniques, model inference flow, current limitations, and proposed improvements for production deployment.

Key Achievements:

- SVM Classifier: 82.5% accuracy, 0.90 recall
- Real-time processing: 67ms per frame
- Dataset: 400 synthetic samples with balanced classes
- Preprocessing pipeline: +9% accuracy improvement in cluttered scenarios

---

System Architecture

```
Hardware Radar → Signal Acquisition → Preprocessing → Model Inference → Decision Output
     (77 GHz)      (128×256 samples)    (FFT + Norm)    (SVM/CNN)      (Metal/Non-Metal)
```

Pipeline Components

1. Signal Acquisition Module

- **Input:** Raw IQ data from 77 GHz FMCW radar
- **Specifications:**
  - Sample rate: 256 samples per chirp
  - Chirp rate: 128 chirps per frame
  - Frame rate: 10-30 FPS (configurable)
  - Data format: Complex float32 (I/Q channels)

2. Real-Time Signal Processing

```
def process_radar_frame(raw_signal):
    # Apply window function
    windowed = apply_hamming_window(raw_signal)

    # 2D FFT processing
    range_fft = fft(windowed, axis=1)  # Range dimension
    doppler_fft = fft(range_fft, axis=0)  # Doppler dimension

    # Generate magnitude heatmap
    heatmap = np.abs(doppler_fft)

    # Downsample to model input size (64×64)
    heatmap_resized = resize(heatmap, (64, 64))

    # Normalize
    heatmap_normalized = (heatmap_resized - mean) / std

    return heatmap_normalized
```

Processing Time: ~35ms per frame (CPU), ~8ms (GPU)

Preprocessing Pipeline

Stage 1: Background Subtraction

Purpose: Remove static clutter and environmental noise

```
def background_subtraction(current_frame, background_model):
    # Subtract learned background
    foreground = current_frame - background_model

    # Apply threshold
    foreground[foreground < threshold] = 0

    return foreground
```

Performance Impact: +3% accuracy improvement

Stage 2: Noise Filtering

Purpose: Reduce false positives from sensor noise

Techniques:

- Gaussian smoothing (σ=1.0)
- Median filtering (kernel size: 3×3)
- Morphological operations (erosion + dilation)

```
def noise_filter(heatmap):
    # Gaussian blur
    smoothed = gaussian_filter(heatmap, sigma=1.0)

    # Median filter
    denoised = median_filter(smoothed, size=3)

    return denoised
```

Performance Impact: +6% accuracy improvement (cumulative: +9%)

Stage 3: CFAR Detection

Purpose: Adaptive thresholding for target detection

Algorithm: Cell-Averaging CFAR (CA-CFAR)

- Guard cells: 2
- Training cells: 8
- False alarm rate: 10^-4

```
def cfar_detection(heatmap, pfa=1e-4):
    # Calculate adaptive threshold
    for each_cell in heatmap:
        guard_region = get_guard_cells(cell, size=2)
        training_region = get_training_cells(cell, size=8)

        noise_power = np.mean(training_region)
        threshold = noise_power * scaling_factor(pfa)

        if cell_value > threshold:
            detections.append(cell)

    return detections
```

Model Flow

Inference Pipeline

```
Input Heatmap (64×64)
    ↓
Preprocessing (normalization)
    ↓
Model Selection:
├─ SVM (Primary) → 82.5% accuracy, <5ms inference
└─ CNN (Backup) → 51.2% accuracy, ~40ms inference
    ↓
```

```
Confidence Thresholding (>0.7)
    ↓
Post-processing (smoothing over 5 frames)
    ↓
Final Decision: Metal / Non-Metal
```

Model Selection Logic

Primary: SVM Classifier

- **Advantages:**
    - Fast inference (<5ms on CPU)
    - High accuracy (82.5%)
    - Excellent recall (0.90) - fewer false negatives
    - Low memory footprint (~100KB)
- **Feature Extraction:**
    - Flatten 64×64 heatmap → 4096 features
    - Apply PCA for dimensionality reduction (optional)
    - Normalize features

Backup: CNN Classifier

- **Use Case:** When computational resources allow
- **Architecture:** 2.27M parameters
- **Inference Time:** 40ms (CPU), 8ms (GPU)

