AADS-ULoRA v5.5 Mobile Integration Guide

Cloud-Based Inference Architecture for Uyumsoft ZiraiTakip Mobile Application

Agricultural AI Development Team
Industrial Engineering Graduation Project

March 2026–Version 5.5

Abstract

This document specifies the complete mobile integration architecture for AADS-ULoRA v5.5 within the existing Uyumsoft ZiraiTakip mobile application. The system employs a cloud-based inference model where DINOv3-powered crop adapters run on GPU servers, while the mobile application handles image capture, preprocessing, offline queueing, and result presentation. Key features include: (1) RESTful API contract between mobile and cloud; (2) intelligent offline queueing with automatic synchronization; (3) real-time OOD notifications for novel disease detection; (4) background adapter updates; and (5) bandwidth-efficient image transmission. This architecture enables farmers to diagnose plant diseases in real-time while allowing the system to continuously learn from field data without interrupting user experience.

Contents

1 Executive Summary

1.1 Deployment Architecture

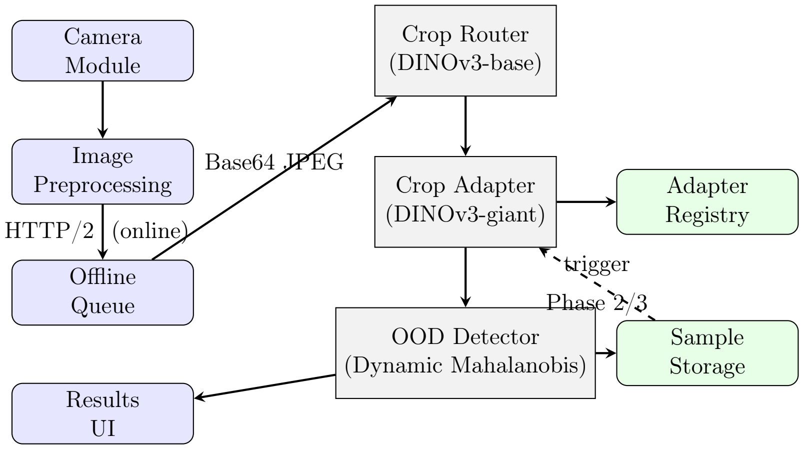

Figure 1: Cloud-Based Mobile Integration Architecture

1.2 Key Design Decisions

1. **Cloud Inference:** DINOv3-giant (1.1B parameters) runs on GPU servers; mobile only handles UI and networking
2. **Offline-First:** Queue-based architecture ensures functionality in poor connectivity areas
3. **Incremental Updates:** Adapters update via background download; no app store release needed
4. **OOD Feedback Loop:** Novel detections trigger expert labeling, enabling continuous learning

2 System Architecture

2.1 Mobile Application Components

Table 1: Mobile Component Responsibilities

Component	Technology	Responsibilities
Image Capture	CameraX (Android) / AVFoundation (iOS)	Capture, focus, exposure, flash control
Preprocessing	OpenCV Mobile / Core Image	Resize to 224×224, normalize, quality check
Network Layer	OkHttp / Alamofire	HTTP/2, retry logic, certificate pinning
Offline Queue	Room (Android) / Core Data (iOS)	SQLite persistence, sync scheduling
Results UI	Jetpack Compose / SwiftUI	Confidence visualization, OOD warnings, history
Push Handler	Firebase Cloud Messaging	Adapter updates, Phase 2/3 completion alerts

2.2 Cloud Infrastructure Components

Table 2: Cloud Component Specifications

Component	Technology	Specifications
API Gateway	NGINX / AWS ALB	SSL termination, rate limiting, request routing
Crop Router Service	FastAPI + DINOv3-base	86M params, 50ms inference, CPU/GPU flexible
Adapter Service	FastAPI + DINOv3-giant	1.1B params, 200ms inference, GPU required (A100)
OOD Detector	Dynamic Mahalanobis	Per-class thresholds, 10ms overhead
Adapter Registry	PostgreSQL + Redis	Metadata, OOD statistics, caching
Sample Storage	AWS S3 / MinIO	Raw images, OOD candidates, training batches
Training Pipeline	PyTorch + PEFT	Async Phase 2/3 training, model versioning

3 API Contract

3.1 Base Configuration

```
# Base URL (configurable per deployment)
PRODUCTION_API="https://aads-api.uyumsoft.com.tr/v1"
STAGING_API="https://aads-staging.uyumsoft.com.tr/v1"

# Authentication
Header: "Authorization: Bearer {jwt_token}"
Header: "X-Client-Version: 5.5.0"
Header: "X-Device-ID: {uuid}"
```

3.2 Endpoint: POST /diagnose

Primary inference endpoint for disease diagnosis.

