Quasar x AI 2026

ML Wing

==========

1.Problem Statement: Self-Healing Web Application

==========

Background:

Imagine a website that watches users struggle and fixes itself automatically. No waiting for developers — the app learns from problems and resolves them using intelligent automation and GenAI.

Challenge:

Build a web application that:

1. Tracks user struggles (e.g., clicks, errors, slow pages)

2. Classifies problems using simple rules

3. Automatically fixes itself using GenAI + file operations

4. Validates changes before applying them

5. Learns from history to avoid repeating mistakes

Part 1: User Behavior Tracking

Capture signals like:

- User frustrations: rage clicks, dead clicks, quick back button, repeated form submissions

- Technical events: JS errors, slow API responses, slow page loads

- User journey data: page sequence, scroll depth, time spent

Deliverable: A simple dashboard showing real-time events.

Part 2: Problem Classification

Use simple rules to classify issues:

UI Issues: confusing buttons, unclear clickable areas

Performance: slow page loads, slow API, memory issues

Functionality Bugs: form validation errors, JS errors, broken endpoints

Optional: Use basic clustering (K-Means) for anomaly detection.

Part 3: Auto-Code Modification

Automatically fix issues in the code:

Steps:

1. Identify the problematic code (based on user data)

2. Generate fix using GenAI (e.g., GPT-4)

3. Replace the problematic code in the source file

4. Run tests and validate before deploying

5. Rollback if tests fail

Safety rules:

- Backup files before modification

- Limit changes per file

- Don’t modify critical code (auth/payment)

Part 4: Web Application Demo

Create a Task Management App with intentional issues:

- Slow-loading task list

- Confusing buttons

- Form validation problems

- Slow API

- Missing error handling

Dashboard should display:

- Real-time problems

- Detected problem categories

- Fix status (success/failure)

- History of changes

Stack Suggestions:

Frontend: React or Vue.js

Backend: Python FastAPI or Node.js Express

Database: PostgreSQL or MongoDB

Cache: Redis

GenAI: OpenAI GPT-4 or Anthropic Claude

Deliverables:

---

1. Live Demo: Task app + self-healing system + dashboard

2. Code Repository: Source code + instructions

3. Technical Document: Architecture + approach + results

4. Demo Video: Show problems and automatic fixes

Evaluation Criteria:

- Automation quality: detecting and fixing problems

- System intelligence: accuracy of classification and fix success rate

- Full-stack implementation: frontend tracking, backend API, dashboard

- Innovation & UX: usability, creativity

==========

==========

PS - 2

AI-POWERED NEWS INTELLIGENCE PLATFORM

==========

PROBLEM:

Build an ML system to:

✓ Detect fake/misleading news

✓ Analyze sentiment & emotions

✓ Extract & verify factual claims

✓ Generate trustworthy summaries with citations

Goal: Automate news credibility assessment.

==========

PIPELINE:

News Aggregation → Fake News Detection → Sentiment Analysis →

Fact-Checking → Summarization → Trust Score Assignment

==========

PART 1: FAKE NEWS DETECTION (30 pts)

Objective: Classify news as Real/Fake

Features: text length, punctuation, ALL CAPS ratio, sentiment, entities

Hint: Look for patterns common in sensational content

Models: TF-IDF + Logistic Regression / Random Forest (baseline), BERT/DistilBERT (advanced)

Datasets: LIAR, Fake News Detection

Metrics: Accuracy, Precision/Recall/F1, False Negative Rate

Output:

```
{
  "label": "Real" / "Fake",
  "confidence": 0.91,
  "source_credibility": 82
}
```

==========

PART 2: SENTIMENT & EMOTION ANALYSIS (25 pts)

Objective: Multi-label emotion + entity-level sentiment

Emotions: Joy, Sadness, Anger, Fear, Surprise, Disgust, Trust

Hint: Consider words around key entities and emotional cues

Models: Word embeddings + BiLSTM/CNN, optional DistilBERT

Datasets: GoEmotions, Sentiment140

Metrics: Macro/Micro F1, Hamming Loss

Output:

```
{
  "overall_sentiment": "Negative (62%)",
  "emotions": {"concern":0.45,"skepticism":0.28},
  "entities":[{"name":"Jerome Powell","sentiment":"negative"}]
}
```

