Development Tasks by Domain

Project Goal: Build a working embedded intelligent microscopy system that identifies and counts microscopic marine organisms for SIH 2025 finals.

Core System: Raspberry Pi 4/5 + IMX477 HQ Camera + 300X C-mount Macro Lens + LED backlight + TFLite CNN model + FastAPI + Streamlit Dashboard

Pipeline: Capture → Preprocess → Segment → Classify → Count → Store → Visualize

---

1. HARDWARE & EMBEDDED SYSTEMS

Critical (Must Work for Demo)

- [ ] Install Raspberry Pi OS (64-bit) on Pi 4/5
- [ ] Update system packages: `sudo apt update && sudo apt upgrade`
- [ ] Enable camera interface via `sudo raspi-config`
- [ ] Install dependencies:

```
sudo apt install python3-pip python3-opencv libcamera-tools
pip3 install picamera2 numpy pillow opencv-python
```

- [ ] Connect IMX477 HQ Camera to Pi CSI port
- [ ] Test camera: `libcamera-hello --list-cameras` and `libcamera-jpeg -o test.jpg`
- [ ] Connect 300X C-mount macro lens to IMX477
- [ ] Set up LED backlight on GPIO pin (test with simple on/off script)
- [ ] Create local storage structure:

```
/home/pi/aqualens/
  images/raw/
  images/processed/
  models/
  database/
  logs/
```

- [ ] Write basic camera capture script (`camera_control.py`):
  - Use Picamera2 to capture high-res images (minimum 2592×1944)

    - Save with timestamp naming: `YYYYMMDD_HHMMSS.jpg`
    - Test capture with and without LED backlight

Important (For Better Demo)

- [ ] Implement LED PWM brightness control (`led_control.py`)
- [ ] Test different LED brightness levels for optimal image clarity
- [ ] Create calibration script:
    - Use known-size reference object (e.g., 100µm scale bar)
    - Calculate µm per pixel ratio
    - Save calibration constant to config file
- [ ] Set up Neo-6M GPS module (UART/USB connection)
- [ ] Install GPS library: `pip3 install gpsd-py3` or use `gps3`
- [ ] Write GPS module (`gps_handler.py`):
    - `get_coordinates()` → returns (lat, lon) or None
    - Timeout after 30s if no GPS fix
- [ ] Create main system control script (`system_main.py`)
- [ ] Test complete capture workflow: LED on → capture → LED off → save with GPS

Nice-to-Have (If Time Permits)

- [ ] Implement auto-focus using Laplacian variance (capture multiple focus levels, select sharpest)
- [ ] Add battery monitoring (if using UPS HAT)
- [ ] Create systemd service for auto-start: `/etc/systemd/system/aqualens.service`
- [ ] Design/3D print waterproof enclosure
- [ ] Power optimization (sleep mode between captures)

---

2. IMAGE PREPROCESSING

Critical (Must Work for Demo)

- [ ] Create preprocessing pipeline module (`preprocess.py`)
- [ ] Implement basic functions:
    - `normalize_image(img)` - Convert to 0-1 range or standardize
    - `denoise(img)` - Gaussian blur (cv2.GaussianBlur with kernel 3×3 or 5×5)
    - `resize_for_inference(img)` - Resize to model input size (e.g., 224×224)
- [ ] Test preprocessing on sample images
- [ ] Visualize before/after preprocessing

Important (For Better Results)

- [ ] Implement illumination correction:
  - CLAHE (Contrast Limited Adaptive Histogram Equalization)
  - Function: `correct_illumination(img)`
- [ ] Implement background removal:
  - Adaptive thresholding or Otsu's method
  - Function: `remove_background(img)`
- [ ] Create preprocessing config file (`preprocess_config.yaml`):
  - Store parameters like blur kernel size, CLAHE clip limit, etc.
- [ ] Add image quality check:
  - Reject blurry images (Laplacian variance threshold)
  - Check brightness/contrast

Nice-to-Have

- [ ] Advanced illumination correction (Retinex, homomorphic filtering)
- [ ] Morphological operations (opening/closing to remove noise)
- [ ] Edge enhancement for better segmentation

