

AI Predictive Maintenance Development Document

Objective: Enhance vehicle safety and efficiency by predicting maintenance needs using an AI-driven system before issues escalate, reducing downtime and maintenance costs.

System Architecture and design document: Please refer document below-

[AI_Driven_Vehicle_Design_Document](#)

Detailed Component Overview:

1) NLP for Diagnostic Logs-

Functionality Overview: This component processes these logs to extract actionable insights, aiding in early diagnosis and prevention of potential failures.

Technical Implementation:

a) Data Ingestion Pipeline:

- Set up Google Cloud Storage to store vehicle sensor data and diagnostic logs.

Google Cloud

AutonomusVehicles

Search (/) for resources, docs, products, and more

Search

Cloud Storage

Overview

PREVIEW

Buckets

Monitoring

Settings

Marketplace

Release Notes

Bucket details

GO TO PATH

REFRESH

bucket_data

Location: us (multiple regions in United States)

Storage class: Standard

Public access: Not public

Protection: Soft Delete

OBJECTS

CONFIGURATION

PERMISSIONS

PROTECTION

LIFECYCLE

OBSERVABILITY

INVENTORY REPORTS

OPERATION

Folder browser

bucket_data

AutomotiveVehiclesData

car-damage-detection-data

json

CREATE FOLDER

UPLOAD

TRANSFER DATA

OTHER SERVICES

Filter by name prefix only

Filter

Filter objects and folders

Show Live objects only

Name	Size	Type	Created	Storage
vehicle_diagnostics_logs.csv	2 KB	text/csv	Sep 28, 2024, 2:33:53 PM	Stand

- **Create an Event trigger** via Eventarc trigger which automatically invokes the `publish_vehicle_logs` function whenever a new file is uploaded to the specified bucket in Google Cloud Storage.

Google Cloud | AutonomusVehicles | eventarc

Eventarc | Create trigger

Triggers | Channels

Trigger name *
datatrigger

Trigger type *
Google sources

Event provider *
Cloud Storage

Event type *
google.cloud.storage.object.v1.finalized

SHOW DETAILS

Event data content type
application/json

Bucket *
☒ bucket_data BROWSE
 Google Cloud Storage bucket to subscribe to

Region
us (multiple regions in United States)

CREATE CANCEL

Publish vehicle logs function- [publish_vehicle_logs.py](#)

Create a Pub/Sub topic to receive notifications of new entries added to Cloud Storage. Acts as a messaging service to trigger the processing function whenever new log data is available.

Google Cloud | AutonomusVehicles | pub/sub

Pub/Sub | Topics | CREATE TOPIC | DELETE

LIST | METRICS

Filter Filter topics

Topic ID ↑	Encryption key	Topic name	
vehicle-sensor-stream	Google-managed	projects/autonomusvehicles/topics/vehicl...	⋮

Pub/Sub Lite

Lite Reservations

Lite Topics

Lite Subscriptions

Release Notes

Set up a subscription to this topic that triggers a Cloud Function whenever new data is published.

The screenshot shows the Google Cloud Pub/Sub console. The top navigation bar includes the Google Cloud logo, the project name 'AutonomusVehicles', and a search bar containing 'pub/sub'. The left sidebar shows the 'Pub/Sub' section with 'Subscriptions' selected. The main content area displays details for a subscription named 'vehicle-data-s...'. The subscription name is 'projects/autonomusvehicles/subscriptions/vehicle-data-subscription', the state is 'active', and the topic name is 'projects/autonomusvehicles/topics/vehicle-sensor-stream'. Below the details, there are tabs for 'METRICS', 'DETAILS', and 'MESSAGES'. The 'METRICS' tab is active, showing a 'GO BACK TO CLASSIC METRICS' link and a 'SAVE AS DASHBOARD' button. A 'RESET ZOOM' button and a time range selector (set to '1 hour') are also present. The 'Overview' section shows 'Health' and 'Pull' status. A 'Metrics' card displays 'Oldest unacked message age' as '1s'. The bottom of the screen shows a Windows taskbar with a search bar and various application icons.