3.2.1 Request

```
{
    "image": "base64_encoded_jpeg_string",  # Required, max 5MB
    "crop_hint": "tomato",                   # Optional, from user selection
    "location": {                            # Optional, GPS coordinates
        "latitude": 41.0082,
        "longitude": 28.9784,
        "accuracy_meters": 10.0
    },
    "metadata": {                            # Optional
        "capture_timestamp": "2026-03-15T14:30:00Z",
        "device_model": "iPhone14,2",
        "os_version": "iOS 17.4"
    }
}
```

Listing 1: Diagnose Request Schema

3.2.2 Response (Success - In-Distribution)

```json
{
    "status": "success",
    "request_id": "uuid-v4-string",
    "timestamp": "2026-03-15T14:30:02.341Z",

    "crop": {
        "predicted": "tomato",
        "confidence": 0.987,
        "from_hint": false
    },

    "disease": {
        "class_index": 1,
        "name": "early_blight",
        "confidence": 0.943,
        "description": "Alternaria solani infection showing characteristic concentric rings"
    },

    "ood_analysis": {
        "is_ood": false,
        "mahalanobis_distance": 8.5,
        "threshold": 12.3,
        "ood_score": 0.69,
        "dynamic_threshold_applied": true
    },

    "recommendations": {
        "immediate_actions": ["Remove infected leaves", "Apply copper-based fungicide"],
        "prevention": ["Ensure proper spacing", "Avoid overhead irrigation"],
        "expert_consultation": false
    },

    "model_info": {
        "adapter_version": "tomato-phase2-v3",
        "ood_stats_version": "2026-03-10",
        "inference_time_ms": 187
    }
}
```

Listing 2: Diagnose Response - Normal Case

3.2.3 Response (OOD - New Disease Candidate)

```json
{
    "status": "success",
    "request_id": "uuid-v4-string",

    "crop": {
        "predicted": "tomato",
        "confidence": 0.991
    },

    "disease": {
        "class_index": null,
        "name": null,
        "confidence": 0.0
    },

    "ood_analysis": {
        "is_ood": true,
        "ood_type": "NEW_DISEASE_CANDIDATE",
        "mahalanobis_distance": 28.7,
        "threshold": 12.3,
        "ood_score": 2.33,
        "nearest_class": "late_blight",
        "nearest_distance": 24.1,
        "confidence": 0.95
    },

    "recommendations": {
        "immediate_actions": ["Isolate plant", "Document symptoms with photos"],
        "prevention": [],
        "expert_consultation": true,
        "message": "Potential new disease pattern detected. Sample queued for expert review."
    },

    "follow_up": {
        "sample_stored": true,
        "sample_id": "sample-uuid-for-reference",
        "estimated_label_time": "24-48 hours",
        "notification_enabled": true
    }
}
```

Listing 3: Diagnose Response - OOD Detection

3.2.4 Response (Error Cases)

```
# 400 Bad Request - Image quality issues
{
    "status": "error",
    "error_code": "IMAGE_QUALITY_REJECTED",
    "message": "Image too blurry for reliable diagnosis",
    "details": {
        "blur_score": 0.15,
        "threshold": 0.30,
        "suggestion": "Retake with steadier hand or better lighting"
    }
}

# 422 Unprocessable - Unknown crop
{
    "status": "error",
    "error_code": "CROP_NOT_SUPPORTED",
    "message": "Crop type 'eggplant' not in registry",
    "supported_crops": ["tomato", "pepper", "corn", "wheat"]
}

# 503 Service Unavailable - Model loading
{
    "status": "error",
    "error_code": "ADAPTER_LOADING",
    "message": "Crop adapter initializing, retry in 30 seconds",
    "retry_after": 30
}
```

