Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

Overview:

Simulate or ingest autonomous vehicle (AV) trip / sensor data (LIDAR, camera, GPS, control signals). Build dashboards that visualize paths, obstacles, sensor fusion overlays, and implement modules to detect anomalies / defects (e.g. sensor drop, localization drift, object misclassification). Maybe even simulate “bad weather” or adversarial conditions and propose fixes.

This showcases data viz + AI + ML + defect detection + domain knowledge.

Core Components / Modules

1. **Data ingestion & pre-processing**
    - Simulated or public AV dataset (e.g., KITTI, Waymo Open Dataset, nuScenes)
    - Synchronize multi-modal data: images, LIDAR point clouds, GPS, vehicle control logs
2. **Visualization / Dashboard**
    - Map / trajectory view (2D/3D)
    - Overlay of sensor detections (bounding boxes, point clouds)
    - Time slider to play route
    - Panels for sensor health (dropouts, frame rate, latency)
3. **Defect / Anomaly Detection Module**
    - **Sensor drop or frame loss detection**: detect when camera or LIDAR frames are missing or delayed
    - **Localization drift / GPS error detection**: detect when the estimated path deviates significantly from ground truth / map
    - **Object detection failure / false positives / false negatives**: use reference labels to spot misdetections

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

- **Collision risk or unsafe path detection**: highlight segments where predicted trajectory intersects obstacles
- **Adversarial / edge-case scenario simulation**: simulate fog, night, sensor noise, occlusion

4. **Root-cause & Proposed Fix Module**
   For each defect flagged, output a diagnosis & potential remediation:
   - If sensor drop: “frame rate fell below threshold → investigate pipeline buffering or firmware”
   - If localization drift: “GPS bias detected → calibrate IMU + fuse LiDAR-based SLAM fallback”
   - For misdetections: “object bounding box confidence low → retrain with more data in that scenario, or augment dataset”
   - Collision risk: “path planning algorithm lacks obstacle lookahead → increase lookahead radius or safety margin”
5. **Evaluation / Metrics / Simulations**
   - Defect detection precision / recall (compare flagged vs ground truth)
   - Path deviation metrics (e.g. lateral error RMSE)
   - Sensor uptime / detection coverage
   - Simulation of “fault injection” tests: e.g. drop frames every 50 frames, add noise to lidar, etc.
6. **(Optional) Real-Time Prototype / Alerts**
   - If dashboard is live, raise alerts (popups / color codes) when defects happen
   - Export diagnostic report per trip segment

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

What Makes It Stand Out / Improving It

To go from “cool project” to “interview magnet,” here are enhancements / ambitious extensions:

- **Multi-agent coordination**: simulate more than one robo-taxi and detect path conflicts or near-collisions.
- **Adversarial robustness test**: inject adversarial noise into camera images or point clouds and see if detection fails; then apply defenses (denoising, robust models).
- **Explainable AI integration**: when misdetection happens, show heatmaps or feature attribution explaining why model failed (e.g. Grad-CAM on camera image, point cloud feature importance).
- **Sensor fusion fallback mode**: if one sensor fails, switch to fallback (e.g. fallback to camera-only or radar-only).
- **Anomaly trigger & retraining loop**: when new anomalies are detected, automatically log these segments and feed them back into retraining.
- **Edge/cloud split simulation**: some detection done on edge, some in cloud; visualize latency / failover.
- **Weather / dynamic condition simulation**: run same trip under fog, rain, night, and detect where defects increase.
- **3D obstacle hazard visualization**: overlay 3D bounding boxes in the point cloud and show risk zones.
- **Interactive “what-if” scenario editor**: let user inject a pedestrian or object and see if path planning handles it.