---

3. MACHINE LEARNING & MODEL DEVELOPMENT

Critical (Must Work for Demo)

- [ ] Download plankton datasets:
  - SEANOE: https://www.seanoe.org/data/00829/94052/
  - EAWAG: https://opendata.eawag.ch/dataset/deep-learning-classification-of-zooplankton-from-lakes
  - Figshare: https://figshare.com/articles/dataset/.../17010605
- [ ] Organize dataset into `/train` , `/val` , `/test` folders (70/15/15 split)
- [ ] Select 5-10 common plankton classes for MVP (don't try all species at once)
- [ ] Create simple data loader (`data_loader.py`):
  - Load images and labels
  - Basic augmentation: rotation, flip, brightness adjustment
- [ ] Choose lightweight model architecture:
  - **Option 1:** MobileNetV2 (pretrained on ImageNet) + fine-tune
  - **Option 2:** EfficientNet-Lite
  - **Option 3:** Custom small CNN
- [ ] Write training script (`train.py`):

    - Use transfer learning (freeze early layers, train last few layers)
    - Loss: categorical cross-entropy
    - Optimizer: Adam with learning rate ~1e-4
    - Metrics: accuracy, per-class precision/recall
- [ ] Train for minimum 20 epochs, save best model weights
- [ ] Achieve at least 70% validation accuracy for MVP (aim for 80%+)

Important (For Better Accuracy)

- [ ] Implement comprehensive data augmentation:
    - Rotation (±180°)
    - Horizontal/vertical flips
    - Zoom (0.8-1.2×)
    - Brightness/contrast (±20%)
    - Add slight Gaussian noise
- [ ] Handle class imbalance:
    - Weighted loss function OR
    - Oversampling minority classes OR
    - Use class weights in training
- [ ] Implement learning rate scheduling (ReduceLROnPlateau)
- [ ] Add early stopping (patience ~5 epochs)
- [ ] Use TensorBoard for training visualization
- [ ] Create evaluation script (`evaluate.py`):
    - Generate confusion matrix
    - Per-class precision/recall/F1
    - Save classification report
- [ ] Test on validation set and analyze errors
- [ ] Fine-tune hyperparameters (learning rate, batch size, augmentation strength)

Nice-to-Have

- [ ] Try multiple architectures and ensemble predictions
- [ ] Implement Grad-CAM for explainability
- [ ] Use mixup or cutmix augmentation
- [ ] Export training logs and model card documentation

---

4. MODEL OPTIMIZATION FOR EDGE DEPLOYMENT

Critical (Must Work on Pi)

- [ ] Install TensorFlow Lite converter (or use PyTorch Mobile if using PyTorch)
- [ ] Convert trained model to TFLite format (`convert_tflite.py`):

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
```

- [ ] Apply post-training quantization:
  - Dynamic range quantization (reduces size by ~4×)
  - OR INT8 quantization with representative dataset
- [ ] Save as `model_quantized.tflite`
- [ ] Test quantized model accuracy (should be within 2-3% of original)
- [ ] Copy model to Pi: `/home/pi/aqualens/models/`
- [ ] Install TFLite runtime on Pi:

