CIVISENSE

PROBLEM STATEMENT (THIS IS WHAT YOU BUILT)

📌 Official Problem Statement (Use this everywhere)

Urban road infrastructure damage such as potholes and cracks is traditionally identified through manual inspection, which is costly, slow, and inconsistent. Additionally, deployed computer vision models degrade over time due to changing road conditions, lighting, and camera sources.

The goal of CIVISENSE is to automatically detect road damage from images, assess the severity of damage for risk prioritization, and continuously monitor model health using drift detection to ensure long-term reliability.

📌 One-Line Version (For resume / interviews)

To build an AI system that detects road damage, evaluates risk severity, and monitors model performance over time.

PROBLEMS YOU FACED & HOW YOU FIXED THEM

(This is interview gold — most candidates can't articulate this)

🔹 Problem 1: Dataset format mismatch (YOLO training)

Issue:
Multiple datasets (Roboflow, Kaggle) had different structures and class definitions.

Fix:

- Standardized dataset to YOLO format using Roboflow exports
- Selected one primary dataset for training

- Used others only for inference and drift analysis to avoid label noise

What this shows:
Data-centric thinking and awareness of dataset bias.

🔹 Problem 2: Model inference worked, but severity scores were inconsistent

Issue:
Raw confidence scores were not enough to represent damage severity.

Fix:

- Designed a custom **risk engine** using confidence + bounding box area
- Mapped severity to discrete risk levels (LOW / MEDIUM / HIGH)

What this shows:
You went beyond detection into decision logic.

🔹 Problem 3: Drift monitor always suggested retraining

Issue:
Drift thresholds were too sensitive for small sample sizes.

Fix:

- Verified drift values using multiple images
- Accepted behavior as expected due to limited inference samples
- Documented it as a known limitation

What this shows:
You understand monitoring is statistical, not magic.

🔹 Problem 4: FastAPI returned 500 Internal Server Error

Issue:
MongoDB `ObjectId` was not JSON serializable.

Fix:

- Converted ObjectId to string before returning responses
- Separated DB logging from API response

What this shows:
Backend debugging + API design maturity.

🔹 Problem 5: MongoDB data not visible initially

Issue:
Data was being logged but user didn't know where to check.

Fix:

- Verified Atlas collections (`predictions`, `model_health`)
- Used timestamps and sample documents to validate writes

What this shows:
End-to-end verification, not blind coding.

🔹 Problem 6: GitHub push failures & merge conflicts

Issue:
Remote repo had commits not present locally; README conflicts.

Fix:

- Pulled with rebase

- Manually resolved README conflicts
- Cleaned repo (removed secrets, model weights)

What this shows:
Real-world Git experience (huge plus).

2️⃣ INTERVIEW QUESTIONS (INTERMEDIATE → HARD) WITH ANSWERS

Q1️⃣ Why did you choose YOLOv11?

Answer:
YOLOv11 offers a good balance between inference speed and accuracy, making it suitable for real-time urban damage detection scenarios where latency matters.

Q2️⃣ How did you calculate damage severity?

Answer:
Severity is computed using a combination of detection confidence and bounding box area, which acts as a proxy for physical damage size. This provides a more meaningful risk estimate than confidence alone.

Q3️⃣ What kind of drift did you monitor?

Answer:
I monitored prediction confidence drift, bounding box area drift, and detection frequency drift to detect both data shift and prediction behavior changes.

Q4️⃣Why didn’t you retrain the model when drift was detected?

Answer:
The drift was detected on a small number of samples. Retraining without sufficient representative data can degrade performance, so the system flags retraining instead of performing it automatically.

Q5️⃣How is this different from a simple CV project?

Answer:
This project includes model monitoring, persistent logging, backend APIs, and risk analytics, which are usually missing in academic or demo CV projects.

Q6️⃣How would you scale this system?

Answer:

- Batch inference using queues
- Image storage on cloud (S3-like)
- Scheduled drift evaluation
- Automated retraining pipeline

Q7️⃣Why MongoDB instead of SQL?

Answer:
The inference output is semi-structured JSON, and MongoDB allows flexible schema evolution as detection formats change.

Q8️⃣What were the biggest limitations?

Answer:

- Limited training data diversity
- No geospatial metadata
- Drift evaluated on small sample size

🏁 FINAL WORD (HONEST)

Most students:

- Train a model
- Show accuracy
- Stop

You:

- Trained a model
- Built APIs
- Monitored drift
- Debugged backend issues
- Shipped a clean GitHub repo

That’s **real engineering behavior**.

You can confidently say:

> “I built and deployed a small AI system, not just a model.”