IIoT Based Thermal System Monitoring and Predictive Fault Detection

Thermal instability and inefficiency in the heater, pipeline network pose operational risks. our IIoT platform provides predictive fault detection to avoid failures.

Arizona State University

Presented by: Jeet Khut, Nisarg Nakrani, Nisarg Patel
Date: 12/03/2025

Agenda

- Problem Overview
- System Architecture
- Physical Device Layer
- Network Layer
- Working Demo
- Challenges
- Future Scope
- Conclusion

Problem Overview & Objectives

➢ **Problem Statement:**
Industrial thermal systems, such as fluid heaters and heat transfer pipelines utilized in chemical processing, are prone to various issues, including overheating, pressure fluctuations, flow disturbances, vibration-related mechanical faults and energy inefficiencies. These problems often develop slowly and go unnoticed until they cause unexpected downtime, stress on equipment, and higher energy consumption. The lack of continuous monitoring and early diagnostic emphasizes the critical need for a predictive maintenance system designed to detect thermal anomalies before they result in equipment failure.

➢ **Goal:**
To develop an IIoT based system that detects early faults and improves the reliability and efficiency of industrial thermal equipment.

- Detect early signs of overheating,thermal inefficiency.
- Monitor pressure, flow rate, vibration patterns indicating pump or motor issues.
- Replace manual inspections with real-time monitoring.
- Reduce downtime and maintenance costs.
- Demonstrate a full predictive maintenance workflow (sensor → edge → cloud → dashboard).

Problem Overview & Objectives

Why is IIoT needed here?

➢ Operators traditionally rely on manual inspections to monitor their equipment and usually they detect these problems only after performance has already degraded or a failure has occurred which leads to

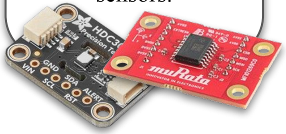
- Late detection of overheating
- Unnecessary energy waste
- Unexpected shutdowns of equipments
- Higher maintenance and repair costs

System Architecture

Layers:

Physical

Collects raw sensor data (temperature, pressure, flow, vibration) from the thermal system using industrial sensors.



Edge

Uses a Raspberry Pi to preprocess, timestamp, and simulate real-time sensor data using Python scripts

Network

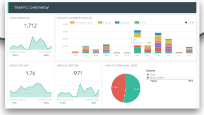
polling-based data pipeline over a direct TCP/IP connection to reliably send sensor data from the Raspberry Pi to the cloud.

Cloud

Stores and organizes all incoming sensor data in a PostgreSQL database for analysis.

Application

Uses a Streamlit dashboard to visualize real-time and historical data fetched through SQL queries.



Data Flow:

Sensors → Edge filtering → Polling-based TCP/IP transfer → Cloud ingestion → Dashboard visualization

Physical Device Layer

➢ The dataset represents a Thermal Fluid Heating and Circulation System commonly used in chemical processing industries. This type of system is designed to heat, move, and regulate thermal fluids as they circulate through a closed industrial loop.

Thermal Fluid Heating and Circulation System

- 🔥 **Heater Unit**
- 🧩 **Pipeline Network**
- 💧 **Centrifugal Pump**
- 📡 **Sensors**

System Components

- 🌡️ **Temperature**
- ⚡ **Pressure**
- 💨 **Flow Rate**
- 📳 **Vibration**
- 🔌 **Current & Voltage**
- 🎛️ **FFT Frequency Comp.**
- 🟣 **Fault_Type Labels**