- Use **Cloud Functions** to analyze logs to summarize key issues and publish the data to Pub/Sub.

Steps:

1. Develop a Python function using the Natural Language Toolkit (NLTK) to process and analyze text.
2. Deploy this function in Google Cloud Functions, triggered by the Pub/Sub subscription.
3. Function writes processed data to Google BigQuery for further analysis and historical tracking.

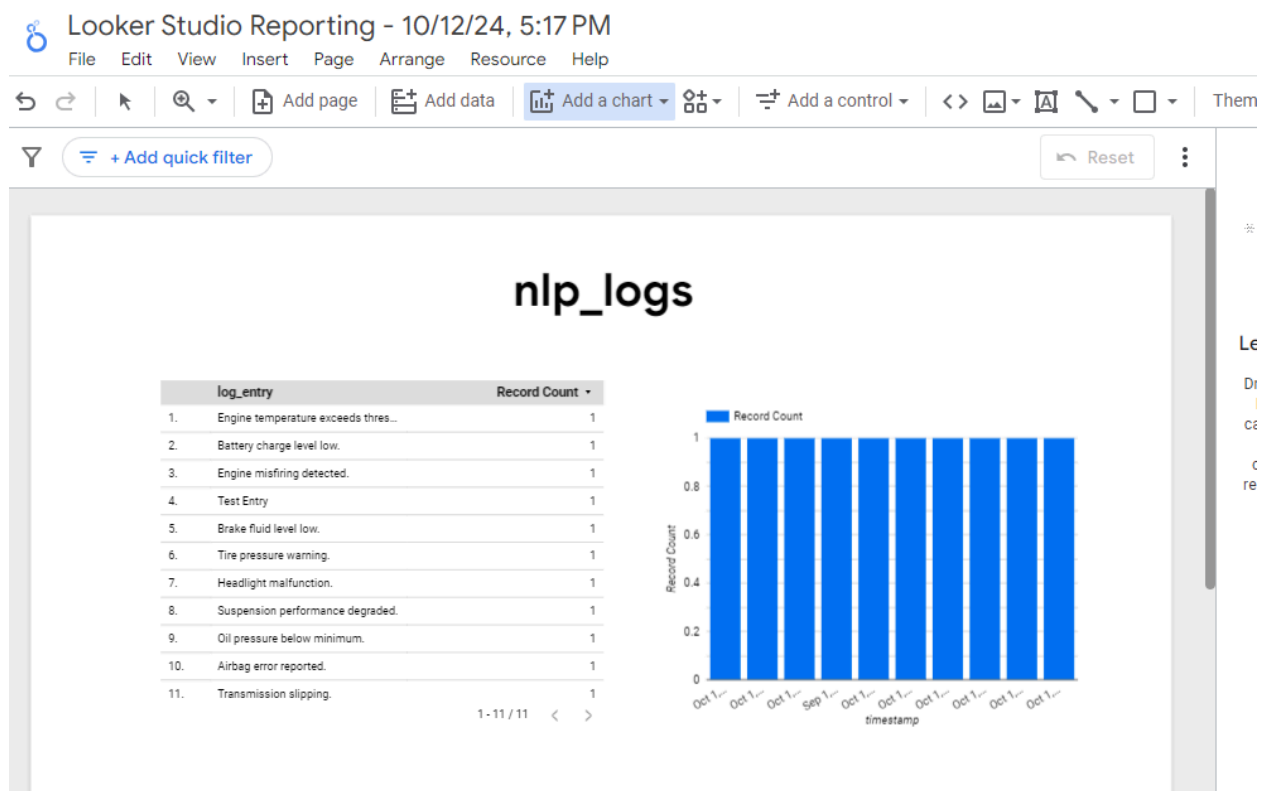
Process nlp functions- [process_nlp_function](#)

The screenshot shows the Google BigQuery interface. On the left is the Explorer panel with a search bar and a tree view of resources including Queries, Notebooks, Data canvases, Data preparations, Workflows, External connections, and vehicle_diagnostics. The main panel displays a query titled 'Untitled query' with the SQL: `select * from autonomousvehicles.vehicle_diagnostics.nlp_logs`. Below the query is the 'Query results' section, which includes tabs for JOB INFORMATION, RESULTS, CHART, JSON, EXECUTION DETAILS, and EXECUTION GRAPH. The 'RESULTS' tab is active, showing a table with 10 rows of data. The table has columns: Row, log_entry, summary, and timestamp. The data includes various vehicle diagnostic messages and their corresponding timestamps.