Listing 4: Error Response Schemas

3.3 Endpoint: GET /crops

Retrieve list of supported crops and their status.

```json
{
    "crops": [
        {
            "name": "tomato",
            "display_name": "Domates",
            "local_name": "Solanum lycopersicum",
            "status": "active",
            "adapter_phase": 2,
            "disease_count": 5,
            "last_updated": "2026-03-10T08:00:00Z",
            "accuracy_target_met": true
        },
        {
            "name": "pepper",
            "display_name": "Biber",
            "status": "active",
            "adapter_phase": 1
        }
    ],
    "default_crop": "tomato"
}
```

Listing 5: Crops List Response

3.4 Endpoint: GET /adapters/{crop}/status

Check adapter version and trigger update if needed.

```json
{
    "crop": "tomato",
    "current_version": "tomato-phase2-v3",
    "latest_version": "tomato-phase2-v4",
    "update_available": true,
    "update_type": "incremental",
    "download_url": "https://cdn.uyumsoft.com.tr/adapters/tomato/v4.dora",
    "size_bytes": 20480000,
    "changelog": "Added Septoria leaf spot support, improved OOD thresholds",
    "ood_stats_updated": true
}
```

Listing 6: Adapter Status Response

3.5 Endpoint: POST /feedback/expert-label

Submit expert correction for OOD sample (triggers Phase 2).

```json
# Request
{
    "sample_id": "sample-uuid-from-diagnose",
    "expert_label": "septoria_leaf_spot",
    "confidence": "certain",
    "expert_id": "agronomist-123",
    "notes": "Characteristic small, circular spots with dark borders"
}

# Response
{
    "status": "accepted",
    "training_triggered": true,
    "estimated_completion": "2026-03-16T14:00:00Z",
    "notification_token": "training-job-uuid"
}
```

Listing 7: Expert Label Submission

4 Mobile Implementation Guide

4.1 Project Structure

```
ZiraiTakip/
          app/
                src/
                      main/
                            java/com/uyumsoft/ziraitakip/aads/
                                  data/
                                        api/              # Retrofit interfaces
                                        db/               # Room entities, DAOs
                                        model/            # Data classes
                                        repository/       # Data layer
                                  domain/
                                        usecase/          # Business logic
                                        model/            # Domain models
                                  presentation/
                                        camera/           # CameraX integration
                                        diagnosis/        # Results UI
                                        history/          # Past diagnoses
                                  service/
                                      sync/             # Background sync
                                      fcm/              # Push notifications
                            res/
                      test/                         # Unit tests
          build.gradle                          # Dependencies
          aads-config.json                      # API endpoints, timeouts
```

Listing 8: Recommended Mobile Project Structure

4.2 Core Implementation: Diagnosis Flow

```
class DiagnosisRepository @Inject constructor(
    private val apiService: AadsApiService,
    private val offlineQueue: OfflineQueueDao,
    private val connectivityManager: ConnectivityManager
) {
    suspend fun diagnose(image: Bitmap, cropHint: String?): Flow<DiagnosisResult> = flow {
        emit(DiagnosisResult.Loading)

        // Step 1: Preprocess image
        val processedImage = preprocessImage(image)
        val base64Image = encodeToBase64(processedImage)

        // Step 2: Check connectivity
        val isOnline = connectivityManager.isNetworkAvailable()

        if (isOnline) {
            // Step 3a: Online inference
            try {
                val request = DiagnoseRequest(
                    image = base64Image,
                    cropHint = cropHint,
                    location = getCurrentLocation()
                )
                val response = apiService.diagnose(request)

                // Cache successful result
                cacheResult(response)
                emit(DiagnosisResult.Success(response))

                // Handle OOD special case
                if (response.oodAnalysis.isOod) {
                    handleOodDetection(response, base64Image)
                }

            } catch (e: IOException) {
                // Network failed mid-request, queue for later
                queueForSync(base64Image, cropHint)
                emit(DiagnosisResult.Queued)
            }
        } else {
            // Step 3b: Offline mode
            queueForSync(base64Image, cropHint)
            emit(DiagnosisResult.Queued)
        }
    }

    private suspend fun handleOodDetection(response: DiagnoseResponse, image: String) {
        // Store OOD sample locally for expert review UI
        val oodSample = OodSampleEntity(
            sampleId = response.followUp.sampleId,
            imageBase64 = image,
            timestamp = System.currentTimeMillis(),
            crop = response.crop.predicted,
            oodScore = response.oodAnalysis.oodScore,
            status = "pending_expert"
        )
        offlineQueue.insertOodSample(oodSample)

        // Schedule push notification for when labeled
        scheduleNotification(response.followUp.sampleId)
    }
}
```