Deployment Code

```python
class RadarClassifier:
    def __init__(self):
        self.svm = load_model('svm_model.pkl')
        self.scaler = load_scaler('scaler.pkl')
        self.frame_buffer = []

    def predict(self, heatmap):
        # Flatten and normalize
        features = heatmap.flatten()
        features_scaled = self.scaler.transform(features)

        # SVM prediction
        prediction = self.svm.predict(features_scaled)
        confidence = self.svm.predict_proba(features_scaled)

        # Temporal smoothing
        self.frame_buffer.append(prediction)
        if len(self.frame_buffer) > 5:
            self.frame_buffer.pop(0)
```

```
        # Majority voting
        final_prediction = mode(self.frame_buffer)

        return final_prediction, confidence
```

Real-Time Performance

Component	Time (ms)	Throughput
Signal Acquisition	10	100 FPS
2D FFT Processing	35	28.6 FPS
Preprocessing	15	66.7 FPS
SVM Inference	5	200 FPS
Post-processing	2	500 FPS
Total Pipeline	**67ms**	**14.9 FPS**

Note: With GPU acceleration, total time reduces to ~35ms (28.6 FPS)

Limitations and Improvements

Current Limitations

1. Dataset Limitations

- **Synthetic Data Only:** All training data is simulated
- **Limited Scenarios:** Only binary classification (metal/non-metal)
- **Simplified Physics:** Real-world radar has multipath, interference
- **Impact:** Model may not generalize to real hardware

Evidence: CNN performance (51.2%) suggests overfitting to synthetic patterns

2. Model Performance

- **CNN Underperformance:** 51.2% accuracy (barely better than random)
- **Root Causes:**
  - Learning rate issues
  - Insufficient training epochs
  - Possible class imbalance in training
- **Impact:** Cannot reliably use CNN as backup model

3. Real-Time Constraints

- **CPU Bottleneck:** 67ms per frame limits to 14.9 FPS
- **Latency:** Total system latency ~100ms including decision logic
- **Impact:** May miss fast-moving objects or require frame skipping

4. Environmental Sensitivity

- **Weather:** Rain, fog affect radar signal quality
- **Clutter:** Dense environments create false positives
- **Range:** Performance degrades beyond 5 meters
- **Impact:** Detection accuracy drops from 82.5% to ~64% in clutter

5. Hardware Dependencies

- **Calibration Required:** Each radar unit needs individual calibration
- **Temperature Drift:** Sensor performance varies with temperature
- **Power Consumption:** Continuous operation requires ~5W power
- **Impact:** Field deployment requires careful integration

Proposed Improvements

Short-Term (1-3 months)

1. Real-World Data Collection

- **Action:** Deploy prototype with labeled data collection
- **Goal:** 1000+ real samples from actual radar hardware
- **Expected Impact:** +15-20% accuracy improvement

2. CNN Architecture Optimization

- **Actions:**
  - Reduce learning rate (0.001 → 0.0001)
  - Increase training epochs (50 → 100)
  - Add more aggressive data augmentation
  - Implement focal loss for class imbalance
- **Expected Impact:** CNN accuracy 51.2% → 75-80%

3. GPU Acceleration

- **Action:** Implement CUDA/TensorRT inference
- **Hardware:** NVIDIA Jetson Nano or better
- **Expected Impact:** 67ms → 35ms (28.6 FPS real-time)

Medium-Term (3-6 months)

4. Multi-Class Classification

- **Extension:** Beyond metal/non-metal
  - Plastic
  - Wood
  - Glass
  - Composite materials
- **Dataset Required:** 500+ samples per class
- **Expected Accuracy:** 70-75% (5-class)

5. Advanced Preprocessing

- **Techniques:**
  - Adaptive CFAR with multiple thresholds
  - Machine learning-based clutter removal
  - Temporal filtering across multiple frames
- **Expected Impact:** +5-8% accuracy in cluttered environments

6. Ensemble Methods

- **Approach:** Combine SVM + CNN + Random Forest
- **Voting Scheme:** Weighted by individual confidence
- **Expected Impact:** +3-5% accuracy improvement

Long-Term (6-12 months)

7. Deep Learning Enhancements

- **Architecture:** Attention mechanisms (Transformer-based)
- **Benefits:** Better feature extraction, spatial relationships
- **Expected Accuracy:** 85-90%

8. Edge Deployment Optimization

- **Techniques:**
  - Model quantization (FP32 → INT8)
  - Knowledge distillation (compress 2.27M → 500K params)
  - Pruning (remove 40-50% of weights)
- **Impact:** 2-3x faster inference, 70% less memory

9. Multi-Sensor Fusion

- **Integration:** Combine radar with camera/LiDAR
- **Benefits:** Redundancy, cross-validation
- **Expected Accuracy:** 90-95%