Row	log_entry	summary	timestamp
1	Test Entry	This is a test summary.	2021-09-01 12:00:00 UTC
2	Headlight malfunction.	Replace headlight bulbs.	2023-10-01 15:00:00 UTC
3	Tire pressure warning.	Check tire pressure.	2023-10-01 11:00:00 UTC
4	Transmission slipping.	Evaluate transmission clutch.	2023-10-01 17:00:00 UTC
5	Brake fluid level low.	Refill brake fluid.	2023-10-01 13:00:00 UTC
6	Airbag error reported.	Inspect airbag systems.	2023-10-01 12:00:00 UTC
7	Engine temperature exceeds th...	Check coolant system.	2023-10-01 08:00:00 UTC
8	Oil pressure below minimum.	Immediate oil check required.	2023-10-01 10:00:00 UTC
9	Suspension performance degra...	Suspension check advised.	2023-10-01 14:00:00 UTC
10	Battery charge level low.	Possible battery replacement n...	2023-10-01 09:00:00 UTC

Theoretical Final Steps:

- **Data Integration-** Connect Looker Studio to BigQuery to automatically pull in real-time NLP processed data for immediate analysis.



- **Dashboard Design-** Design a user-friendly dashboard that displays critical metrics like error frequency, issue types, and urgent alerts through easy-to-read charts and conditional color coding.

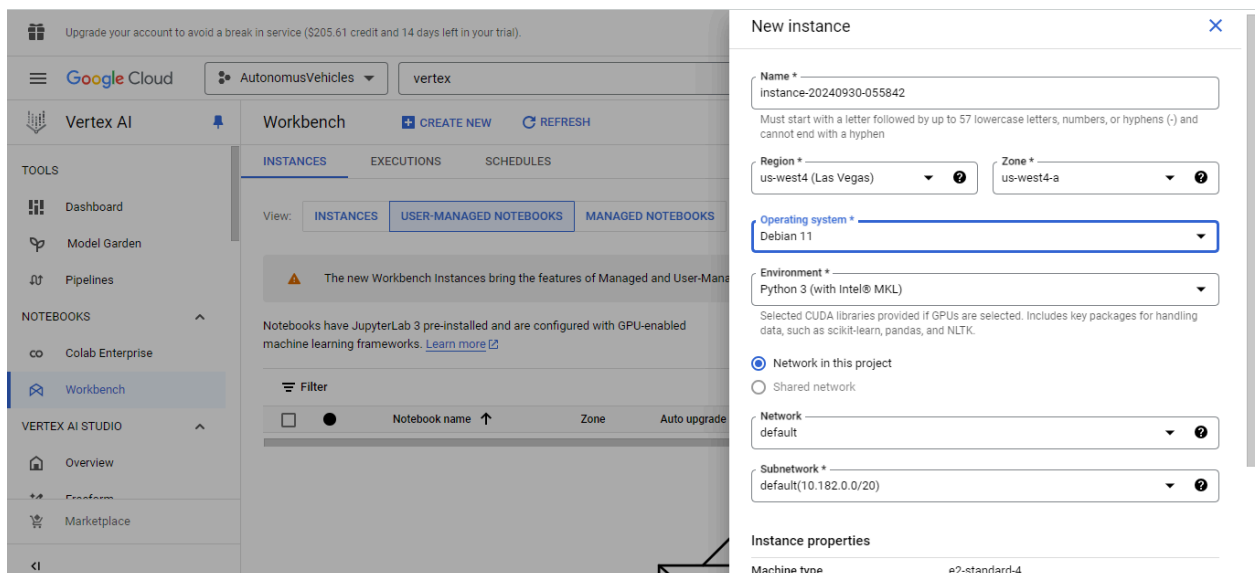
- **Alert Simulation-** Set theoretical thresholds for alerts that, when exceeded, highlight urgent issues on the dashboard to prompt immediate attention.

