SANJIVANI 2.0 - Implementation Plan

Objective: Transform SANJIVANI from an SIH 2024 project into a portfolio-grade, production-ready AI crop disease detection platform with proper architectural separation, credible AI implementation, and real-world readiness.

User Review Required

> [!IMPORTANT] **Scope Reduction for Quality** The AI model will be reduced from 38 disease classes to **3 crops with 8-10 critical diseases**. This allows for:
>
> - Higher accuracy and confidence
> - Better validation and testing
> - More credible demo
> - Realistic performance metrics
>
> Expandable architecture allows adding more classes later.

> [!WARNING] **Breaking Changes**
>
> - API response format will change (structured output)
> - AI model will be retrained (different classes)
> - Database schema may need migration
> - Frontend components will require updates

> [!IMPORTANT] **Offline-First Architecture** Adding PWA support and offline capabilities will require:
>
> - Service worker implementation
> - IndexedDB for local storage
> - Background sync for queued scans
> - Potential increase in initial bundle size

Proposed Changes

Phase 1: Architecture Foundation

Backend Structure Refactoring

[MODIFY] backend/main.py

- Separate inference logic into dedicated module
- Add model performance endpoints (`/health`, `/metrics`)
- Implement structured response format
- Add request validation with Pydantic v2

[NEW] backend/ai/inference_engine.py

- Isolated AI inference logic
- Model loading and caching
- Preprocessing pipeline
- Benchmark tracking

[NEW] backend/knowledge/disease_knowledge.json

- Versioned disease database
- Structured treatment protocols
- Severity mappings
- Multilingual support structure

[NEW] backend/knowledge/knowledge_engine.py

- Query disease information
- Map predictions to treatments
- Version management
- Validation logic

Phase 2: AI System Redesign

Dataset & Training

[NEW] backend/ai/dataset_config.py

```
# Focused scope for credibility
CROPS = ["Tomato", "Potato", "Rice"]
DISEASES = {
    "Tomato": ["Early_Blight", "Late_Blight", "Leaf_Mold",
"Healthy"],
    "Potato": ["Early_Blight", "Late_Blight", "Healthy"],
    "Rice": ["Bacterial_Blight", "Brown_Spot", "Healthy"]
}
# Total: ~10 classes
```

[MODIFY] backend/train_model.py

- Use MobileNetV2 explicitly (document choice)
- Add comprehensive metrics tracking
- Generate confusion matrix
- Export dual formats (.h5 + .tflite)
- Save metadata (accuracy, training time, model size)

[NEW] backend/ai/model_evaluator.py

- Evaluate model performance
- Generate benchmark reports
- Track inference time
- Validate edge-readiness

[NEW] backend/models/model_metadata.json

```
{
  "version": "2.0.0",
  "architecture": "MobileNetV2",
  "input_size": [224, 224, 3],
  "classes": 10,
  "accuracy": 0.94,
  "precision": 0.93,
  "recall": 0.92,
  "f1_score": 0.925,
  "model_size_mb": 14.2,
  "avg_inference_ms": 45,
  "trained_date": "2025-12-26"
}
```

Phase 3: API Contracts v2

Structured Prediction Response

[NEW] backend/schemas/prediction.py

```
class PredictionResponse(BaseModel):
    crop: str
    disease: str
    confidence: float  # 0-1
    severity: Literal["Low", "Moderate", "High", "Critical"]
    explanation: str
    recommended_actions: RecommendedActions
    metadata: PredictionMetadata
class RecommendedActions(BaseModel):
    immediate: List[str]
    short_term: List[str]
    preventive: List[str]

class PredictionMetadata(BaseModel):
    model_version: str
    inference_time_ms: float
    visual_features: Optional[List[str]]
```

[NEW] API Endpoint: `GET /api/v2/model/metrics`

- Return model performance benchmarks
- Model version information
- System health status

Phase 4: Frontend Refactoring

Component Updates

[MODIFY] src/pages/Scan.tsx

- Handle new structured response
- Display confidence with visual bar
- Show severity badge
- Render action cards (immediate, short-term, preventive)
- Add inference time display
- Better error states

[DELETE] src/layouts/OSLayout.tsx

- Remove experimental UI or move to feature flag

[NEW] src/components/scan/ResultCard.tsx

- Structured display of prediction
- Confidence visualization
- Severity badge component
- Action checklist UI

[NEW] src/components/scan/ActionCard.tsx

- Display recommended actions
- Categorized by urgency
- Checkbox interaction for tracking