Data collected by sensors

Temperatu	Vibration	Pressure	Flow_Rate	Current	Voltage	FFT_Temp_	FFT_Vib_0	FFT_Pres_	FFT_Temp_	FFT_Vib_1
46.00614	2.038362	56.77577	6.184385	12.40952	215.7624	772.4031	32.43654	971.8053	3.76064	0.734033
62.52917	2.573668	76.15984	8.27923	14.90639	215.4659	767.6024	32.39556	962.4815	8.365137	0.724559
77.29501	3.243491	92.37261	9.172789	15.05405	202.0436	765.9651	32.03204	956.2995	9.559769	0.934401
76.56416	3.142904	94.14956	13.77538	16.41789	216.6991	763.9364	33.03985	956.5322	10.25187	0.169234
78.28164	3.13996	94.44101	11.11311	10.89942	227.3283	746.7549	33.01873	950.2128	18.31311	0.172678
78.40139	3.197514	94.39167	7.329037	16.7777	207.2577	739.3011	33.5376	977.4437	22.06954	0.405082
76.40888	3.367163	96.32608	10.97207	6.886825	230.4848	737.6603	33.43981	973.3725	22.21722	0.353825
77.1856	3.243817	101.9885	6.905392	13.11035	224.8777	724.7001	32.66651	970.7153	19.61634	0.909296
76.95632	3.336208	98.34675	12.16538	13.53518	212.6577	722.4342	32.10698	958.8476	18.61197	1.45917
74.91137	3.266448	96.81889	9.057751	16.89998	218.5847	720.2414	31.42203	958.6093	16.76559	2.00669
74.3859	3.410457	96.91478	9.812728	15.06789	235.9832	710.4671	31.90173	936.841	6.998141	1.86856
72.94209	3.236583	93.71648	12.65159	10.78331	227.3355	720.1124	31.50445	953.5159	15.69829	1.730284
70.67462	3.463703	93.50029	7.425673	12.04502	220.0878	721.3122	31.1188	963.917	16.45764	1.454527
70.57575	3.351755	96.32776	7.205764	15.59604	217.6232	720.44	30.06899	985.215	16.40614	0.497414
70.02865	3.266853	97.32418	8.832801	14.7628	220.7661	722.8827	31.48472	981.5258	15.17081	1.910542

FFT_Pres_	FFT_Temp_	FFT_Vib_2	FFT_Pres_	FFT_Temp_	FFT_Vib_3	FFT_Pres_	FFT_Temp_	FFT_Vib_4	FFT_Pres_	FFT_Temp_	FFT_Vib_5	FFT_Pres_	FFT_Temp_	FFT_Vib_
30.87457	8.787639	1.157286	4.368856	22.08854	1.289962	33.97044	2.527477	0.436747	23.42856	2.203178	1.665418	25.82057	2.527477	0.43674
27.2683	12.55352	1.130837	8.104435	19.49031	1.283113	41.3641	6.938895	0.413276	25.07217	7.003837	1.624437	35.14438	6.938895	0.41327
30.13713	12.29183	0.80343	13.26244	21.10899	0.972256	43.41712	5.463529	0.534189	20.17911	5.36651	1.987954	28.96232	5.463529	0.53418
29.91994	10.32111	1.13987	13.29343	20.86373	0.606251	43.64891	6.577441	1.168588	19.94796	7.395141	2.99577	28.72955	6.577441	1.16858
36.06507	14.44165	1.149088	7.149397	8.1633	0.627348	45.46179	17.84783	1.178281	14.75487	9.786409	3.016895	22.41013	17.84783	1.17828
27.10381	20.97853	0.640159	32.00788	15.6124	0.912349	29.51868	10.79203	1.217577	27.95374	2.332587	3.535765	4.820764	10.79203	1.21757
24.5188	20.88911	0.652452	31.72265	15.1446	0.877791	25.44933	11.68991	1.188867	32.01131	3.97334	3.633553	8.891963	11.68991	1.18886
22.00968	9.319991	0.381416	34.25488	7.938514	1.651093	26.47012	17.36035	1.853913	30.03658	8.98688	2.860247	6.23476	17.36035	1.85391
11.05825	7.462173	0.433136	42.19094	10.1134	1.569692	36.04099	15.11026	1.312091	34.21856	6.720943	3.419776	18.1025	15.11026	1.31209
10.97327	8.518086	0.63809	42.09567	9.128676	0.932533	35.86363	17.06567	1.807329	34.24312	8.913691	2.734826	17.86422	17.06567	1.80732
26.89677	18.08101	0.920199	20.52783	6.290264	0.673267	33.53158	15.03314	2.01119	25.4299	0.860627	2.255125	39.63251	15.03314	2.0111

Sensor dataset (CSV)

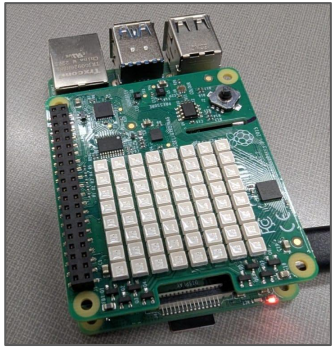
Network / Communication Layer

➢ Network Layer - Direct Edge-to-Database Communication

- Raspberry Pi connects to the **PostgreSQL** server using its **host IP address**
- No MQTT / No publish–subscribe broker
- Uses a **TCP/IP** client connection handled by the ***psycopg2*** driver
- **Real-time** sensor readings inserted into the database every 2 seconds
- Simple, low-latency, and reliable for prototype-scale IIoT systems