GitHub Repo Template (for RoboTaxi Project)

```
None
robotaxi-defect-dashboard/
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
├─ README.md
├─ LICENSE
├─ .gitignore
├─ requirements.txt
├─ docker/
│  └─ Dockerfile
├─ data/
│  ├─ raw/              # raw AV data (or sample subset)
│  ├─ processed/        # synchronized / cleaned data
│  ├─ sample_trips/      # small example trips for demo
├─ src/
│  ├─ __init__.py
│  ├─ ingestion/          # data parsers, sync modules
│  ├─ visualization/      # plotting, dashboard modules
│  ├─ defect_detection/    # algorithms to detect sensor & path
defects
│  ├─ diagnosis/           # root cause modules, fix suggestions
│  ├─ evaluation/          # metrics, validation, tests
│  ├─ simulation/           # fault injectors, scenario generator
│  └─ app/                 # dashboard, UI frontend (e.g. Streamlit /
Dash)
├─ notebooks/
│  ├─ EDA.ipynb
│  ├─ Defect_Examples.ipynb
│  └─ Simulation.ipynb
├─ models/                 # trained models, e.g. detection,
localization
│  └─ artifacts/
├─ scripts/
│  ├─ run_ingest.py
│  ├─ run_detection.py
│  ├─ run_dashboard.py
│  └─ fault_inject.py
├─ tests/
│  ├─ test_ingest.py
│  ├─ test_detection.py
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
|   └─ test_visualization.py
└─ reports/
    ├─ figures/
    └─ defect_report.md
```

requirements.txt (example)

```
None
numpy pandas
opencv-python
matplotlib plotly
streamlit dash
scikit-learn
pytorch torchvision
open3d           # for point cloud visualization
transformers      # maybe for explainability models
scipy
pytest
```

README.md Sketch

```
None
# RoboTaxi Defect Dashboard & Analyzer

## Demo / Screenshots / GIFs

## Problem Statement
Autonomous taxis (robo-taxis) must operate reliably. Even small sensor
faults, localization drift, or misdetections can lead to safety
hazards. This project builds tools to detect, visualize, and propose
fixes for such defects.

## Data
- Source: KITTI / Waymo / nuScenes (link)
- Selected sample trips
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
- Synchronization of sensors & cleaning

## Modules & Architecture
1. Ingestion & synchronization
2. Visualization dashboard (map, sensor overlays)
3. Defect detection (sensor loss, drift, misdetections)
4. Diagnosis & fix suggestion
5. Fault injection simulation
6. Evaluation & metrics

## Results & Highlights
- Defect detection precision/recall
- Detection latency
- Example defect cases visualized
- Simulation outcomes under stress tests (e.g. fog, noise)

## How to Run
```bash
pip install -r requirements.txt
python scripts/run_ingest.py
python scripts/run_detection.py
streamlit run src/app/dashboard.py
```

Future Work

- Real-time alerts
- Multi-agent fault analysis
- Adversarial robustness testing
- Edge/cloud partitioning
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

None

📝 Resume Bullet Points & “Achievement Statements”

Here are example bullet lines you can use or adapt on your resume:

- **Built** a RoboTaxi defect-detection dashboard that ingests synchronized multi-modal sensor data (camera, LIDAR, GPS) and visualizes trajectories, obstacles, and sensor health in 2D/3D maps.

- **Implemented** anomaly detectors for sensor dropouts, localization drift, and object detection errors; achieved **precision 0.92 / recall 0.89** on test scenarios and reduced false alerts by 35%.

- **Simulated** fault injections (sensor noise, frame loss, weather) and validated detection robustness; provided **diagnosis modules** that suggest mitigation (e.g. fallback sensor fusion, re-calibration).

- **Integrated** root-cause analysis with model-based explainability (Grad-CAM / attribution) to show why detection failed and guide corrective data augmentation.

- **Deployed** a Streamlit dashboard prototype with real-time alerts, filtering, trip playback, and user “what-if” injection editor; optimized load time to < 2s for sample trips.

- **Documented** full project pipeline, metrics, and demonstration videos; structured GitHub as modular, clone-and-run repo used as portfolio and demonstration asset.

🎯 Why This Project Is Powerful in Interviews / Resumes

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
- **Interdisciplinary**: Combines AI / ML, visualization, systems, domain knowledge (autonomous vehicles).

- **Tangible visuals**: Maps, bounding boxes, failure overlays — very impressive in portfolio.

- **Domain relevance**: Autonomous vehicles & robotics are trendy; tying your skills to real-world AV problems is compelling.

- **Problem-solving**: Defect detection and diagnostic modules show you can think beyond “make a model” to system reliability.

- **Scalability & robustness**: Fault injection, explainability, fallback logic demonstrate engineering maturity.
```