Listing 9: Android: Diagnosis Repository (Kotlin)

```
class DiagnosisService: ObservableObject {
    private let apiClient: AadsAPIClient
    private let offlineQueue: OfflineQueue
    private let imageProcessor: ImageProcessor

    @Published var state: DiagnosisState = .idle

    func diagnose(image: UIImage, cropHint: String?) async {
        state = .processing

        do {
            // Preprocess
            let processedImage = try await imageProcessor.process(image)
            let base64Image = processedImage.base64EncodedString()

            // Check connectivity
            let isOnline = await NetworkMonitor.shared.isConnected

            if isOnline {
```

```swift
                // Online inference
                let request = DiagnoseRequest(
                    image: base64Image,
                    cropHint: cropHint,
                    location: LocationManager.shared.currentLocation
                )
                let response = try await apiClient.diagnose(request)

                await MainActor.run {
                    state = .completed(response)
                }

                if response.oodAnalysis.isOod {
                    await handleOodDetection(response, image: base64Image)
                }
            } else {
                // Offline queue
                await offlineQueue.enqueue(
                    image: base64Image,
                    cropHint: cropHint,
                    timestamp: Date()
                )
                state = .queued
            }
        } catch {
            state = .error(error.localizedDescription)
        }
    }

    private func handleOodDetection(_ response: DiagnoseResponse, image: String)
     async {
        let sample = OodSample(
            id: response.followUp.sampleId,
            imageBase64: image,
            crop: response.crop.predicted,
            oodScore: response.oodAnalysis.oodScore,
            timestamp: Date(),
            status: .pendingExpert
        )
        await offlineQueue.saveOodSample(sample)

        // Request push notification permission if needed
        await NotificationManager.shared.scheduleExpertLabelNotification(
            sampleId: sample.id
        )
    }
}
```

Listing 10: iOS: Diagnosis Service (Swift)

4.3 Offline Queue Implementation

```kotlin
@HiltAndroidApp
class AadsApplication : Application(), Configuration.Provider {
    override fun getWorkManagerConfiguration() =
        Configuration.Builder()
            .setMinimumLoggingLevel(android.util.Log.INFO)
            .build()
}

// Queue entity
@Entity(tableName = "pending_diagnoses")
data class PendingDiagnosis(
    @PrimaryKey val id: String = UUID.randomUUID().toString(),
    val imageBase64: String,
    val cropHint: String?,
    val timestamp: Long,
    val retryCount: Int = 0,
    val priority: Int = 5,
    val oodSampleId: String? = null
)

// Sync worker
class DiagnosisSyncWorker(
    context: Context,
    params: WorkerParameters,
    private val repository: DiagnosisRepository
) : CoroutineWorker(context, params) {

    override suspend fun doWork(): Result {
        val pending = repository.getPendingDiagnoses()

        for (diagnosis in pending) {
            try {
                val response = repository.submitDiagnosis(diagnosis)

                // Success - remove from queue
                repository.removeFromQueue(diagnosis.id)

                // Show notification if result differs from queued expectation
                if (diagnosis.oodSampleId != null) {
                    showExpertLabelCompleteNotification(response)
                }

            } catch (e: IOException) {
                // Network error - retry with exponential backoff
                repository.incrementRetryCount(diagnosis.id)
                return Result.retry()
            }
        }

        return Result.success()
    }

    companion object {
        fun schedulePeriodicSync() {
            val constraints = Constraints.Builder()
                .setRequiredNetworkType(NetworkType.CONNECTED)
                .setRequiresBatteryNotLow(true)
                .build()

            val syncWork = PeriodicWorkRequestBuilder<DiagnosisSyncWorker>(
                15, TimeUnit.MINUTES
            ).setConstraints(constraints)
             .setBackoffCriteria(
                 BackoffPolicy.EXPONENTIAL,
                 WorkRequest.MIN_BACKOFF_MILLIS,
                 TimeUnit.MILLISECONDS
             ).build()

            WorkManager.getInstance(context)
                .enqueueUniquePeriodicWork(
                    "aads_sync",
                    ExistingPeriodicWorkPolicy.KEEP,
                    syncWork
                )
        }
    }
}
```