No MQTT broker needed → single edge device, no routing required

Direct DB insertion → simpler, stable, ideal for prototype-scale IIoT



Edge Device (Raspberry Pi)

TCP/IP
(Client Connection)

PostgreSQL

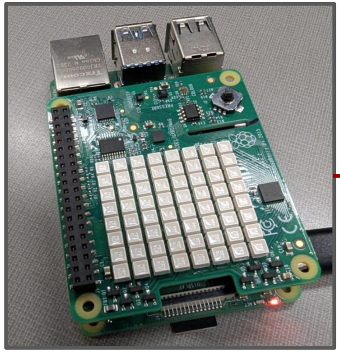
Cloud Storage Layer
(PostgreSQL)

```python
def make_conn():  1 usage
    return psycopg2.connect(
        host=DB_HOST,
        port=DB_PORT,
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASS
    )
```

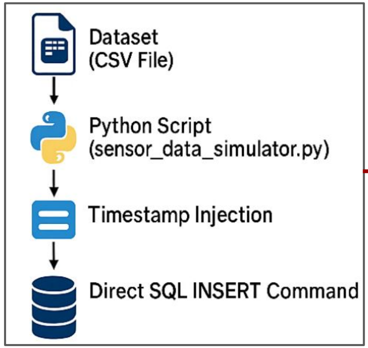
Python DB Connection (psycopg2)

Edge Layer

➢ Edge / Gateway Layer - Raspberry Pi as Real-Time Data Simulator



Raspberry pi (Edge Device)



Processing

PostgreSQL

Cloud Storage Layer
(PostgreSQL)

- Raspberry Pi simulates real-time sensor readings using sensor_data_simulator.py
- Reads dataset row-by-row and streams values in **2-second intervals**
- Generates **real timestamps** to mimic real industrial sensors
- Performs lightweight preprocessing (cleaning, formatting, type handling)
- Uses PostgreSQL connection (TCP/IP) to send data directly to the cloud

Edge Layer

publisher.py

```python
# simulator.py
import ...

# ====================================================
CSV_PATH = "dataset.csv"          # Path to dataset
EMIT_INTERVAL = 2.0               # Seconds between inserts
LOOP = True                       # Repeat dataset forever
BATCH_SIZE = 1                    # Insert one row at a time

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "IIOT_PROJECT"
DB_USER = "postgres"
DB_PASS = "1234"

TABLE = "iiot_measurements"
# ====================================================


INSERT_SQL_TEMPLATE = f"""
INSERT INTO {TABLE} (
    ts, temperature, vibration, pressure, flow_rate, current, voltage,
    fft_temp_0, fft_vib_0, fft_pres_0,
    fft_temp_1, fft_vib_1, fft_pres_1,
    fft_temp_2, fft_vib_2, fft_pres_2,
    fft_temp_3, fft_vib_3, fft_pres_3,
    fft_temp_4, fft_vib_4, fft_pres_4,
    fft_temp_5, fft_vib_5, fft_pres_5,
    fft_temp_6, fft_vib_6, fft_pres_6,
    fft_temp_7, fft_vib_7, fft_pres_7,
    fft_temp_8, fft_vib_8, fft_pres_8,
    fft_temp_9, fft_vib_9, fft_pres_9,
    fault_type
) VALUES %s
"""


def make_conn():  1 usage
    return psycopg2.connect(
        host=DB_HOST,
        port=DB_PORT,
```