```
pip3 install tflite-runtime
```

- [ ] Write inference script for Pi (`inference.py`):
  - Load TFLite model
  - Preprocess input image
  - Run inference
  - Return class predictions + confidence scores
- [ ] Test inference on Pi with sample images
- [ ] Benchmark inference time (target: <2 seconds per image)

Important (For Performance)

- [ ] Profile model on Pi:
  - Measure CPU usage
  - Measure memory consumption
  - Measure inference latency
- [ ] Optimize if too slow:
  - Reduce input image size (224×224 → 160×160)
  - Use smaller model architecture
  - Enable XNNPACK delegate for faster inference
- [ ] Implement batch processing if analyzing multiple organisms

Nice-to-Have

- [ ] Try pruning model weights
- [ ] Explore TensorRT or ONNX runtime

5. DETECTION & SEGMENTATION

Critical (For Counting Multiple Organisms)

- [ ] Implement basic organism detection (`detect.py`):
  - **Option 1 (Simple):** Contour detection
    - Convert to grayscale/binary
    - Find contours with cv2.findContours
    - Filter by size (remove noise)
  - **Option 2 (Better):** Traditional blob detection
    - SimpleBlobDetector with size/circularity filters
- [ ] Extract bounding boxes for each detected organism
- [ ] Crop individual organisms from image
- [ ] Pass each crop to classification model
- [ ] Test on images with 2-10 organisms

Important (For Overlapping Organisms)

- [ ] Implement watershed segmentation (`segment.py`):
  - Distance transform + find peaks as markers
  - Apply watershed algorithm to separate touching organisms
  - Function: `separate_overlaps(binary_img)`
- [ ] Visualize segmentation masks (color-coded regions)
- [ ] Handle edge cases (partially visible organisms, debris)

Nice-to-Have

- [ ] Train YOLO model for object detection
- [ ] Use U-Net for semantic segmentation
- [ ] Implement morphological operations for better separation

---

6. COUNTING & ANALYSIS

Critical (Must Work for Demo)

- [ ] Create counting module (`count.py`)
- [ ] Implement per-species counting:

    - Input: list of (class_name, confidence) tuples
    - Output: dictionary {species: count}
- [ ] Calculate total organism count
- [ ] Filter low-confidence predictions (threshold: 0.5 or 0.6)
- [ ] Save counts to structured format (JSON or CSV)

Important (For Better Analysis)

- [ ] Implement size measurement:
    - Use calibration constant (µm/pixel)
    - Calculate organism size from bounding box area
    - Function: `calculate_size(bbox_area, calibration_factor)`
- [ ] Add confidence statistics:
    - Average confidence per species
    - Min/max confidence
- [ ] Calculate biodiversity metrics:
    - Shannon diversity index
    - Simpson's diversity index
    - Species richness

Nice-to-Have

- [ ] Abundance trends over time
- [ ] Size distribution histograms
- [ ] Spatial distribution analysis

---

7. BACKEND & DATA MANAGEMENT

Critical (Must Work for Demo)

- [ ] Create FastAPI project structure on Pi:

```
backend/
  main.py
  models.py       # Pydantic models
  database.py     # SQLite setup
  routes/
    capture.py
    analyze.py
```