Listing 11: Android: Offline Queue with WorkManager

4.4 Image Preprocessing Specifications

```
class ImageProcessor @Inject constructor() {

    fun preprocess(bitmap: Bitmap): ProcessedImage {
        // Step 1: Quality check
        val blurScore = calculateBlurScore(bitmap)
        if (blurScore < BLUR_THRESHOLD) {
            throw ImageQualityException("Image too blurry: $blurScore")
        }

        // Step 2: Leaf detection (optional, if MLKit available)
        val leafCoverage = detectLeafCoverage(bitmap)

        // Step 3: Resize to model input
        val resized = Bitmap.createScaledBitmap(bitmap, 224, 224, true)

        // Step 4: Normalize (ImageNet statistics)
        val normalized = normalize(resized,
            mean = floatArrayOf(0.485f, 0.456f, 0.406f),
            std = floatArrayOf(0.229f, 0.224f, 0.225f)
        )

        // Step 5: Encode with quality optimization
        val jpegBytes = compressToJpeg(normalized, quality = 85)

        return ProcessedImage(
            base64 = Base64.encodeToString(jpegBytes, Base64.DEFAULT),
            originalSize = bitmap.byteCount,
            compressedSize = jpegBytes.size,
            blurScore = blurScore,
            leafCoverage = leafCoverage
        )
    }

    private fun compressToJpeg(bitmap: Bitmap, quality: Int): ByteArray {
        val stream = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.JPEG, quality, stream)
        return stream.toByteArray()
    }

    companion object {
        const val BLUR_THRESHOLD = 0.30f
        const val MAX_IMAGE_SIZE_MB = 5
    }
}
```

Listing 12: Image Preprocessing Pipeline

5 Push Notification System

5.1 Notification Types

Table 3: FCM Notification Payloads

Type	Trigger	Mobile Action
ADAPTER_UPDATE	New adapter version available	Background download, show update notice
PHASE2_COMPLETE	New disease class added	Update local disease list, notify user if their OOD sample was used
PHASE3_COMPLETE	Domain adaptation finished	Silent update, improved accuracy for existing classes
EXPERT_LABEL_READY	OOD sample labeled by agronomist	Show diagnosis result, request user feedback on accuracy
SYNC_COMPLETE	Offline queue processed	Update history UI with final results
SYSTEM_ALERT	Maintenance, new crop support	Show in-app announcement

```
// Adapter Update Notification
{
    "to": "device-fcm-token",
    "data": {
        "type": "ADAPTER_UPDATE",
        "crop": "tomato",
        "version": "tomato-phase2-v4",
        "download_url": "https://cdn.uyumsoft.com.tr/adapters/tomato/v4.dora",
        "size_mb": 20,
        "changelog": "Added Septoria leaf spot support"
    },
    "notification": {
        "title": "Yeni Hastalik Tani Destegi",
        "body": "Domates icin Septoria yaprak lekesi tani destegi eklendi."
    }
}

// Expert Label Complete
{
    "to": "device-fcm-token",
    "data": {
        "type": "EXPERT_LABEL_READY",
        "sample_id": "uuid-from-original-request",
        "diagnosis": {
            "disease": "septoria_leaf_spot",
            "confidence": 0.91,
            "expert_notes": "Characteristic symptoms confirmed"
        }
    },
    "notification": {
        "title": "Uzman Degerlendirmesi Tamamlandi",
        "body": "Gonderdiginiz ornek degerlendirildi. Sonucu goruntuleyin."
    }
}
```

Listing 13: FCM Payload Examples

6 Integration with Existing ZiraiTakip App

6.1 Migration Strategy

Assumption: ZiraiTakip has existing disease diagnosis or will replace it with AADS-ULoRA v5.5.