10. Adaptive Learning

- **Approach:** Online learning with user feedback
- **Benefits:** Continuously improve in deployment
- **Implementation:** Federated learning across multiple units

---

Deployment Checklist

Pre-Deployment

- Collect 500+ real-world samples for validation
- Calibrate radar hardware in target environment
- Benchmark inference speed on deployment hardware
- Test in various environmental conditions (rain, fog, night)

- Establish baseline accuracy metrics

Integration

- Implement data acquisition interface
- Set up preprocessing pipeline
- Load trained models (SVM primary, CNN backup)
- Configure confidence thresholds
- Implement logging and monitoring

Validation

- A/B testing against manual inspection
- Measure false positive/negative rates
- Test latency under load
- Verify power consumption
- Document failure modes

Monitoring

- Real-time accuracy tracking
- Alert on confidence drops
- Log edge cases for retraining
- Track model drift over time
- Schedule periodic recalibration

Conclusion & Future Roadmap

The mmWave radar AI system demonstrates strong feasibility for automated metal detection with **82.5% classification accuracy** using SVM. The real-time pipeline achieves **14.9 FPS on CPU**, which is adequate for many industrial and security screening applications. The preprocessing pipeline successfully improves detection accuracy by **9%** in cluttered environments through background subtraction and noise filtering techniques.

Critical Next Steps

Immediate Priorities (1-3 months):

1. **Real-world data collection:** Validate synthetic training with 1000+ samples from actual radar hardware
2. **GPU acceleration:** Reduce pipeline latency from 67ms to 35ms for 28.6 FPS real-time performance
3. **CNN optimization:** Improve CNN accuracy from 51.2% to 75-80% through hyperparameter tuning and architectural changes
4. **Field testing:** Deploy prototype in target environments to measure real-world performance degradation

Medium-term Goals (3-6 months):

- Expand to multi-class classification (metal, plastic, wood, glass, composite)
- Implement ensemble methods combining SVM + CNN + Random Forest for improved accuracy

- Develop adaptive preprocessing for varying environmental conditions
- Optimize for edge deployment with model quantization and pruning

Long-term Vision (6-12 months):

- Achieve 90%+ accuracy with deep learning enhancements (attention mechanisms, transformers)
- Multi-sensor fusion integrating radar with camera/LiDAR for redundancy
- Online learning capability with federated learning across multiple deployed units
- Full edge deployment on embedded systems (NVIDIA Jetson, mobile devices)

Production Readiness Assessment

Current Status: Proof-of-concept validated

Criterion	Notes
Algorithm Performance	SVM meets 82.5% requirement
Real-time Processing	14.9 FPS adequate but GPU recommended
Robustness	Only synthetic data validated
Hardware Integration	Requires actual radar interfacing
Safety/Security	Needs dual-validation and human-in-loop
Scalability	Modular architecture supports scaling

Recommendation: System is ready for **controlled pilot deployment** with human oversight. Not recommended for fully autonomous operation until real-world validation is complete.

Key Takeaways

1. **SVM outperforms CNN** on this dataset due to limited training samples (400) relative to CNN complexity (2.27M parameters)
2. **Preprocessing is critical:** Simple techniques (background subtraction, noise filtering) provide significant accuracy gains (+9%)
3. **Real-time is achievable:** 67ms latency on CPU is acceptable for non-safety-critical applications; GPU can reduce to 35ms
4. **Synthetic-to-real gap** is the primary deployment risk that must be addressed through field data collection
5. **Modular design** enables incremental improvements without system redesign

---

References & Resources

Technical Documentation:

- Project README: GitHub Repository
- Jupyter Notebooks: `/notebooks/` (complete implementation)
- Source Code: `/src/` (radar simulator, models, preprocessing)
- Results: `/outputs/` (27 visualizations, trained models, metrics)

Performance Metrics:

- Classification Results: `/outputs/results/classification_results.json`
- Detection Results: `/outputs/results/hidden_object_detection_results.json`
- Trained Models: `/data/models/` (SVM, CNN checkpoints)

Contact Information:

- **Author:** Arnav Deshmukh
- **Email:** arnavdeshmukh2004@gmail.com
- **GitHub:** https://github.com/Arnav-0/GURUJI-AIR-ASSIGNMENT

Document Version: 1.0
Last Updated: November 22, 2025
Project: mmWave Radar AI - Deployment Logic Documentation
Assignment: Guruji Air - Part 4

This document is part of the mmWave Radar AI assignment submission. For complete implementation details, please refer to the GitHub repository and accompanying Jupyter notebooks.