Phase 5: PWA & Offline Support

[NEW] public/service-worker.js

- Cache static assets
- Cache disease knowledge database
- Intercept API requests

[NEW] public/manifest.json

- PWA configuration
- Icons and splash screens
- Offline page

[NEW] src/lib/offline-queue.ts

- Queue scans when offline
- Background sync when online
- IndexedDB storage

[NEW] src/hooks/useOfflineSync.ts

- Monitor online/offline status
- Trigger sync when connection restored
- UI feedback for sync state

Phase 6: Documentation & Portfolio Positioning

[MODIFY] README.md

- Add system architecture diagram (mermaid)
- AI pipeline explanation with benchmarks
- Production-ready positioning
- Performance metrics table
- Edge-readiness statement

[NEW] docs/ARCHITECTURE.md

- Detailed system design
- Layer separation explanation
- Data flow diagrams

- Technology choices rationale

[NEW] docs/AI_PIPELINE.md

- Dataset composition
- Model architecture details
- Training methodology
- Evaluation metrics
- Benchmark results
- Edge deployment considerations

[NEW] docs/API.md

- Complete API reference
- Request/response examples
- Error codes
- Rate limiting
- Versioning strategy

[NEW] BENCHMARKS.md

- Model performance metrics
- Inference time benchmarks
- Sample predictions with confidence
- Confusion matrix visualization
- Comparison with baseline

Verification Plan

Automated Tests

Backend Tests

```
# Model evaluation
pytest backend/tests/test_inference_engine.py
# API contract validation
pytest backend/tests/test_api_v2.py
# Knowledge engine
pytest backend/tests/test_knowledge_engine.py
```

Frontend Tests

```
# Component rendering with new response
npm test -- src/components/scan/
# Offline queue functionality
npm test -- src/lib/offline-queue.test.ts
# PWA functionality
npm run test:e2e -- offline.spec.ts
```

Manual Verification

1. **AI Pipeline Validation**

- Train model with focused dataset
- Verify metrics meet threshold (>90% accuracy)
- Test inference time (<100ms)
- Validate .tflite export

2. **Offline Functionality**
    - Disable network in DevTools
    - Queue multiple scans
    - Re-enable network
    - Verify background sync
3. **UI/UX Testing**
    - Test new result card layout
    - Verify action cards display correctly
    - Check severity badges
    - Validate confidence visualization
4. **Documentation Review**
    - Architecture diagram clarity
    - Benchmark table accuracy
    - API examples work correctly
    - README positioning is compelling

Portfolio Readiness Checklist

- [ ] System architecture diagram in README
- [ ] AI pipeline fully documented with benchmarks
- [ ] Sample predictions with screenshots
- [ ] Performance metrics table (accuracy, inference time, model size)
- [ ] Edge-readiness clearly stated
- [ ] Docker deployment working
- [ ] Code is well-commented and professional
- [ ] No "demo" or "tutorial" vibes in presentation

Timeline Estimate

Phase	Estimated Time	Priority
Phase 1: Architecture Planning	1 day	Critical
Phase 2: AI System Redesign	3-4 days	Critical
Phase 3: Backend Refactoring	2 days	High
Phase 4: Frontend Updates	2 days	High
Phase 5: PWA Implementation	2 days	Medium
Phase 6: Documentation	1-2 days	High
Testing & Validation	2 days	Critical
Total	**13-17 days**	-

Success Criteria

·**Technical Excellence**

- Model accuracy >90% on focused dataset
- Inference time <100ms
- Clear architectural separation
- No logic mixing between layers

·**Professional Presentation**

- Architecture diagram worthy of system design interview
- Benchmarks that prove competency
- Documentation that shows engineering maturity
- Code quality that passes senior engineer review

·**Real-World Readiness**

- Offline functionality works
- PWA installable
- Docker deployment smooth
- Production-ready error handling

·**Portfolio Impact**

- "Independently rebuilt AI system with production-grade architecture"
- Shows understanding of edge AI
- Demonstrates full-stack + ML competency
- Credible, not demo-ware

Next Steps

1 **Review this plan** - Confirm scope, timeline, breaking changes
2 **Approve architecture** - Ensure layer separation makes sense
3 **Begin Phase 1** - Start with dataset narrowing and AI pipeline
4 **Iterative implementation** - Build, test, document in cycles
5 **Final validation** - Ensure portfolio-readiness before considering complete