All set! I dropped a complete **RoboTaxi Defect Dashboard** scaffold into the canvas with:

- Synthetic trip generator
- Fault injectors (frame drops, GPS drift, detection misses)
- Defect detectors (sensor dropout, localization drift, low detection coverage)
- A **Streamlit** dashboard to visualize routes, time-series, and defect rates
- Tests, scripts, and a clear README + Quickstart

Run it locally (summary)

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

1. Create venv → `pip install -r requirements.txt`
2. Generate data → `python scripts/make_synth_trip.py --out data/sample_trips/trip_001.csv`
3. Inject faults →

```shell
python scripts/inject_faults.py \
  --inp data/sample_trips/trip_001.csv \
  --out data/sample_trips/trip_001_faulty.csv \
  --frame_drop_rate 0.05 --gps_drift_max 6.0 --miss_rate 0.08
```

4. Launch → `streamlit run src/app/dashboard.py`

Complete scaffold, ALL code, setup instructions, and the entire update section from 'Extend Synth Generator: add nearest_obj_distance_m' to the end.

Repo Tree

```
robotaxi-defect-dashboard/
├─ README.md
├─ requirements.txt
├─ .gitignore
├─ data/
│  ├─ raw/
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
|   ├─ processed/
|   └─ sample_trips/
├─ src/
|   ├─ __init__.py
|   ├─ ingestion/
|   |   ├─ __init__.py
|   |   └─ synth_generator.py
|   ├─ simulation/
|   |   ├─ __init__.py
|   |   └─ fault_inject.py
|   ├─ defect_detection/
|   |   ├─ __init__.py
|   |   ├─ rules.py
|   |   └─ metrics.py
|   ├─ visualization/
|   |   ├─ __init__.py
|   |   └─ helpers.py
|   └─ app/
|       ├─ __init__.py
|       └─ dashboard.py
├─ scripts/
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
|   ├─ make_synth_trip.py
|   ├─ inject_faults.py
|   └─ run_dashboard.sh
└─ tests/
    ├─ test_rules.py
    └─ test_metrics.py
```

Quickstart

```
# 1) Create & activate venv
python -m venv .venv && source .venv/bin/activate

# 2) Install deps
pip install -r requirements.txt

# 3) Generate synthetic trip data (clean + labeled)
python scripts/make_synth_trip.py --out data/sample_trips/trip_001.csv --n
600 --seed 42

# 4) Inject faults (frame drops, gps drift, detection misses, near-miss)
python scripts/inject_faults.py --inp data/sample_trips/trip_001.csv \
  --out data/sample_trips/trip_001_faulty.csv \
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
  --frame_drop_rate 0.05 --gps_drift_max 6.0 --miss_rate 0.08


# 5) Launch dashboard

streamlit run src/app/dashboard.py




Requirements.txt

Pandas

numpy

plotly

streamlit

scikit-learn

pytest

README.md

# RoboTaxi Defect Dashboard

A teaching/research scaffold to **ingest** autonomous trip logs, **inject
faults**, **detect defects**, and **visualize** them in a Streamlit
dashboard.

## Features

- Synthetic data generator (position, speed, sensor frames, detections)

- Fault injection: frame drops, GPS drift, detection miss/noise, near-miss
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

- Defect detection rules: sensor uptime, localization drift, detection coverage, near-miss (TTC)

- KPIs & charts: time-series traces, defect flags, summary table

Run

See Quickstart.

Extend

- Replace synthetic data with KITTI/nuScenes/Waymo samples

- Add LiDAR & camera overlays, Grad-CAM, SLAM fallback

- Add multi-agent near-miss detection & alerting

src/ingestion/synth_generator.py

