1) Predictive Network Fault Detection & Automated Dispatch

Predictive Network Fault Detection & Automated Field Dispatch

Why this is the best fit:

- Highly relevant to a telecom company (SLT-Mobitel) — directly touches network uptime, customer experience and ops cost.
- Combines AI (time-series anomaly detection / predictive models) + automation (auto-ticketing, technician dispatching) — scores well on AI integration.
- Clear measurable impact (reduced downtime, faster MTTR, fewer manual tickets) — great for presentation & metrics.

Alternate strong ideas (if you prefer):

- **Automated Customer Query Triage & Resolution** (LLM-driven helpdesk + auto-resolve common billing queries).
- **Smart Field Technician Scheduling** (optimize assignments with constraints + predictive maintenance signals).

2) Solution overview

Goal: Detect anomalies in network telemetry (alarms, throughput, error rates) early and automatically create prioritized incident tickets and dispatch nearest qualified technicians or trigger automated remedial actions.

Main capabilities

- Ingest network telemetry (SNMP/metric stream, alarms, log lines)
- Real-time anomaly detection (statistical + ML)
- Root-cause suggestion (correlate alarms, probable component)
- Auto-ticket creation with priority & suggested SLA
- Auto-dispatch / schedule technicians (matching skills, location, shift)
- Dashboard for operators + Alerting (SMS / Slack / Email)
- Feedback loop: ticket outcomes used to retrain/improve model

Tools & tech (recommended for prototype)

- Data ingestion: MQTT / Kafka / HTTP webhook
- Orchestration: Node-RED for prototype automation flows (ingest → detect → ticket)

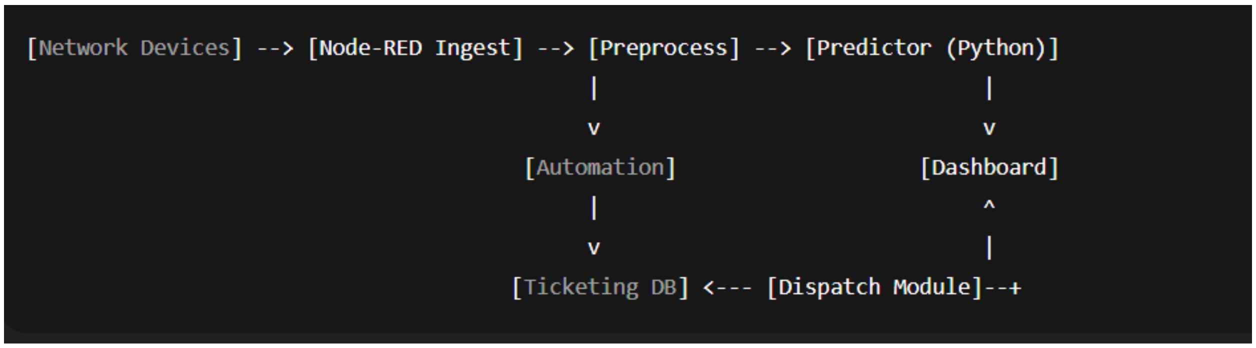
- ML: Python (scikit-learn, xgboost or simple LSTM/Prophet) for anomaly prediction
- Storage: time-series DB (InfluxDB) or simple CSV for prototype
- Ticketing API: simulate or integrate with ServiceNow/Zendesk (or just create DB entry)
- Dashboard: Streamlit or simple HTML + charting

3) Architecture (textual + simple diagram)

Flow:

1. **Network Devices** → send metrics/logs →
2. **Ingestion Layer** (MQTT/HTTP) → Node-RED captures messages →
3. **Preprocessing** (aggregate, rolling windows) → Python microservice (predictor) →
4. **Anomaly Detector** returns score & probable root cause →
    - if score > threshold → **Automation Engine (Node-RED)** creates ticket & calculates priority
    - else log as normal
5. **Dispatch Module** (matches tech by location/skill/shift) → sends assignment via SMS/email
6. **Dashboard** shows incidents, model confidence, historical trends
7. **Feedback**: technician closes ticket → label saved for model retraining

ASCII diagram:



```
[Network Devices] --> [Node-RED Ingest] --> [Preprocess] --> [Predictor (Python)]
                                            |                              |
                                            v                              v
                                       [Automation]                   [Dashboard]
                                            |                              ^
                                            v                              |
                                      [Ticketing DB] <--- [Dispatch Module]--+
```

4) Mapping to Judging Criteria

- **Creativity & Innovation (30%)**
  Use hybrid detection: combine simple statistical thresholds + ML model + correlated alarm graph to reduce false positives. Add intelligent auto-dispatch with travel-time optimization.
- **Relevance to SLT-MOBITEL (25%)**
  Directly improves network reliability, reduces customer complaints, translates to cost savings.

- **AI Integration & Innovation (25%)**
  Real-time anomaly prediction + auto root-cause inference + feedback loop for continual learning.
- **Presentation & Clarity (20%)**
  Provide crisp demo: live simulated telemetry → alert → ticket creation → dispatch. Use visuals (timeline, before/after KPIs).