==========

PART 3: FACT-CHECKING (30 pts)

Steps:

1. Claim Extraction: binary classifier (numbers, dates, entities)
   Hint: Statements with verifiable numbers often matter most
2. Evidence Retrieval: TF-IDF or embedding similarity
   Hint: Similar text in trusted sources usually confirms claims

3. Verification: NLI (SUPPORTED / REFUTED / NOT_ENOUGH_INFO)

Hint: Check if evidence agrees, contradicts, or is inconclusive

Datasets: FEVER, LIAR, COVID-19 Fake News

Metrics: Claim F1, Verification accuracy, Evidence recall

Output:

{
"claims":[
{"text":"GDP grew 5.2%","verification":"SUPPORTED","confidence":0.87},
{"text":"All economists agree","verification":"REFUTED","confidence":0.92}
]
}

==========

PART 4: CONTENT GENERATION (15 pts)

Summary: Extractive/Abstractive (TextRank, BART/T5)

Hint: Include only verified facts and highlight uncertain ones

Citations & Trust: ✅/⚠/❌

Trust Score: combine fake news + fact-check + source credibility

Output Example:

[Trust Score: 85/100]

Summary: "Fed raised rates 0.75% ✅ [Source: Fed.gov]. Claims of unanimous support ❌."

==========

IMPLEMENTATION ROADMAP

Week 1-2: Data prep & baseline models

Week 3-4: Advanced models (BERT, BiLSTM/DistilBERT)

Week 5: Fact-checking module

Week 6: Integration & dashboard (Flask/Streamlit)

==========

EVAL METRICS

Fake News: Accuracy, F1, ROC-AUC, False Negatives

Sentiment: Multi-label F1, Hamming Loss

Fact-Checking: Claim F1, Verification accuracy, Evidence recall

Content: ROUGE, human readability/factuality

==========

TECH STACK

ML/NLP: transformers, torch/tf, scikit-learn, xgboost

Text: spacy, nltk, sentence-transformers

Data: pandas, numpy

Web: beautifulsoup4, requests

Viz & Deployment: matplotlib, seaborn, plotly, flask, streamlit

==========

BONUS FEATURES (Optional)

☐ Multi-source comparison

☐ Bias detection (political lean)

☐ Temporal sentiment tracking

☐ Explainable AI (LIME/SHAP)

☐ Multi-lingual support

==========

GOOD LUCK!

==========

PS -3

==========

ADAPTIVE RANSOMWARE DETECTION SYSTEM

ML & Security Operations Challenge

==========

CHALLENGE OVERVIEW

Build an AI-powered system that detects ransomware attacks in a network.

Focus on three signals: malicious URLs, malware files, and unusual network activity.

Combine these signals to detect full ransomware attack chains.

RANSOMWARE KILL CHAIN

Stage 1: Initial Access → Malicious URL clicked, malware downloaded

Stage 2: Persistence & Lateral Movement → Malware installs backdoor, scans network, attempts logins

Stage 3: Execution → Ransomware encrypts files across the network

ML TASKS (100 POINTS TOTAL)

PART A: MALICIOUS URL DETECTION (30 points)

Task: Detect phishing/malicious URLs

Requirements:

✓ Classifier to detect malicious vs benign URLs

✓ Hint: Use features like URL length, digit/special char count, presence of keywords; models like Random Forest or Logistic Regression work well

✓ Evaluate using accuracy, precision, recall

Dataset: [Malicious URLs Dataset](https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset)

PART B: MALWARE FILE CLASSIFICATION (40 points)

Task: Detect ransomware files

Requirements:

✓ Use static file features (PE header info, imports/exports, file size, entropy)

✓ Hint: Train a classifier using Random Forest, XGBoost, or a small neural network; handle class imbalance using weights or oversampling

✓ Evaluate using accuracy and confusion matrix

Dataset: [EMBER 2018 Malware Dataset](https://www.kaggle.com/datasets/vivekanandabharupati/ember2018)