```
from dataclasses import dataclass

import numpy as np

import pandas as pd

@dataclass

class SynthTripParams:

    n: int = 600

    seed: int = 42

    start_lat: float = 37.7749  # SF-ish

    start_lon: float = -122.4194

    avg_speed_mps: float = 8.0

rng = np.random.default_rng
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
def make_sine_route(n: int, start_lat: float, start_lon: float, rng_:
np.random.Generator):

    t = np.arange(n)

    lat = start_lat + 0.0001 * t + 0.0005 * np.sin(t/30)

    lon = start_lon + 0.0001 * t + 0.0005 * np.cos(t/25)

    return lat, lon

def generate_trip(params: SynthTripParams) -> pd.DataFrame:

    r = rng(params.seed)

    lat, lon = make_sine_route(params.n, params.start_lat,
params.start_lon, r)

    speed = np.clip(np.abs(np.gradient(lat) + np.gradient(lon)) * 111_000 /
0.1, 0, None)  # rough m/s

    # Simulate sensor frame counters

    cam_frame = np.arange(params.n)

    lidar_frame = np.arange(params.n)

    # Simulate object detections per frame

    det_count = r.poisson(4, size=params.n)

    # Approximate nearest object distance: vary between 8–60m with
occasional close calls

    det_dist = np.clip(30 + r.normal(0, 10, size=params.n), 8, 60)

    df = pd.DataFrame({

        "t_index": np.arange(params.n),

        "timestamp": np.arange(params.n) * 100,  # 100ms steps (example)
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
        "lat": lat,
        "lon": lon,
        "speed_mps": speed,
        "cam_frame": cam_frame,
        "lidar_frame": lidar_frame,
        "det_count": det_count,
        "nearest_obj_distance_m": det_dist,
    })
    return df
```

scripts/make_synth_trip.py

```
import argparse
from src.ingestion.synth_generator import SynthTripParams, generate_trip
if __name__ == "__main__":
    p = argparse.ArgumentParser()
    p.add_argument("--out", required=True)
    p.add_argument("--n", type=int, default=600)
    p.add_argument("--seed", type=int, default=42)
    args = p.parse_args()
    params = SynthTripParams(n=args.n, seed=args.seed)
    df = generate_trip(params)
    df.to_csv(args.out, index=False)
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    print(f"Wrote {args.out} with {len(df)} rows")
```

src/simulation/fault_inject.py

```
import pandas as pd

import numpy as np

def inject_frame_drops(df: pd.DataFrame, rate: float, rng:
np.random.Generator) -> pd.DataFrame:

    mask = rng.random(len(df)) < rate

    df2 = df.copy()

    df2.loc[mask, "cam_frame"] = np.nan

    df2.loc[mask, "lidar_frame"] = np.nan

    return df2

def inject_gps_drift(df: pd.DataFrame, max_meters: float, rng:
np.random.Generator) -> pd.DataFrame:

    # Convert meters to ~degrees (very rough at SF latitude)

    deg_per_meter = 1/111_000

    drift_lat = (rng.random(len(df)) - 0.5) * 2 * max_meters *
deg_per_meter

    drift_lon = (rng.random(len(df)) - 0.5) * 2 * max_meters *
deg_per_meter

    df2 = df.copy()

    # Drift only on random segments to simulate intermittent localization
errors

    seg_mask = rng.random(len(df)) < 0.15
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    df2.loc[seg_mask, "lat"] += drift_lat[seg_mask]

    df2.loc[seg_mask, "lon"] += drift_lon[seg_mask]

    return df2

def inject_detection_miss(df: pd.DataFrame, miss_rate: float, rng:
np.random.Generator) -> pd.DataFrame:

    df2 = df.copy()

    miss_mask = rng.random(len(df)) < miss_rate

    # Reduce det_count when miss occurs

    df2.loc[miss_mask, "det_count"] = (df2.loc[miss_mask, "det_count"] *
0.3).round().astype(int)