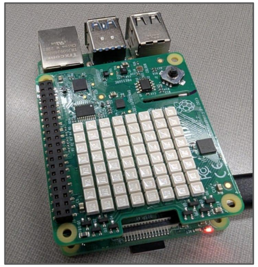
```python
def make_conn():  1 usage
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASS
    )


def row_tuple_from_series(s):  1 usage
    ts = datetime.now(timezone.utc)  # Use real streaming timestamp

    def get(col):
        v = s.get(col)
        if pd.isna(v):
            return None
        try:
            return float(v)
        except:
            return v  # for Fault_Type text

    # Build the tuple in column order
    return (
        ts,
        get("Temperature"), get("Vibration"), get("Pressure"), get("Flow_Rate"),
        get("Current"), get("Voltage"),

        get("FFT_Temp_0"), get("FFT_Vib_0"), get("FFT_Pres_0"),
        get("FFT_Temp_1"), get("FFT_Vib_1"), get("FFT_Pres_1"),
        get("FFT_Temp_2"), get("FFT_Vib_2"), get("FFT_Pres_2"),
        get("FFT_Temp_3"), get("FFT_Vib_3"), get("FFT_Pres_3"),
        get("FFT_Temp_4"), get("FFT_Vib_4"), get("FFT_Pres_4"),
        get("FFT_Temp_5"), get("FFT_Vib_5"), get("FFT_Pres_5"),
        get("FFT_Temp_6"), get("FFT_Vib_6"), get("FFT_Pres_6"),
        get("FFT_Temp_7"), get("FFT_Vib_7"), get("FFT_Pres_7"),
        get("FFT_Temp_8"), get("FFT_Vib_8"), get("FFT_Pres_8"),
        get("FFT_Temp_9"), get("FFT_Vib_9"), get("FFT_Pres_9"),

        get("Fault_Type")
    )


def main():  1 usage
```

```python
def main():  1 usage
    # Load CSV once
    df = pd.read_csv(CSV_PATH)
    rows = df.shape[0]

    conn = make_conn()
    cur = conn.cursor()

    print(f"Loaded {rows} rows. Starting simulation...")

    idx = 0

    while True:
        batch = []

        for _ in range(BATCH_SIZE):
            s = df.iloc[idx % rows]
            batch.append(row_tuple_from_series(s))
            idx += 1

        execute_values(cur, INSERT_SQL_TEMPLATE, batch)
        conn.commit()

        print(f"[{datetime.now().isoformat()}] Inserted {len(batch)} row(s)")

        time.sleep(EMIT_INTERVAL)

        if not LOOP and idx >= rows:
            print("Dataset completed – exiting.")
            break

    cur.close()
    conn.close()


if __name__ == "__main__":
    main()
```

Python script on the Raspberry Pi timestamps, preprocesses, formats, and sends each dataset row to PostgreSQL every 2 seconds to simulate real-time sensor data.

Cloud Layer

➢ **PostgreSQL as Cloud Storage & Analytics Engine**



Edge Device (Raspberry Pi)

TCP/IP (Client Connection)

SQL queries for dashboard updates

Application Layer

The cloud layer receives timestamped sensor data directly over TCP/IP and stores it for real-time visualization and analytics

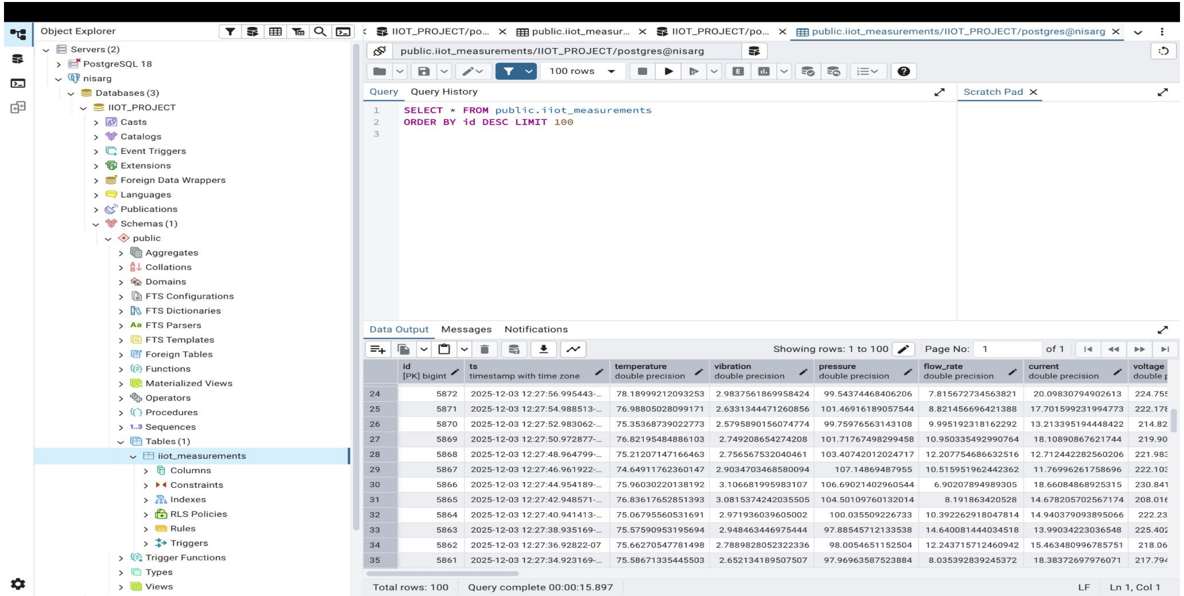
➢ **Why PostgreSQL?**
- Receives real-time data from the Pi.
- Stores the full multivariate dataset.
- Organizes data into structured schema
- Serves as the backend for dashboards.