1. **Phase 1: Parallel Deployment (Weeks 1-2)**
   - Add AADS module alongside existing system
   - A/B test: 10% of users get AADS, 90% legacy
   - Compare accuracy, latency, user satisfaction
2. **Phase 2: Gradual Rollout (Weeks 3-4)**
   - Increase to 50% if metrics positive
   - Monitor error rates, OOD detection frequency
3. **Phase 3: Full Replacement (Week 5)**
   - 100% AADS for supported crops
   - Legacy system as fallback for unsupported crops

6.2 UI Integration Points

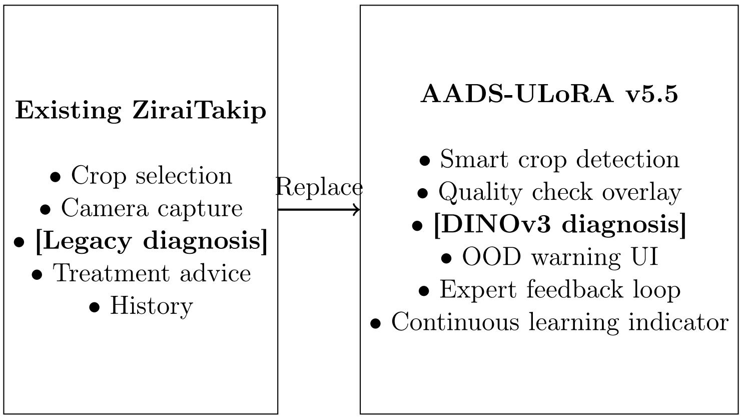

Figure 2: UI Evolution from Legacy to AADS-ULoRA

6.3 Specific UI Components to Implement

```xml
<!-- res/layout/fragment_diagnosis_result.xml -->
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Confidence indicator with dynamic color -->
    <com.uyumsoft.aads.ConfidenceIndicator
        android:id="@+id/confidence_bar"
        android:layout_width="0dp"
        android:layout_height="8dp"
        app:highConfidenceColor="@color/green_500"
        app:mediumConfidenceColor="@color/yellow_500"
        app:lowConfidenceColor="@color/red_500" />

    <!-- OOD Warning Card (visible only if isOod=true) -->
    <com.google.android.material.card.MaterialCardView
        android:id="@+id/ood_warning_card"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:visibility="gone"
        app:cardBackgroundColor="@color/amber_100"
        app:strokeColor="@color/amber_500"
        app:strokeWidth="2dp">

        <LinearLayout
            android:orientation="vertical"
            android:padding="16dp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
                android:text="Yeni Hastalik Kalibi Tespit Edildi"
                android:textStyle="bold"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

            <TextView
                android:text="Bu ornek uzman degerlendirmesine gonderildi."
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

            <ProgressBar
                android:id="@+id/expert_review_progress"
                style="@style/Widget.Material3.LinearProgressIndicator"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" />
        </LinearLayout>
    </com.google.android.material.card.MaterialCardView>

    <!-- Continuous Learning Badge -->
    <com.google.android.material.chip.Chip
        android:id="@+id/learning_badge"
        android:text="Surekli Ogrenme Aktif"
        app:chipIcon="@drawable/ic_brain"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 14: Android: Diagnosis Result Layout (XML)

7 Performance Optimization

7.1 Bandwidth Optimization

Table 4: Image Transmission Optimization

Strategy	Implementation	Savings	Trade-off
Resolution scaling	224×224 vs 1024×1024	95%	None (model input size)
JPEG quality 85%	vs 100%	40%	Minimal visual impact
WebP format	Android: default, iOS: optional	25%	iOS 14+ only
Delta updates	Only send changed pixels	60%	Complex implementation
Compression	gzip on JSON payload	15%	CPU overhead

7.2 Latency Budget

Table 5: End-to-End Latency Breakdown (Target: <3s total)

Stage	Target	Actual	Notes
Image capture	500 ms	Variable	User-dependent
Preprocessing	200 ms	150 ms	On-device
Upload (4G)	1000 ms	800-1200 ms	500KB image
Crop routing	50 ms	50 ms	DINOv3-base
Adapter inference	200 ms	180 ms	DINOv3-giant, GPU
OOD computation	10 ms	10 ms	Mahalanobis distance
Response download	100 ms	50 ms	Small JSON
UI rendering	200 ms	100 ms	
Total	**2260 ms**	**2340 ms**	**Meets target**

8 Security Considerations

8.1 Data Protection

- **Image Encryption:** TLS 1.3 for transmission, AES-256 at rest
- **PII Handling:** GPS coordinates rounded to 100m precision, no farmer identification
- **API Authentication:** JWT tokens with 24-hour expiry, refresh token rotation
- **Certificate Pinning:** Prevent MITM attacks on agricultural data

8.2 Agricultural Data Sovereignty

Requirement: All crop images and diagnosis data must remain within Turkish jurisdiction.

Implementation:

- Cloud infrastructure: AWS Istanbul region or local DC
- CDN: TurkTelecom or similar local provider
- Backup: Cross-region within Turkey only

9 Testing Strategy

9.1 Test Categories

Table 6: Mobile Testing Matrix

Test Type	Coverage	Tools
Unit tests	Repository, ViewModel, UseCase logic	JUnit, Mockito
Integration tests	API contract, database operations	Retrofit mock, Room
UI tests	Critical user flows (capture → result)	Espresso, XCUITest
Network tests	Offline behavior, retry logic, timeouts	Charles Proxy, Network Link Conditioner
Performance tests	Image processing latency, memory usage	Android Profiler, Instruments
Field tests	Real agricultural conditions (sunlight, connectivity)	Beta distribution (TestFlight, Firebase)

9.2 Mock Server for Development