2) Computer vision module

Objective: Develop a Computer Vision system that analyzes images from vehicle cameras to detect signs of wear, damage, or other maintenance needs before they lead to failure.

Technical Implementation:

1. **Data Collection and data storage:** Set up onboard cameras in strategic locations on the vehicle to capture images of critical components and areas. Configure Google Cloud Storage to store and manage the large volumes of image data collected from vehicles. [COCO Damage Detection - Trained Models](#). This dataset provides pretrained models that are trained on the COCO dataset, specifically fine-tuned for damage detection.
2. **Choose a Pre-Trained Model:** You can use a **pre-trained YOLOv5 model** from **TensorFlow**.
3. **Create a Vertex AI Workbench (Notebook Instance)** Navigate to Vertex AI: From the Navigation Menu, go to AI/Vertex AI > Workbench > User-managed notebooks. Create a new dataset configuration file `car_damage.yaml` to point to the Cloud Storage paths. Install YOLOv5 in the Notebook to read from the bucket. [car_damage.yaml](#)



4. **Test YOLOv5 with Your Dataset:** Use the YOLOv5 detection script to run inference on a few validation images from the dataset.

```
python detect.py --weights yolov5s.pt --img 640 --conf 0.25 --source  
gs://bucket__data/car-damage-detection-data/validation/00-damage
```

This will run YOLOv5's inference on the images and display the damaged part of cars.

5. **Visualize the Output** After running the inference, YOLOv5 will output images with bounding boxes drawn around detected objects. These images will be saved in the **runs/detect/exp** folder in your YOLOv5 directory.

Workspace: [Workspace](#)

6. **Train the YOLOv5 Model:** Run the YOLOv5 training command with the configuration file created earlier (car_damage.yaml):

```
python train.py --img 640 --batch 16 --epochs 50 --data car_damage.yaml --weights  
yolov5s.pt --cache
```

7. **Deploy the Model:** The trained YOLOv5 model is deployed as a Google Cloud Function. This function processes images from the car's cameras, detecting damage when it occurs.
8. **Analyze and Notify:** The Cloud Function analyzes the images and, upon confirming damage, sends a notification with the details (e.g., type and extent of damage) to the car owner's mobile app.

3) Physics-Informed Neural Networks (PINNs)

Objective: The goal is to develop a Physics-Informed Neural Network (PINN) that predicts vehicle component wear and stress over time based on sensor data (temperature, pressure, vibration). The model will simulate the physical behavior of vehicle components (e.g., engine, brakes) and predict failure times, allowing for proactive maintenance.

Tools:

- **Vertex AI Workbench:** This platform provides a managed environment for developing machine learning models. We use it to train and deploy our PINN.
- **TensorFlow:** A machine learning library used to define, train, and deploy the PINN model.
- **NumPy:** A library for numerical operations, used here to simulate sensor data and perform mathematical operations.
- **SciPy:** Useful for solving differential equations and integrating scientific computing tasks into the model.
- **Matplotlib:** A plotting library to visualize the simulated data and model predictions.

Functionality: A PINN integrates physical laws (like differential equations) into the structure of a neural network. It will simulate the behavior of specific car components (e.g., engine, brakes) and predict how long they will last based on real-world data.