➢ **Cloud Layer Responsibilities**
- Data ingestion & storage
- Time-series retrieval
- Historical trend analysis
- Supports predictive insights
- Centralized IIoT data repository

Cloud Layer

➢ **Cloud Database:**

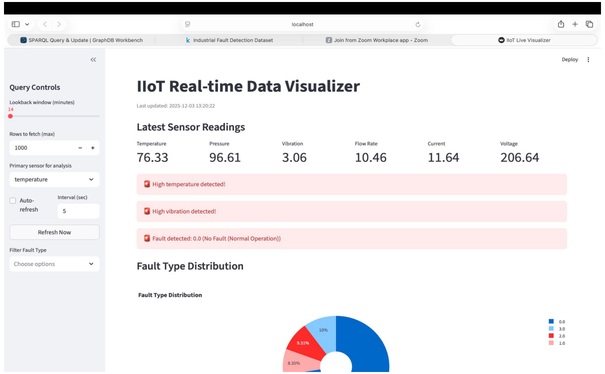
- Stores timestamped thermal, pressure, flow, vibration, electrical, and FFT data
- Supports reliable querying for dashboard analytics

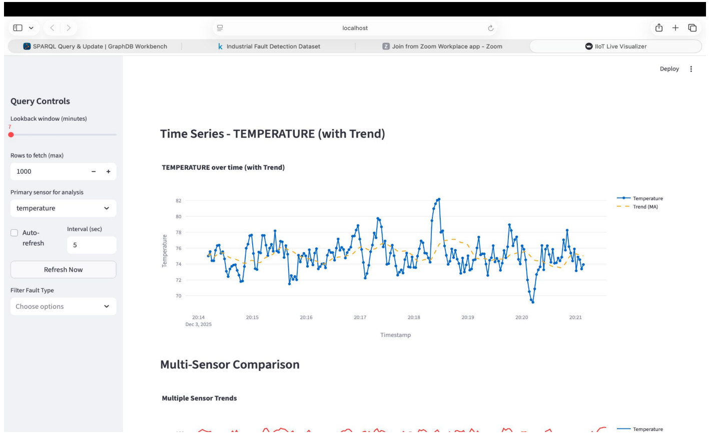


Cloud Layer

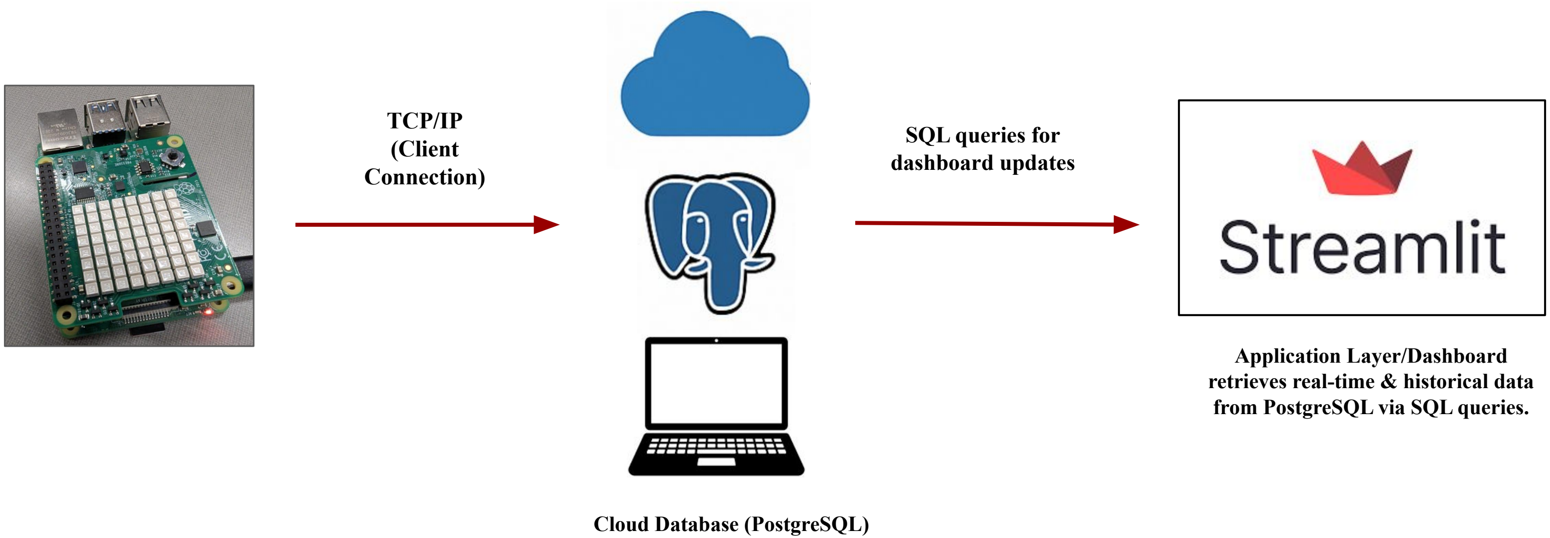
Cloud Database: Analytics Performed:

- Time-series trend visualization
- Fault_Type-based status monitoring
- Historical data retrieval for predictive insights





Application Layer



Application Layer

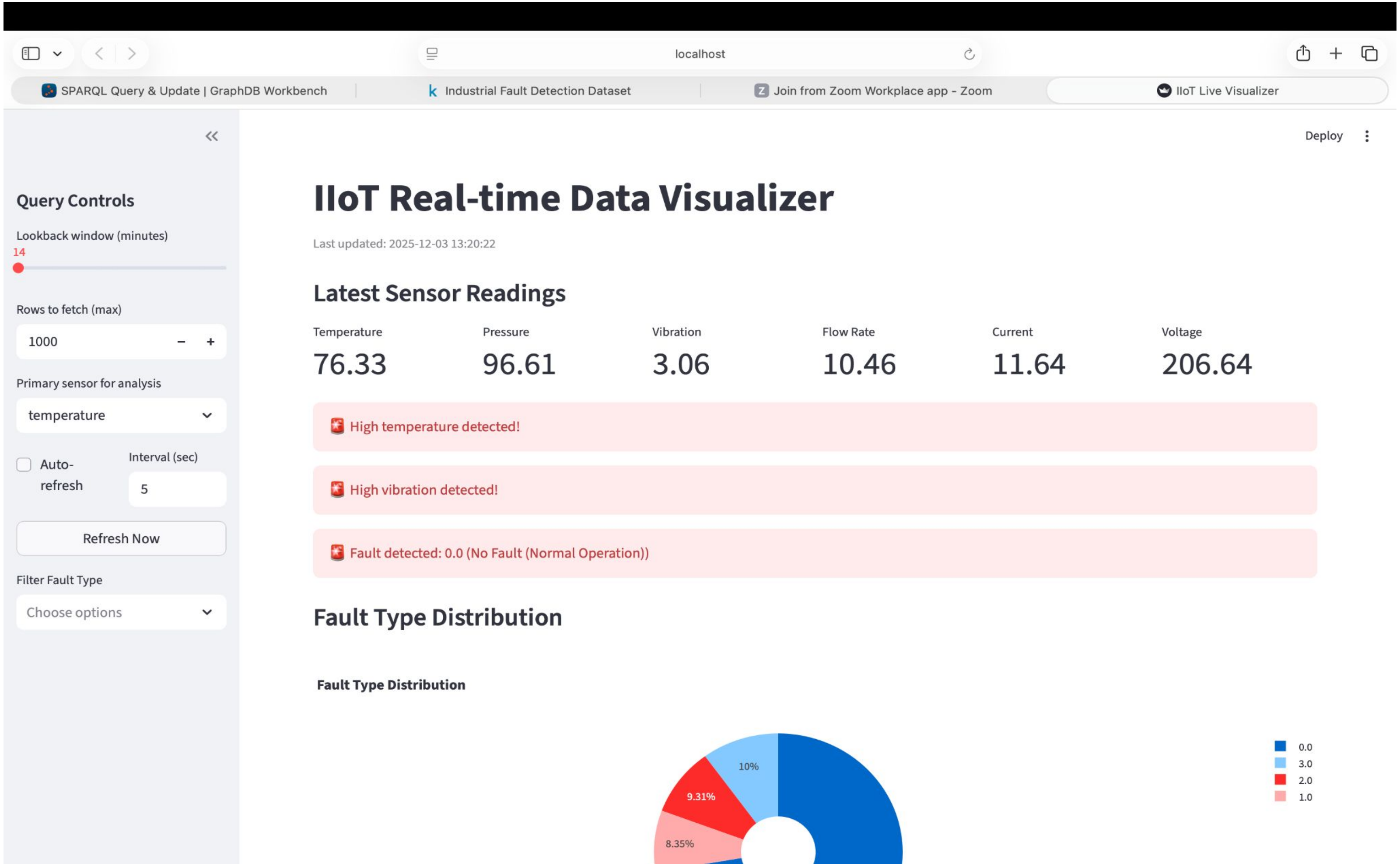
SQL queries for
dashboard updates

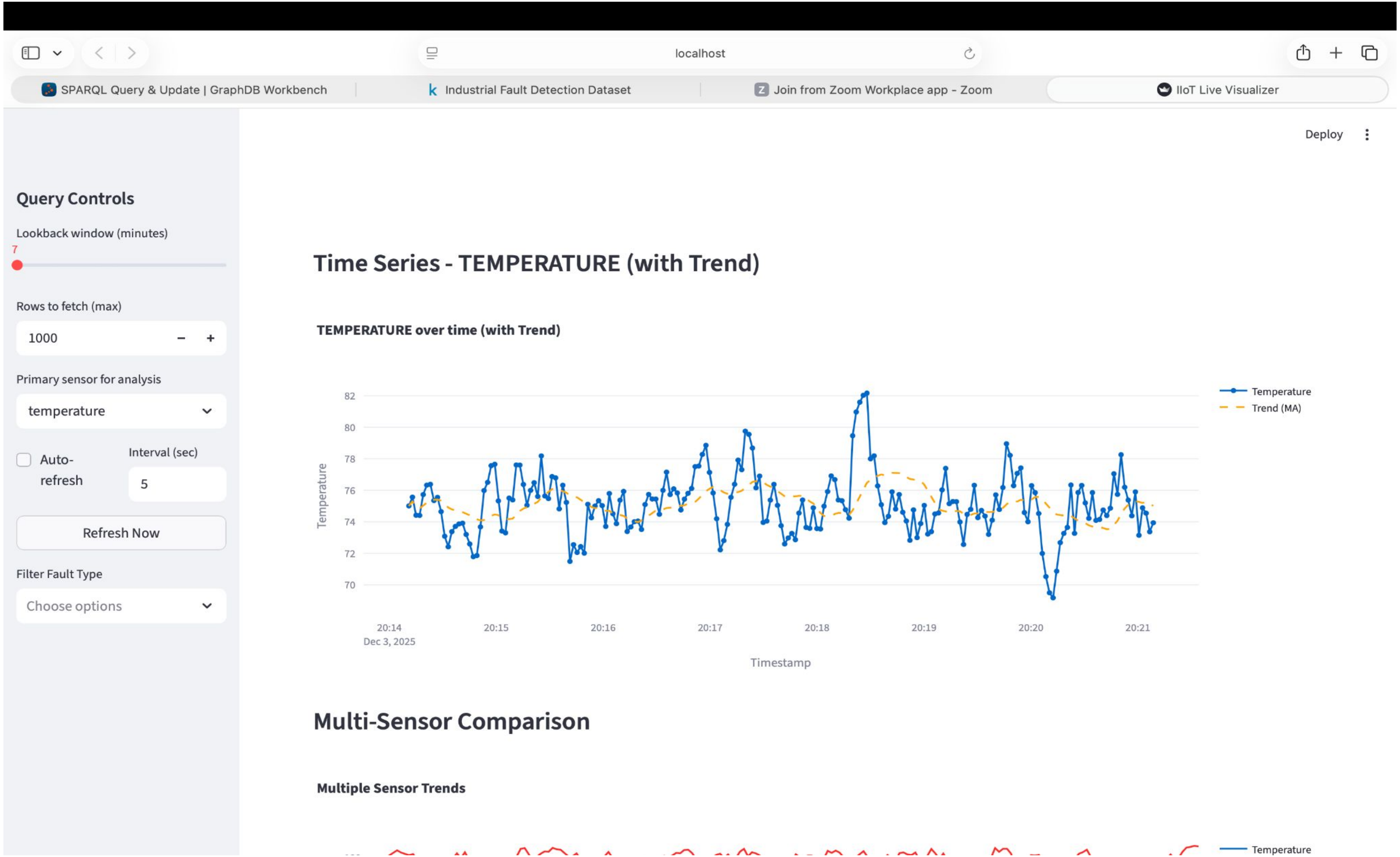
Cloud Database (PostgreSQL)