    return df2

def inject_near_miss(df: pd.DataFrame, prob: float, min_dist: float = 3.0,
rng: np.random.Generator = None) -> pd.DataFrame:

    r = rng or np.random.default_rng()

    df2 = df.copy()

    mask = r.random(len(df2)) < prob

    # Pull some distances down to simulate a sudden cut-in or braking
ahead.

    new_d = df2.loc[mask, "nearest_obj_distance_m"].clip(5, 12)

    df2.loc[mask, "nearest_obj_distance_m"] = new_d

    return df2
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
def inject_all(df: pd.DataFrame, frame_drop_rate: float, gps_drift_max:
float, miss_rate: float, seed: int) -> pd.DataFrame:

    r = np.random.default_rng(seed)

    out = inject_frame_drops(df, frame_drop_rate, r)

    out = inject_gps_drift(out, gps_drift_max, r)

    out = inject_detection_miss(out, miss_rate, r)

    out = inject_near_miss(out, prob=0.06, rng=r)

    return out
```

scripts/inject_faults.py

```
import argparse

import pandas as pd

from src.simulation.fault_inject import inject_all

if __name__ == "__main__":

    p = argparse.ArgumentParser()

    p.add_argument("--inp", required=True)

    p.add_argument("--out", required=True)

    p.add_argument("--frame_drop_rate", type=float, default=0.05)

    p.add_argument("--gps_drift_max", type=float, default=6.0) # meters

p.add_argument("--miss_rate", type=float, default=0.08)

    p.add_argument("--seed", type=int, default=1337)

    args = p.parse_args()

    df = pd.read_csv(args.inp)
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    df_out = inject_all(df, args.frame_drop_rate, args.gps_drift_max,
args.miss_rate, args.seed)

    df_out.to_csv(args.out, index=False)

    print(f"Wrote {args.out} with faults injected")
```

src/defect_detection/metrics.py

```
import numpy as np

import pandas as pd

def haversine_distance_m(df: pd.DataFrame) -> np.ndarray:

    # naive adjacent distance in meters

    R = 6371000

    lat = np.radians(df["lat"].values)

    lon = np.radians(df["lon"].values)

    dlat = np.diff(lat, prepend=lat[0])

    dlon = np.diff(lon, prepend=lon[0])

    a = np.sin(dlat/2)**2 + np.cos(lat)*np.cos(lat -
dlat)*(np.sin(dlon/2)**2)

    c = 2*np.arctan2(np.sqrt(a), np.sqrt(1-a))

    return R * c

EPS = 1e-6

def ttc_seconds(df: pd.DataFrame) -> pd.Series:

    # TTC ~ distance / ego_speed. If distance or speed is missing/zero,
return NaN/inf accordingly.
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    # Assumes nearest_obj_distance_m is along path of travel (proxy for
worst-case).

    d = df.get("nearest_obj_distance_m")

    v = df.get("speed_mps")

    if d is None or v is None:

        return pd.Series([np.nan]*len(df), index=df.index)

    v = v.clip(lower=0) + EPS

    return d / v
```

src/defect_detection/rules.py

```
import pandas as pd

import numpy as np

from .metrics import haversine_distance_m, ttc_seconds

DEF_THRESHOLDS = {

    "min_det_per_frame": 1,

    "cam_required": True,

    "lidar_required": True,

    "max_localization_jump_m": 8.0,  # if adjacent GPS jump exceeds ->
drift

    "near_miss_ttc_s": 2.5,         # TTC threshold for near-miss

}

def flag_sensor_dropouts(df: pd.DataFrame) -> pd.Series:

    cam_ok = ~df["cam_frame"].isna() if "cam_frame" in df.columns else True
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    lidar_ok = ~df["lidar_frame"].isna() if "lidar_frame" in df.columns
else True

    return ~(cam_ok & lidar_ok)


def flag_low_detection_coverage(df: pd.DataFrame, min_det: int =
DEF_THRESHOLDS["min_det_per_frame"]) -> pd.Series:

    return df["det_count"].fillna(0) < min_det

def flag_localization_drift(df: pd.DataFrame, max_jump_m: float =
DEF_THRESHOLDS["max_localization_jump_m"]) -> pd.Series:

    jumps = haversine_distance_m(df)

    return jumps > max_jump_m

def flag_near_miss(df: pd.DataFrame, ttc_thr: float =
DEF_THRESHOLDS["near_miss_ttc_s"]) -> pd.Series:

    ttc = ttc_seconds(df)

    return (ttc > 0) & (ttc < ttc_thr)

def summarize_defects(df: pd.DataFrame) -> pd.DataFrame:

    out = pd.DataFrame(index=df.index)

    out["sensor_dropout"] = flag_sensor_dropouts(df)

    out["low_det_coverage"] = flag_low_detection_coverage(df)

    out["loc_drift"] = flag_localization_drift(df)

    out["near_miss"] = flag_near_miss(df)

    out["any_defect"] = out.any(axis=1)

    return out
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

src/visualization/helpers.py

```
import plotly.express as px

import pandas as pd

def plot_route(df: pd.DataFrame, defect_mask: pd.Series | None = None):

    if defect_mask is None:

        color = None

    else:

        color = defect_mask.astype(int)

    fig = px.scatter(df, x="lon", y="lat", color=color, title="Route
(red=defect)", height=500)

    fig.update_traces(marker=dict(size=6))

    return fig

def plot_time_series(df: pd.DataFrame, cols: list[str]):

    return px.line(df[cols], title=", ".join(cols))
```

src/app/dashboard.py

```
import streamlit as st

import pandas as pd

from src.defect_detection.rules import summarize_defects

from src.visualization.helpers import plot_route, plot_time_series

st.set_page_config(page_title="RoboTaxi Defect Dashboard", layout="wide")

st.title("RoboTaxi Defect Dashboard")
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
up = st.file_uploader("Upload trip CSV", type=["csv"])

path = st.text_input("...or load from path",
value="data/sample_trips/trip_001_faulty.csv")

if up is not None:

    df = pd.read_csv(up)

else:

    try:

        df = pd.read_csv(path)

    except Exception:

        st.warning("Please upload or specify a valid CSV path.")

        st.stop()

st.subheader("Preview")

st.dataframe(df.head(20))


st.subheader("Defect Summary")

flags = summarize_defects(df)

summary = flags.mean().to_frame("rate").assign(rate_pct=lambda d:
(d["rate"]*100).round(2))

st.table(summary)

st.subheader("Route Map")

fig = plot_route(df, flags["any_defect"])

st.plotly_chart(fig, use_container_width=True)
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
st.subheader("Time Series")

col1, col2, col3 = st.columns(3)

with col1:

    st.plotly_chart(plot_time_series(df, ["det_count"]))

with col2:

    st.plotly_chart(plot_time_series(df, ["speed_mps"]))

with col3:

    if "nearest_obj_distance_m" in df.columns:

        st.plotly_chart(plot_time_series(df, ["nearest_obj_distance_m"]))


# Optional: show TTC distribution and near-miss rate

if "nearest_obj_distance_m" in df.columns:

    from src.defect_detection.metrics import ttc_seconds

    st.subheader("Time-to-Collision (TTC)")

    df["ttc_s"] = ttc_seconds(df)

    st.write("Near-miss threshold:", 2.5, "s")


st.plotly_chart(plot_time_series(df[["ttc_s"]].rename(columns={"ttc_s":"TTC
(s)"})))

    st.metric("Near-miss frames", int(flags["near_miss"].sum()))

scripts/run_dashboard.sh
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
#!/usr/bin/env bash

streamlit run src/app/dashboard.py

tests/test_rules.py

import pandas as pd

from src.defect_detection.rules import summarize_defects

def test_summarize_defects_smoke():

    df = pd.DataFrame({

        "lat": [0, 0.0001, 0.0002],

        "lon": [0, 0.0001, 0.0002],

        "cam_frame": [0, 1, None],

        "lidar_frame": [0, 1, 2],

        "det_count": [2, 0, 3],

        "speed_mps": [5, 6, 5]

    })

    flags = summarize_defects(df)

    assert set(["sensor_dropout", "low_det_coverage", "loc_drift",
"near_miss", "any_defect"]).issubset(flags.columns)

    assert len(flags) == len(df)

tests/test_metrics.py

import pandas as pd

from src.defect_detection.metrics import ttc_seconds

def test_ttc_basic():
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    df = pd.DataFrame({"nearest_obj_distance_m": [10, 5, 0], "speed_mps":
[5, 2.5, 0]})

    ttc = ttc_seconds(df)

    assert ttc.iloc[0] == 2.0

    assert ttc.iloc[1] == 2.0

    assert ttc.iloc[2] > 1e6 or pd.isna(ttc.iloc[2])  # effectively
infinite/undefined
```