PART C: NETWORK ANOMALY DETECTION (20 points)

Task: Identify lateral movement or suspicious logins

Requirements:

✓ Highlight unusual connections or repeated failed login attempts

✓ Hint: Use flow features (source IP, destination port, connection count); anomaly detection like Isolation Forest or One-Class SVM can flag deviations

✓ Evaluate using precision/recall

Dataset: [CICIDS 2017 Network Flow Dataset](https://www.kaggle.com/datasets/bertvankeulen/cicids-2017)

PART D: ATTACK SEQUENCE CORRELATION (10 points)

Task: Combine alerts from Parts A, B, C to detect ransomware attacks

Requirements:

✓ Predict if a ransomware attack is ongoing based on alert sequence

✓ Hint: Simple rule-based correlation works; for more advanced, encode alert sequence and train a small LSTM/GRU to predict multi-stage attacks

Example:

Input: [Malicious URL detected, malware execution, repeated SSH failures]

Output: Ransomware attack detected

EVALUATION METRICS

- Accuracy & Precision/Recall per stage
- Ability to detect multi-stage attacks
- Reduction of false negatives
- Optional: Confusion matrix visualization

SUBMISSION REQUIREMENTS

---

1. CODE
   ✓ Jupyter notebooks or Python scripts
   ✓ Data preprocessing, feature extraction
   ✓ Model training & evaluation
   ✓ Requirements.txt
2. REPORT
   ✓ Approach explanation
   ✓ Dataset description
   ✓ Model results & challenges
3. DEMO
   ✓ Show detection on test examples
   ✓ Show alert correlation (URL → malware → network)

DATASETS

---

1. EMBER 2018 - Malware Detection
   https://www.kaggle.com/datasets/vivekanandabharupati/ember2018
2. CICIDS 2017 - Network Flow Data

https://www.kaggle.com/datasets/bertvankeulen/cicids-2017

3. Malicious URLs Dataset

https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

---

GOOD LUCK!

==========

PS - 4

==========

AUTONOMOUS AI LEGAL ADVISOR & COURTROOM ARGUMENTATION SYSTEM

(Indian Justice System)

==========

PROBLEM STATEMENT:

Build an AI Legal Advisor that analyzes disputes and generates courtroom-ready

legal arguments for the Indian Justice System using Knowledge Graph + Hybrid Search.

CORE CAPABILITIES:

---

1. MULTIMODAL INPUT

- Text/Audio input (legal issues)

- Document ingestion (PDFs, scanned papers)

Libraries: Whisper, pydub, PyPDF2, pdfplumber, Tesseract, PaddleOCR

2. KNOWLEDGE GRAPH ARCHITECTURE

Graph Schema:

NODES: Constitutional Articles, Statutes, Case Precedents, Legal Arguments,

Document Entities, Procedures

EDGES: CITES, OVERRULES, INTERPRETS, SUPPORTS, CONTRADICTS, REQUIRES

Libraries: Neo4j, py2neo, NetworkX, igraph

3. HYBRID SEARCH (60% Accuracy + 40% Latency)

Tier 1 - Local Graph Search (<100ms)

→ 2-3 hop neighborhood traversal

Tier 2 - Global Graph Search (<1000ms)

→ PageRank + Community Detection

Tier 3 - Vector Search (<500ms)

→ Semantic similarity matching

Libraries: Neo4j Cypher, NetworkX.pagerank(), FAISS, Qdrant,

sentence-transformers, ChromaDB

4. REAL-TIME JUDICIAL SCRAPING

- Scrape SC/HC/District Court archives

- Embed judgments in vector DB

Libraries: BeautifulSoup4, Scrapy, Selenium, schedule

5. DOCUMENT INTELLIGENCE (Land/Inheritance/Divorce)

- OCR extraction & validation

- Identify gaps, contradictions, inconsistencies

- Map document relationships in graph

Libraries: Tesseract, PaddleOCR, spaCy, regex, pandas

6. COURTROOM ARGUMENT GENERATION

Must ARGUE, Not Summarize:

✗ "There may be property issues"

✓ "Under Section 54, Transfer of Property Act, 1882:

1. Mutation entry INVALID (lacks Tehsildar signature)

2. Counter-Argument: Adverse possession claim

3. Rebuttal: Tax receipts prove continued ownership

Prayer: Declaration of ownership"

Generate:

- Claims with statute citations

- Counter-arguments

- Rebuttals with precedents

- Plaintiff/Respondent perspectives

Libraries: LangChain, llama-index, Anthropic Claude API, OpenAI GPT-4

7. MULTILINGUAL SUPPORT

- All constitutional languages of India

Libraries: IndicTrans2, mBART, AI4Bharat models, Google Translate API

8. EXPLAINABILITY

- Every argument mapped to: Statute + Case Law + Constitutional Article

- Transparent reasoning chains

Libraries: LangChain callbacks, custom citation extraction (regex)

==========

ETHICAL DISCLAIMER:

"This system provides AI-assisted legal reasoning and argument drafts.

Final legal advocacy rests with qualified legal professionals under

Advocates Act, 1961. Not a substitute for lawyers or judges."

==========

SD Wing

1. **Smart Attendance System (AI-Based)**

Problem Statement

Traditional attendance systems are time-consuming, prone to proxy attendance, and require manual effort from instructors. There is a need for a secure, automated system that can reliably mark attendance while minimizing human intervention.

Objective

Design and develop an AI-powered attendance system that automatically marks attendance using face recognition or voice verification, ensuring accuracy and preventing fraudulent entries.

Key Features:

Student registration with biometric data

AI-based identity verification (face or voice)

Real-time attendance marking

Attendance reports and analytics for faculty

AI Integration

Face recognition using computer vision models

Voice authentication using speech embeddings

Expected Outcome

A scalable attendance solution that reduces manual workload, improves accuracy, and prevents proxy attendance.

2. AI Chatbot for College Queries

Problem Statement

Students often face difficulty accessing accurate information related to college procedures such as exams, timetables, fees, and facilities due to limited office hours and delayed responses.

Objective

Build an AI chatbot that answers college-related queries in real time using natural language understanding.

Key Features

Natural language query handling

FAQ and document-based responses

Context-aware follow-up questions

Admin panel to update information

AI Integration

NLP for intent detection and response generation

Embedding-based search for accurate answers

Expected Outcome

A 24×7 intelligent assistant that improves student experience and reduces administrative workload.

3. **Hostel Complaint Tracking System**

Problem Statement

Hostel residents often face delays in resolving maintenance and facility issues due to manual complaint processes and lack of transparency.

Objective

Create a digital platform for logging, tracking, and resolving hostel complaints efficiently.

Key Features

Complaint submission with category and priority

Status tracking and notifications

Admin dashboard for issue management

Resolution time analytics

AI Integration

Automatic complaint categorization

Priority prediction based on complaint history

Expected Outcome

A transparent and efficient complaint resolution system improving hostel living standards.

4. **Library Book Availability & Waitlist System**

Problem Statement

Students frequently encounter unavailable library books with no clarity on return dates or waitlist status.

Objective

Build a digital library management system that shows real-time book availability, manages waitlists, and predicts return dates.

Key Features

Book inventory management

Waitlist queue system

Auto notifications for availability

Admin control panel

AI Integration

Predict book return dates based on past usage

Demand analysis for popular books

Expected Outcome

A smarter library experience with improved accessibility and reduced uncertainty.

APP DEV problem

5. statement

AI Resume Builder & Analyzer Application

Problem Statement

In today’s competitive job market, a large percentage of resumes are rejected by Applicant Tracking Systems (ATS) before they ever reach a recruiter. Most job seekers lack visibility into why their resumes are rejected, which skills they are missing, or how to tailor their resumes for specific job roles. This results in repeated rejections despite having relevant capabilities.

Objective

Design and develop an AI-powered application that analyzes, scores, and optimizes resumes by comparing them against specific job descriptions, helping candidates improve their chances of shortlisting.

Key Features

Resume upload (PDF/DOC) and in-app resume builder

Job description input or selection

ATS compatibility score prediction

Skill gap analysis between resume and job role

AI-powered improvement suggestions

Export optimized, ATS-friendly resume

==========