- Vertex AI Workbench is used for development. PINNs can be implemented using TensorFlow along with some scientific libraries for physics-based simulations. Ensure the required libraries are installed: TensorFlow, SciPy, Matplotlib, NumPy, and other necessary scientific libraries.
- Kaggle Data: AI4I 2020 predictive maintenance dataset is used. The data includes: Temperature, pressure, vibration, and wear information over time.
- Define a PINN architecture in TensorFlow. PINN will solve a differential equation like heat transfer or stress analysis and predict the time to failure.
- Simulate Vehicle Component Data: We need to simulate historical vehicle data that reflects how certain components degrade over time. For example, we can simulate temperature, pressure, or vibration data from sensors in the vehicle. In real time, this component would receive real-time sensor data from the vehicle's onboard diagnostic system and predict when a part is likely to fail.
- Train the PINN Model: We train the PINN model to predict when a component will fail based on simulated data (e.g., temperature or pressure changes over time). The training process uses historical data and physical laws (e.g., stress-strain relationships) to predict failure points. Train the PINN model using the simulated data.
- Evaluate and Predict Failure Time: PINN can predict how much longer a component will last before failure. For example, we can feed the model new sensor data and predict when a specific part (like the engine or brakes) might overheat or break down. In real time: The model continuously monitors the

vehicle's sensor data and provides real-time predictions on when maintenance is required.

Steps:

Step 1: Environment Setup

- Libraries: TensorFlow, SciPy, NumPy, Matplotlib.
- Method: Use `pip` to install libraries in the Jupyter notebook environment on Vertex AI Workbench.

Step 2: Simulate Vehicle Component Data

- Libraries: NumPy, Matplotlib.
- Method: Use `np.linspace()` to simulate time intervals and `np.random.normal()` to add random noise to the data. Visualize the data using `plt.plot()` from Matplotlib.

Step 3: Define the PINN Architecture in TensorFlow

- Library: TensorFlow.
- Method: Use `Sequential()` to create a neural network model with `Dense()` layers for input, hidden, and output layers. Activation functions like `tanh` and `linear` are applied.

Step 4: Incorporate Physics Laws into the Loss Function

- Library: TensorFlow.
- Method: Use `GradientTape()` to compute derivatives required for the physics-informed loss function. Use `tf.reduce_mean()` to calculate the residuals for physical laws and the data loss.

Step 5: Train the PINN Model

- Library: TensorFlow.
- Method: Use `compile()` to define the optimizer (`Adam`) and loss function. Train the model using `fit()` by providing the training data and specifying epochs and validation split.

Step 6: Predict Component Failure

- Libraries: TensorFlow, Matplotlib.

- Method: Use `predict()` to forecast component behavior based on new sensor data. Use Matplotlib's `plt.plot()` to visualize predictions and determine failure points by defining a threshold.

Step 7: Deploy the Model on Vertex AI

- Library: Google Cloud AI Platform (`aiplatform`).
- Method: Use `Model.upload()` to upload the trained model to Vertex AI and `deploy()` to create an endpoint for real-time predictions.

Step 8: Real-Time Predictions and Alerts

- Libraries: Vertex AI, Custom API for Alerts.
- Method: Ingest real-time sensor data and feed it into the deployed model for continuous predictions. When failure is predicted, trigger an alert via a mobile app or dashboard using API-based alerts.

Code : [Code link](#)

Integration steps of PINN model:

Data Ingestion from Real-Time Sensors:

- **Goal:** Set up a mechanism to collect real-time sensor data (temperature, pressure, vibration) from the vehicle.
- **Approach:** Use **Google Cloud IoT** or edge computing to ingest and process real-time data from the vehicle's onboard diagnostic system.

Event-Driven Architecture:

- **Goal:** Set up triggers to automatically run predictions when new sensor data is available.
- **Approach:** Implement **Google Cloud Pub/Sub** or **Eventarc** to trigger predictions when new logs or sensor data are uploaded.

Integration of Notifications:

- **Goal:** Create a notification system to alert car owners of imminent component failure.
- **Approach:** Use an API to send alerts via a **mobile app** or in-car **dashboard**. Could be based on Google Firebase for app notifications or in-car infotainment systems for dashboard alerts.

Evaluation Metrics:

- **Goal:** Define evaluation metrics to assess the performance of the model (e.g., Mean Squared Error, R^2 , precision of failure predictions).
- **Approach:** Track metrics during training and testing using TensorFlow's built-in evaluation methods and adjust hyperparameters to improve the model.