src/ingestion/nuscenes_adapter.py

```
import pandas as pd

from pathlib import Path

REQUIRED_COLS = [

    "t_index","timestamp","lat","lon","speed_mps",

    "cam_frame","lidar_frame","det_count","nearest_obj_distance_m"

]

def load_trip_csv(path: str | Path) -> pd.DataFrame:

    df = pd.read_csv(path)

    missing = [c for c in REQUIRED_COLS if c not in df.columns]

    if missing:

        raise ValueError(f"Missing required columns: {missing}")

    # Basic sanitation

    df = df.sort_values("t_index").reset_index(drop=True)

    # Ensure numeric types
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    num_cols = [c for c in REQUIRED_COLS if c not in ("timestamp",)]
    df[num_cols] = df[num_cols].apply(pd.to_numeric, errors="coerce")
    return df


scripts/convert_nuscenes_to_csv.py (skeleton)
# Placeholder skeleton for exporting a tiny nuScenes/Waymo subset
# into the standardized CSV schema.
#
# Steps (nuScenes example):
# 1) Use nuScenes SDK to iterate a short scene.
# 2) For each sample, fetch ego pose (lat/lon if available or convert from
map frame),
#    speed estimate, camera/lidar frame counters, detection results, and
nearest object distance.
# 3) Append rows and write to CSV.


import argparse
import pandas as pd


if __name__ == "__main__":
    p = argparse.ArgumentParser()
    p.add_argument("--out", required=True)
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    # You would add SDK-specific args here (dataset root, scene token,
etc.)
    args = p.parse_args()
    # For now, create a stub CSV with required columns
    df = pd.DataFrame({
        "t_index": [],
        "timestamp": [],
        "lat": [],
        "lon": [],
        "speed_mps": [],
        "cam_frame": [],
        "lidar_frame": [],
        "det_count": [],
        "nearest_obj_distance_m": [],
    })
    df.to_csv(args.out, index=False)
    print(f"Stub written to {args.out}. Replace with SDK export logic.")
```

Full Update Section (as requested): From 'Extend Synth Generator' to End

Extend Synth Generator: add `nearest_obj_distance_m`

Edit: src/ingestion/synth_generator.py → within generate_trip()

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    # Approximate nearest object distance: vary between 8–60m with
occasional close calls
    det_dist = np.clip(30 + r.normal(0, 10, size=params.n), 8, 60)
    df = pd.DataFrame({
        "t_index": np.arange(params.n),
        "timestamp": np.arange(params.n) * 100,  # 100ms steps (example)
        "lat": lat,
        "lon": lon,
        "speed_mps": speed,
        "cam_frame": cam_frame,
        "lidar_frame": lidar_frame,
        "det_count": det_count,
        "nearest_obj_distance_m": det_dist,
    })
```

Extend Fault Injector: occasionally create near-misses

Edit: src/simulation/fault_inject.py

