KULATHUNGA K.A.K.M — Lead AI Architect (Anomaly Detection & RCA)

Fully aligned with the *actual* current SmartOps architecture, the orchestrator capabilities Peiris implemented, the policy engine design, and the real telemetry stack already deployed.

✅ KULATHUNGA K.A.K.M — COMPREHENSIVE IMPLEMENTATION PLAN

(Clean, restructured, realistic, production-grade)

SECTION 1 — ROLE OVERVIEW & OBJECTIVE

Kulathunga owns **all AI intelligence inside SmartOps**, specifically:

Two core AI agents:

1. **Anomaly Detection Agent**
   - Multivariate, real-time inference
   - Maps telemetry → anomaly type, score, severity
   - Sends structured signals into orchestrator
2. **RCA (Root Cause Analysis) Agent**
   - Inspects anomalies + dependency graphs
   - Produces top-ranked root causes with confidence
   - Produces human-readable explanations

Why his role is critical:

Every autonomous recovery in SmartOps starts with **AI signals** sent to:

```
/v1/signals/anomaly
/v1/signals/rca
```

…which power the closed-loop orchestrator created by Peiris.

Kulathunga must ensure:

✓ AI signals are accurate
✓ RCA is explainable
✓ Real-time inference is stable
✓ Agents integrate smoothly with Orchestrator + Policy Engine
✓ Models learn from successful/failed actions

Workload Share: **25–27%**
Focus: **AI intelligence, ML pipelines, diagnostics, explainability, evaluation**

SECTION 2 — SYSTEM CONTEXT (REAL PROJECT STATE)

SmartOps currently has:

- Peiris's orchestrator with:
  - scale/restart/patch
  - closed-loop manager
  - verification engine
  - complete Kubernetes API integration
- ERP simulator generating chaos modes
- Prometheus Operator, Loki, OTel Collector, Tempo
- Orchestrator already validating anomaly + RCA formats
- Policy Engine design underway

Kulathunga's AI agents plug in at the earliest stage of the automation loop:

```
Telemetry → AI Detect → AI RCA → Policy Engine → Orchestrator → K8s
```

SECTION 3 — HIGH-LEVEL DELIVERABLES

Kulathunga must deliver:

1. **Anomaly Detection Agent**
2. **RCA Diagnostic Agent**
3. **Feature store & telemetry ingestion pipeline**
4. **Model training, tuning & evaluation notebooks**
5. **Chaos validation dataset**
6. **Prometheus + OTel metrics for AI agents**
7. **REST APIs for real-time inference**
8. **Feedback-driven retraining system**
9. **Full documentation bundle for AI side**

Everything below is structured to produce those deliverables reliably.

SECTION 4 — PHASED IMPLEMENTATION PLAN (BEST VERSION)

PHASE 1 — AI Repository & Environment Setup (Week 1–2)

Goals

Create a clean and scalable AI workspace.

Tasks

- Create folder structure:

```
apps/agent-detect/
apps/agent-diagnose/
    src/
    models/
    notebooks/
    pipelines/
```

- Prepare runtime:
  - Python 3.10/3.11
  - PyTorch/TensorFlow (choose one early)
  - scikit-learn
  - Pandas
  - Uvicorn/FastAPI
- Create:
  - `requirements.txt`
  - Dockerfile
  - `Makefile` for train/test/infer
  - CI pipeline for lint + unit tests

Deliverables

✓ Repo builds successfully
✓ Unit test harness
✓ CI runs on PRs

PHASE 2 — Telemetry Ingestion & Feature Engineering (Week 2–4)

Telemetry Sources

- Prometheus metrics
- Loki logs (JSON)
- Tempo traces (optional later)
- ERP Simulator chaos modes

Feature Store Requirements

- Extract fixed-length multivariate windows (30s, 60s, 120s)
- Features:
  - CPU usage
  - Memory usage
  - Error rate
  - Latency and p95/p99
  - Jitter
  - Request throughput
  - Chaos mode indicators
- Store as parquet in:

```
/data/processed/
```

Tasks

- Build Prometheus query pipeline (Python HTTP client)
- Build log sampler from Loki
- Normalize time windows
- Label “normal” vs “faulted” windows using ERP chaos

Deliverables

✓ Dataset ready
✓ Feature schema documented
✓ Telemetry aligned with orchestrator signal format

PHASE 3 — Anomaly Detection Model (Week 4–7)

Model Requirements

- Real-time inference (<3–5 seconds)

- Accurate: F1 ≥ 0.85
- Low false positives

Architecture

- LSTM Autoencoder
- Optional: GRU encoder for low latency
- Adaptive thresholding:
  - Rolling mean
  - MAD
  - z-score
  - Combined threshold logic

Outputs

Agent must return JSON:

```
{
  "windowId": "...",
  "service": "erp-simulator",
  "isAnomaly": true/false,
  "score": 0.87,
  "type": "latency" | "resource" | "error" | "cpu" | ...
}
```

Deliverables

✓ Model saved under `/models/`
✓ Inference engine with ~5s per request
✓ Chaos dataset validation

PHASE 4 — RCA (Root Cause Analysis) Engine (Week 6–9)

Approach (Realistic + Lightweight)