5) Demo plan (quick & persuasive)

Minimum Viable Demo (what to show at submission/presentation):

1. A short video (screen recording) or live demo showing:
   - Synthetic streaming of network metrics (CSV → HTTP POSTs or MQTT)
   - Node-RED flow receiving and forwarding data
   - Python predictor flags an anomaly
   - Auto ticket appears in dashboard (with priority, root cause)
   - Dispatch plan shows technician assignment (map or list)
2. Slides explaining architecture, data, model, impact & cost/benefit
3. Code repo with readme + instructions to run locally

6) Node-RED flow outline (for prototype)

Nodes to create:

- `MQTT in` or `http in` — ingest simulated telemetry
- `function` — transform & compute rolling window metrics (avg, std)
- `http request` — call Python predictor API with preprocessed window
- `switch` — check predictor response (anomaly? score)
- `function` — create ticket payload (title, device, priority, suggested root cause)
- `http request` — call ticketing endpoint (or write to DB/JSON file)
- `function` — compute nearest technician (simple geodistance using lat/lon stored in a small JSON)
- `notification` — send SMS/email stub (or console log)
- `dashboard` nodes — show live metrics & alerts

(If you want, I can produce a ready `.json` Node-RED flow file you can import.)

7) Python prototype (runnable plan + sample code skeleton)

Prototype goals: simulate telemetry, detect anomaly using a simple models (e.g., rolling z-score or an IsolationForest), return anomaly score.

Below is a minimal Python script (pseudocode / runnable) to act as the predictor service and a small simulator.

```
# Save as predictor_service.py
from flask import Flask, request, jsonify
import numpy as np
from sklearn.ensemble import IsolationForest
import joblib
import pandas as pd

app = Flask(__name__)

# For prototype: create a simple IF model trained on synthetic normal data
def train_model():
    # Create normal data: metric values around 100 with small noise
    X = np.random.normal(loc=100, scale=3, size=(1000,1))
    model = IsolationForest(contamination=0.01, random_state=42)
    model.fit(X)
    joblib.dump(model, 'if_model.joblib')
    return model

try:
    model = joblib.load('if_model.joblib')
except:
    model = train_model()

@app.route('/predict', methods=['POST'])
def predict():
    payload = request.json  # expect {'metric_window':[...], 'device': 'dev1'}
    window = np.array(payload.get('metric_window', [])).reshape(-1,1)
    # For proto: use the last value as feature or mean
    feat = np.mean(window).reshape(1, -1)
    score = model.decision_function(feat)[0]  # higher -> normal
    pred = model.predict(feat)[0]  # -1 anomaly, 1 normal
    is_anomaly = (pred == -1)
    return jsonify({'is_anomaly': bool(is_anomaly), 'score': float(score),
'value': float(feat)})
```

```
if __name__ == '__main__':
    app.run(port=5001, debug=True)
```

Simulator (send synthetic telemetry):

```
# Save as simulator.py
import requests, time, random

URL = 'http://localhost:5001/predict'

for i in range(200):

    # simulate normal traffic, inject anomaly around iteration 80

    if 80 <= i < 90:

        window = [random.gauss(140,5) for _ in range(10)]

    else:

        window = [random.gauss(100,2) for _ in range(10)]

    payload = {'metric_window': window, 'device': 'BS-123'}

    r = requests.post(URL, json=payload).json()

    print(i, r)

    time.sleep(0.5)
```

This will demonstrate the predictor marking the injected interval as anomalous. Node-RED can call the `predict` endpoint, receive anomaly → create ticket.

If you want, I can produce a more advanced model (LSTM or Prophet) and training scripts.

8) Presentation slide deck (slide-by-slide content)

Slide 1 — Title

- **AI Horizons — Challenge 4: Auto Masters**
- Project: **Predictive Network Fault Detection & Automated Dispatch**

- Team / Your Name

Slide 2 — Problem Statement

- Manual monitoring of network alarms leads to delayed detection & slow dispatch.
- High MTTR, frequent repeats, manual ticket triage consumes ops time.

Slide 3 — Impact (metrics to improve)

- Mean Time To Repair (MTTR)
- Number of false alarms
- Customer downtime minutes
- Tickets auto-handled %

Slide 4 — Proposed Solution (one-liner)

- Real-time anomaly detection from telemetry + auto-ticketing & intelligent dispatch.

Slide 5 — Architecture (include simple diagram)

- Ingestion → Predictor → Automation → Dispatch → Dashboard

Slide 6 — AI Components

- Anomaly detector (IsolationForest / LSTM)
- Root cause inference (alarm correlation)
- Feedback loop for retraining

Slide 7 — Demo walkthrough

- (Screenshots/video) show telemetry → anomaly → ticket → dispatch

Slide 8 — Evaluation vs Judging Criteria

- Creativity: hybrid detection + automated dispatch
- Relevance: telecom ops use-case
- AI integration: model + root cause + learning
- Presentation clarity: demo + slides + repo

Slide 9 — Implementation & Tools

- Node-RED, Python (Flask), IsolationForest, Streamlit (dashboard), DB (sqlite/Influx)

Slide 10 — Roadmap & Next Steps

- Integrate with real telemetry, scale predictor, optimize dispatch algorithm, pilot deployment

Slide 11 — Ask / Closing

- Prize, contact, demo link, github repo