```
services/
  inference_service.py
```

- [ ] Set up SQLite database (`aqualens.db`):
  - Table: `samples`
    - id (PK), timestamp, lat, lon, image_path, status
  - Table: `detections`
    - id (PK), sample_id (FK), species, count, avg_confidence
  - Table: `organisms` (optional)
    - id (PK), sample_id (FK), species, bbox, size_um, confidence
- [ ] Create database helper functions (`database.py`)
- [ ] Implement core API endpoints:
  - `POST /capture` - Trigger camera capture
    - Call camera_control.capture_image()
    - Save to database with GPS coordinates
    - Return sample_id
  - `POST /analyze/{sample_id}` - Run analysis
    - Load image
    - Preprocess → detect → classify → count
    - Save results to database
    - Return species counts
  - `GET /results/{sample_id}` - Get analysis results
    - Return counts, image path, metadata
- [ ] Test API with curl or Postman
- [ ] Run FastAPI on Pi: `uvicorn main:app --host 0.0.0.0 --port 8000`

Important (For Better System)

- [ ] Add endpoint `GET /samples` - List all samples with filters (date range, location)
- [ ] Add endpoint `GET /export/{sample_id}` - Export as CSV/JSON
- [ ] Implement error handling and logging
- [ ] Add request validation with Pydantic
- [ ] Create background task queue for long-running analysis (optional: Celery/RQ)

Nice-to-Have

- [ ] Cloud storage integration (S3/GCS)
- [ ] Cloud database (PostgreSQL)
- [ ] Authentication (JWT tokens)
- [ ] WebSocket for real-time updates

---

8. FRONTEND & VISUALIZATION

Critical (Must Work for Demo)

- [ ] **Choose framework: Streamlit** (fastest for MVP)
- [ ] Create Streamlit dashboard (`app.py`):
  - Page 1: Upload/Capture
    - Button to trigger capture via API: `POST /capture`
    - Display captured image
    - Button to analyze: `POST /analyze/{id}`
  - Page 2: Results
    - Fetch results: `GET /results/{id}`
    - Display species counts as table
    - Show captured image with detections overlaid
    - Display timestamp and GPS coordinates
  - Page 3: History
    - List all past samples: `GET /samples`
    - Click to view details
- [ ] Deploy Streamlit on Pi or laptop: `streamlit run app.py --server.port 8501`
- [ ] Test complete workflow: capture → analyze → view results

Important (For Better Demo)

- [ ] Add visualizations:
  - **Bar chart:** Species counts (use st.bar_chart or plotly)
  - **Pie chart:** Species distribution
  - **Map:** Show sample locations (use folium or pydeck)
    - Color-code markers by biodiversity
    - Show popup with sample info
- [ ] Display original + preprocessed + segmented images side-by-side
- [ ] Add export button (download CSV/JSON)
- [ ] Show confidence scores alongside counts
- [ ] Add date/time filters for history view

Nice-to-Have

- [ ] Build React Native or Flutter mobile app
- [ ] Real-time dashboard with auto-refresh
- [ ] Advanced charts (time series, heatmaps)

- [ ] Dark mode
- [ ] Offline mode with local caching

---

9. INTEGRATION & TESTING

Critical (System Must Work End-to-End)

- [ ] **Integration Test 1:** Hardware → Capture
  - Trigger camera via Python script
  - LED turns on → capture → LED turns off
  - Image saved to `/images/raw/`
  - Verify image quality (not blurry, good lighting)
- [ ] **Integration Test 2:** Preprocessing Pipeline
  - Load raw image → preprocess → save processed image
  - Visually verify preprocessing improves image
- [ ] **Integration Test 3:** Detection & Classification
  - Load preprocessed image → detect organisms → classify each
  - Verify bounding boxes correctly identify organisms
  - Check classification predictions make sense
- [ ] **Integration Test 4:** Counting & Storage
  - Count organisms by species → save to database
  - Query database to verify data stored correctly
- [ ] **Integration Test 5:** API Workflow
  - Call `POST /capture` → verify image captured
  - Call `POST /analyze/{id}` → verify analysis completes
  - Call `GET /results/{id}` → verify results returned
- [ ] **Integration Test 6:** Frontend Display
  - Open Streamlit dashboard → trigger capture → view results
  - Verify all data displays correctly (images, counts, map)
- [ ] **End-to-End Demo Test:**
  - Place water sample on slide
  - Press capture button
  - Wait for analysis
  - View results on dashboard
  - Verify counts are reasonable

Important (For Robustness)

- [ ] Error handling:

    - What if camera fails? (retry mechanism)
    - What if model crashes? (log error, return gracefully)
    - What if GPS has no fix? (save with null coordinates)
- [ ] Performance testing:
    - Measure end-to-end latency (capture to results)
    - Target: <30 seconds for full workflow
- [ ] Test with various scenarios:
    - Single organism vs. multiple organisms
    - Different species
    - Low-quality images (blur, poor lighting)
    - Empty slide (no organisms)

Nice-to-Have

- [ ] Unit tests for each module
- [ ] CI/CD pipeline with GitHub Actions
- [ ] Load testing for API
- [ ] Automated testing scripts

---

10. DEMO PREPARATION