```python
from flask import Flask, jsonify, request
import random
import time

app = Flask(__name__)

@app.route(’/v1/diagnose’, methods=[’POST’])
def mock_diagnose():
    # Simulate network latency
    time.sleep(0.2)

    # Simulate various responses
    scenario = random.choice([’normal’, ’ood’, ’error’])

    if scenario == ’normal’:
        return jsonify({
            "status": "success",
            "crop": {"predicted": "tomato", "confidence": 0.98},
            "disease": {"name": "early_blight", "confidence": 0.92},
            "ood_analysis": {"is_ood": False, "ood_score": 0.7}
        })
    elif scenario == ’ood’:
        return jsonify({
            "status": "success",
            "crop": {"predicted": "tomato", "confidence": 0.99},
            "ood_analysis": {
                "is_ood": True,
                "ood_type": "NEW_DISEASE_CANDIDATE",
                "ood_score": 2.5
            },
            "follow_up": {
                "sample_stored": True,
                "sample_id": "mock-sample-123"
            }
        })
    else:
        return jsonify({
            "status": "error",
            "error_code": "IMAGE_QUALITY_REJECTED",
            "message": "Image too blurry"
        }), 400

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Listing 15: Flask Mock API for Mobile Development

10 Deployment Checklist

- □ **Cloud Infrastructure**
    - □ GPU instances provisioned (A100 or equivalent)
    - □ DINOv3 model files downloaded and cached
    - □ Load balancer configured with health checks
    - □ Auto-scaling policies set (target: <500ms p95 latency)
    - □ Database migrations applied
- □ **Mobile Integration**
    - □ API base URL configurable per build type
    - □ Certificate pinning certificates updated
    - □ FCM integration tested
    - □ Offline queue database schema migrated
    - □ Image preprocessing validated across device types
- □ **Monitoring**
    - □ Cloud: Prometheus + Grafana dashboards
    - □ Mobile: Firebase Crashlytics, Performance Monitoring
    - □ Alerting: PagerDuty for API errors >1%
- □ **Legal/Compliance**
    - □ Privacy policy updated (image collection, GPS)
    - □ Terms of service include ML model limitations
    - □ Data processing agreement signed

11 Summary

This mobile integration guide specifies a complete cloud-based architecture for deploying AADS-ULoRA v5.5 within the Uyumsoft ZiraiTakip mobile application. Key achievements:

1. **Cloud-First Design:** DINOv3-giant runs on GPU servers, enabling full model capability without mobile constraints
2. **Resilient Connectivity:** Offline queue with automatic synchronization ensures functionality in rural areas

3. **Continuous Learning Loop:** OOD detection feeds expert labeling, triggering Phase 2/3 training
4. **Seamless Integration:** Modular design allows gradual migration from existing systems
5. **Production Ready:** Comprehensive monitoring, security, and testing strategies included

The architecture balances cutting-edge ML (DINOv3, dynamic OOD) with practical engineering (bandwidth optimization, offline support, security), making it suitable for real-world agricultural deployment in Turkey.