- Use dependency graph:
  - ERP → DB → Cache → Worker → API
- Compute deltas in metrics:
  - CPU spike
  - Memory leak
  - Error burst
  - Latency jitter
- Assign probabilities
- Provide explainable output

Output Format

```
{
  "windowId": "...",
  "service": "erp-simulator",
  "rankedCauses": [
     {"svc": "erp-simulator", "cause": "memory_leak", "probability": 0.92}
  ],
  "confidence": 0.88
}
```

Deliverables

✓ RCA accuracy ≥ 80%
✓ Explainability module (rules + metrics-based)
✓ Works on dev cluster chaos tests

PHASE 5 — Model Evaluation & Benchmarking (Week 8–10)

Chaos-based evaluation

Use ERP Simulator:

- `/chaos/memory-leak/enable`
- `/chaos/cpu-spike/enable`
- `/chaos/latency-jitter/enable`
- `/chaos/error-burst/enable`

KPIs

- F1 score
- False positive rate
- Latency per inference
- RCA confidence

Deliverables

✓ `/docs/reports/model_eval.pdf`
✓ Confusion matrices
✓ Comparison between classical vs deep models

PHASE 6 — API Integration & Service Deployment (Week 9–11)

Expose APIs

Two microservices:

1. agent-detect
`POST /v1/anomaly/predict`

2. agent-diagnose
`POST /v1/rca/diagnose`

Requirements

- Respond within <5 seconds
- Add Prometheus metrics:
  - `anomaly_inference_latency_seconds`
  - `anomaly_requests_total`
  - `rca_inference_latency_seconds`

Integration

- Send anomaly → `/v1/signals/anomaly`
- Send RCA → `/v1/signals/rca`

Deliverables

✓ Both AI services callable inside cluster
✓ Orchestrator integration validated

PHASE 7 — Continuous Learning Loop (Week 11–13)

Data Sources

Use orchestrator outputs:

- Verification success
- Action type
- Closed-loop duration
- Deployment status

Tasks

- Build feedback DB
- Tag windows with "success/failure"
- Retrain anomaly & RCA models periodically
- Track drift

Deliverables

✓ Retraining workflow
✓ Drift detection dashboard
✓ Improved MTTR after learning

PHASE 8 — Security, Governance & Deployment (Week 13–15)

RBAC

AI agents should only be allowed to:

- Read telemetry
- NOT directly call Kubernetes
- NOT modify cluster resources

All orchestration must go through Peiris's orchestrator.

Deployment Requirements

- Helm subchart
- Resource limits
- Readiness & liveness probes
- Model artifacts stored safely

Deliverables

✓ RBAC YAML
✓ Helm chart for AI services
✓ GitHub Actions deploy pipeline

PHASE 9 — Documentation & Viva Preparedness (Week 15–16)

Documents to Produce

- ai_agent_architecture.md
- model_training_guide.md
- model_eval.pdf
- api_reference.md
- telemetry_feature_engineering.md
- rca_design.md

Acceptance

✓ All AI docs merged to `/docs/ai/`
✓ Review-ready for examiner

SECTION 5 — KPIs (REALISTIC TARGETS)

Metric	Target	Validation
F1 Score	≥ 0.85	Evaluation notebooks
False Positive Rate	≤ 2%	Chaos dataset
RCA Confidence	≥ 80%	RCA logs
Inference Latency	≤ 5 sec	Prometheus metrics
Auto-Heal MTTR Improvement	≥ 30–40%	Grafana panels
Model Drift Detection	< 24h	Drift monitor

SECTION 6 — COLLABORATION INTERFACES

With Peiris (Orchestrator)

- Validate anomaly & RCA JSON schemas
- Provide signals that map directly to closed-loop actions
- Ensure inference latency matches closed-loop rate

With Gunarathne (Policy Engine)

- Provide anomaly types and RCA causes mapped to DSL keywords
- Provide confidence thresholds
- Respond to policy-engine feedback loop

With Dissanayake (Infra/DevOps)

- Deploy AI agents
- Configure Grafana dashboards
- Enable data scraping from Prometheus

SECTION 7 — FINAL DELIVERABLES

1. **Anomaly Detection Service** (`/apps/agent-detect/`)
2. **RCA Diagnostic Service** (`/apps/agent-diagnose/`)
3. **AI Models & Registry** (`/models/`)
4. **Feature Store & ETL Pipelines** (`/data/processed/`)
5. **Evaluation Reports** (`/docs/reports/`)
6. **AI Documentation Set** (`/docs/ai/`)
7. **Helm Charts & CI Deployments** (`platform/helm/smartops/agent-*`)

✅ SUMMARY — Kulathunga's Role in SmartOps

Kulathunga builds the **intelligence** that allows SmartOps to:

- Detect failures
- Diagnose root causes
- Trigger autonomous remediation
- Learn from actions
- Improve MTTR continuously

He is:

- **Accountable** for all ML components
- **Responsible** for anomaly detection, RCA, and feedback systems
- **Consulted** for policy and orchestrator integrations
- **Informed** about deployment, telemetry, and chaos validation

His AI modules form the **foundation** of predictive self-healing in the entire SmartOps ecosystem.