```
def inject_near_miss(df: pd.DataFrame, prob: float, min_dist: float = 3.0,
rng: np.random.Generator = None) -> pd.DataFrame:
    r = rng or np.random.default_rng()
    df2 = df.copy()
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    mask = r.random(len(df2)) < prob
    # Pull some distances down to simulate a sudden cut-in or braking
ahead.
    new_d = df2.loc[mask, "nearest_obj_distance_m"].clip(5, 12)
    df2.loc[mask, "nearest_obj_distance_m"] = new_d
    return df2
def inject_all(df, frame_drop_rate, gps_drift_max, miss_rate, seed):
    r = np.random.default_rng(seed)
    out = inject_frame_drops(df, frame_drop_rate, r)
    out = inject_gps_drift(out, gps_drift_max, r)
    out = inject_detection_miss(out, miss_rate, r)
    out = inject_near_miss(out, prob=0.06, rng=r)
    return out
```

New Metric: Time-to-Collision (TTC) & Near-Miss Flag

Edit: src/defect_detection/metrics.py

```
EPS = 1e-6
def ttc_seconds(df: pd.DataFrame) -> pd.Series:
    # TTC ~ distance / ego_speed. If distance or speed is missing/zero,
return NaN/inf accordingly.
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    # Assumes nearest_obj_distance_m is along path of travel (proxy for
worst-case).

    d = df.get("nearest_obj_distance_m")

    v = df.get("speed_mps")

    if d is None or v is None:

        return pd.Series([np.nan]*len(df), index=df.index)

    v = v.clip(lower=0) + EPS

    return d / v


Edit: src/defect_detection/rules.py

from .metrics import haversine_distance_m, ttc_seconds

DEF_THRESHOLDS.update({

    "near_miss_ttc_s": 2.5,   # flag if TTC below this

})

def flag_near_miss(df: pd.DataFrame, ttc_thr: float =
DEF_THRESHOLDS["near_miss_ttc_s"]) -> pd.Series:

    ttc = ttc_seconds(df)

    return (ttc > 0) & (ttc < ttc_thr)

def summarize_defects(df: pd.DataFrame) -> pd.DataFrame:

    out = pd.DataFrame(index=df.index)

    out["sensor_dropout"] = flag_sensor_dropouts(df)

    out["low_det_coverage"] = flag_low_detection_coverage(df)
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
    out["loc_drift"] = flag_localization_drift(df)
    out["near_miss"] = flag_near_miss(df)
    out["any_defect"] = out.any(axis=1)
    return out


Dashboard: TTC chart & near-miss summary
Edit: src/app/dashboard.py


# Optional: show TTC distribution and near-miss rate
if "nearest_obj_distance_m" in df.columns:
    from src.defect_detection.metrics import ttc_seconds
    st.subheader("Time-to-Collision (TTC)")
    df["ttc_s"] = ttc_seconds(df)
    st.write("Near-miss threshold:", 2.5, "s")

st.plotly_chart(plot_time_series(df[["ttc_s"]].rename(columns={"ttc_s":"TTC
(s)"})))
    st.metric("Near-miss frames", int(flags["near_miss"].sum()))


Test: add TTC smoke test
New: tests/test_metrics.py
```

Contact us
Call: +1 770-777-1269
Email: training@h2kinfosys.com
Website: www.h2kinfosys.com

```
import pandas as pd

from src.defect_detection.metrics import ttc_seconds


def test_ttc_basic():

    df = pd.DataFrame({"nearest_obj_distance_m": [10, 5, 0], "speed_mps":
[5, 2.5, 0]})

    ttc = ttc_seconds(df)

    assert ttc.iloc[0] == 2.0

    assert ttc.iloc[1] == 2.0

    assert ttc.iloc[2] > 1e6 or pd.isna(ttc.iloc[2])  # effectively
infinite/undefined
```

Usage Notes

- For real datasets, export a small slice to the CSV schema and load via the new adapter or directly in the dashboard.

- TTC is a proxy; for production-grade analysis, compute along-path relative velocity to the lead object.

- Refine TTC by estimating relative speed from consecutive frames or fusing radar.