Critical (For Jury Presentation)

- [ ] **Prepare Demo Script:**
    1. Show physical hardware setup (Pi + camera + lens + LED)
    2. Place water sample on slide
    3. Click capture button on dashboard
    4. Explain pipeline as it runs (capture → preprocess → detect → classify → count)
    5. Show results: species counts, images, map
    6. Navigate to history and show past samples
- [ ] **Test Demo 5+ Times:** Ensure it works reliably
- [ ] **Backup Plan:**
    - Pre-captured images in case camera fails
    - Pre-loaded results in database
    - Screenshots/video of working system
- [ ] Create PPT presentation (10-15 slides):
    - Problem statement recap
    - Solution overview
    - Technical architecture diagram
    - Live demo (or video)

    - Results (accuracy metrics, sample outputs)
    - Impact and applications
    - Future scope
- [ ] Prepare to answer jury questions:
    - Model accuracy? (have numbers ready)
    - Why this hardware? (cost-effective, portable)
    - How handle overlapping organisms? (watershed segmentation)
    - Dataset size? (be honest, explain transfer learning)
    - Inference speed? (have benchmark ready)

Important

- [ ] Record backup demo video (2-3 min):
    - Show hardware
    - Walk through capture and analysis
    - Show results
- [ ] Prepare sample outputs:
    - 3-5 example analysis results
    - Screenshots of dashboard
    - CSV export samples
- [ ] Clean up code and documentation:
    - README with setup instructions
    - Requirements.txt with all dependencies
    - Quick start guide
- [ ] Test setup on fresh Pi (ensure installation works)

Nice-to-Have

- [ ] Create poster/banner for display
- [ ] Print system architecture diagram
- [ ] Prepare handout with project details
- [ ] Have QR code to GitHub repo

PRIORITY MATRIX

🔴 CRITICAL - Must Work (Week 1-2)

Hardware:

- Pi + camera working, can capture images

- LED backlight functional

ML:

- Trained model (even if only 70% accuracy)
- TFLite conversion working
- Inference running on Pi

Software:

- Basic preprocessing (normalize, resize)
- Simple detection (contour-based is fine)
- Counting organisms
- SQLite database storing results

Integration:

- FastAPI running on Pi with endpoints
- Streamlit dashboard showing results
- End-to-end workflow tested

🟡 IMPORTANT - Should Have (Week 3)

Hardware:

- GPS integration
- Calibration for size measurement

ML:

- Better accuracy (80%+)
- Segmentation for overlapping organisms
- Confidence filtering

Software:

- Illumination correction
- Background removal
- Map visualization
- Export to CSV

🟢 NICE-TO-HAVE - If Time (Week 3)

- Grad-CAM explainability
- Mobile app

- Cloud storage/sync
- Federated learning setup
- Advanced biodiversity metrics
- Auto-focus implementation
- Waterproof enclosure

RISKS & MITIGATIONS

Risk	Impact	Mitigation
Model accuracy too low	High	Use transfer learning, focus on 5-10 common species first, augment data heavily
Inference too slow on Pi	High	Quantize model aggressively, reduce input size, use MobileNet/EfficientNet-Lite
Camera image quality poor	High	Test lens early, optimize LED brightness, use CLAHE for illumination
Segmentation fails on overlaps	Medium	Fall back to simple detection, or manually separate for demo
GPS no fix indoors	Low	Use manual coordinates input, or demo with pre-saved GPS data
Dataset too small	Medium	Use transfer learning, heavy augmentation, limit classes to well-represented species
Integration issues	Medium	Define clear API contracts early, test components independently first
Demo hardware failure	High	Have backup pre-recorded video, backup images, test setup multiple times

---

DELIVERABLES CHECKLIST

Code

- [ ] GitHub repository with all code
- [ ] README with setup instructions
- [ ] Requirements.txt / environment.yml
- [ ] Organized folder structure

Models

- [ ] Trained model weights (.h5 or .pth)
- [ ] TFLite quantized model (.tflite)
- [ ] Model card (architecture, accuracy, training details)

Documentation

- [ ] System architecture diagram
- [ ] API documentation (auto-generated via FastAPI /docs)
- [ ] Hardware setup guide
- [ ] Quick start guide

Demo

- [ ] Working physical prototype
- [ ] Demo video (2-3 min)
- [ ] Presentation slides (10-15)
- [ ] Sample results (3-5 examples)

GETTING STARTED

Week 1 Focus:

1. Set up Raspberry Pi and camera
2. Download and organize dataset
3. Start model training on laptop/cloud
4. Create basic preprocessing pipeline

Week 2 Focus:

1. Convert model to TFLite and test on Pi
2. Implement detection and counting
3. Set up FastAPI backend
4. Create basic Streamlit dashboard

Week 3 Focus:

1. Integrate all components
2. Add GPS and size calibration
3. Improve UI with charts and map
4. End-to-end testing

Week 4 Focus: (Most prolly during hackathon)

1. Polish and bug fixes
2. Demo preparation and rehearsal
3. Documentation
4. Backup plans

Remember: The goal is a **working demo** that impresses the jury. Focus on getting the core pipeline working first, then add features. Quality > Quantity.