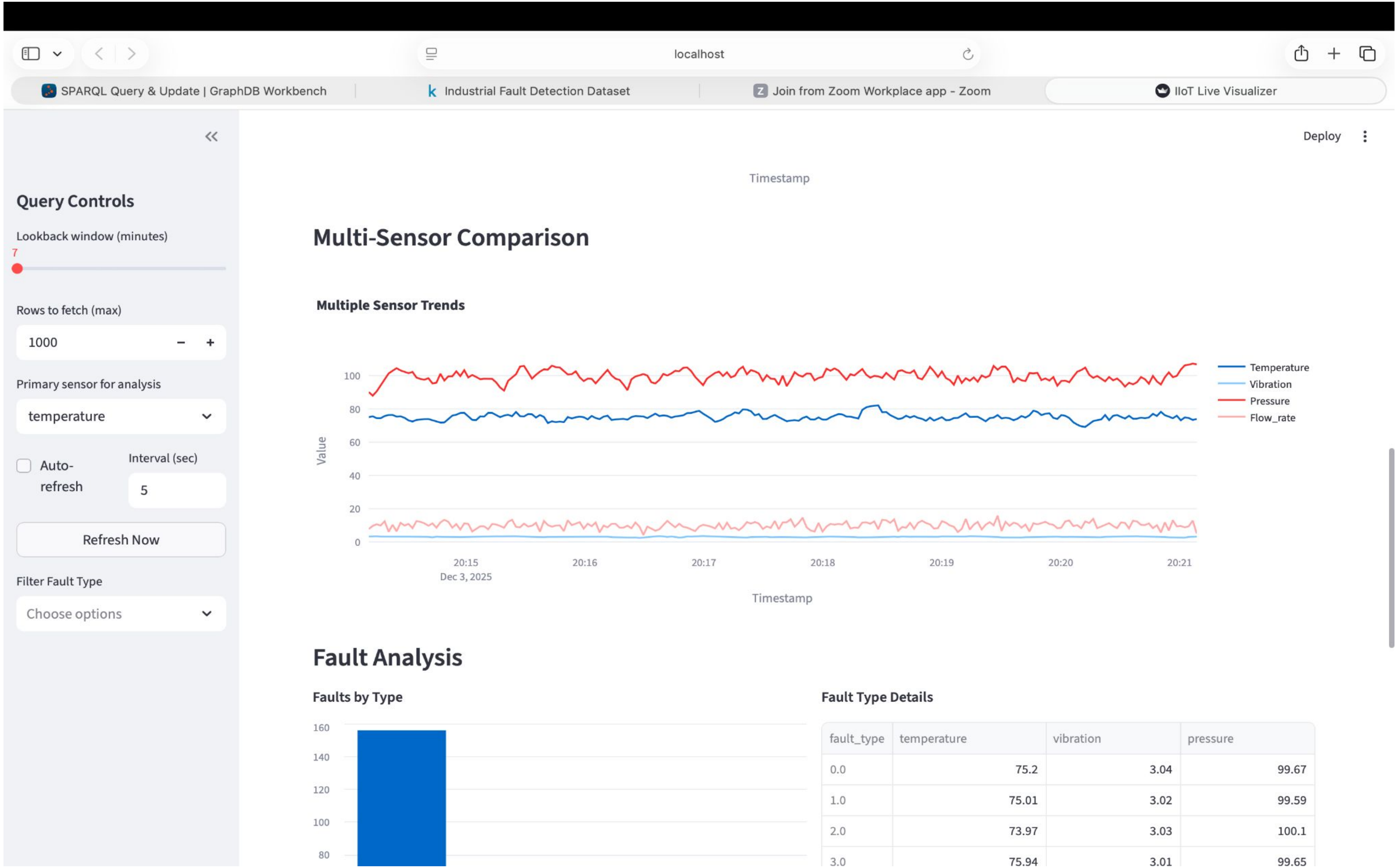
Streamlit

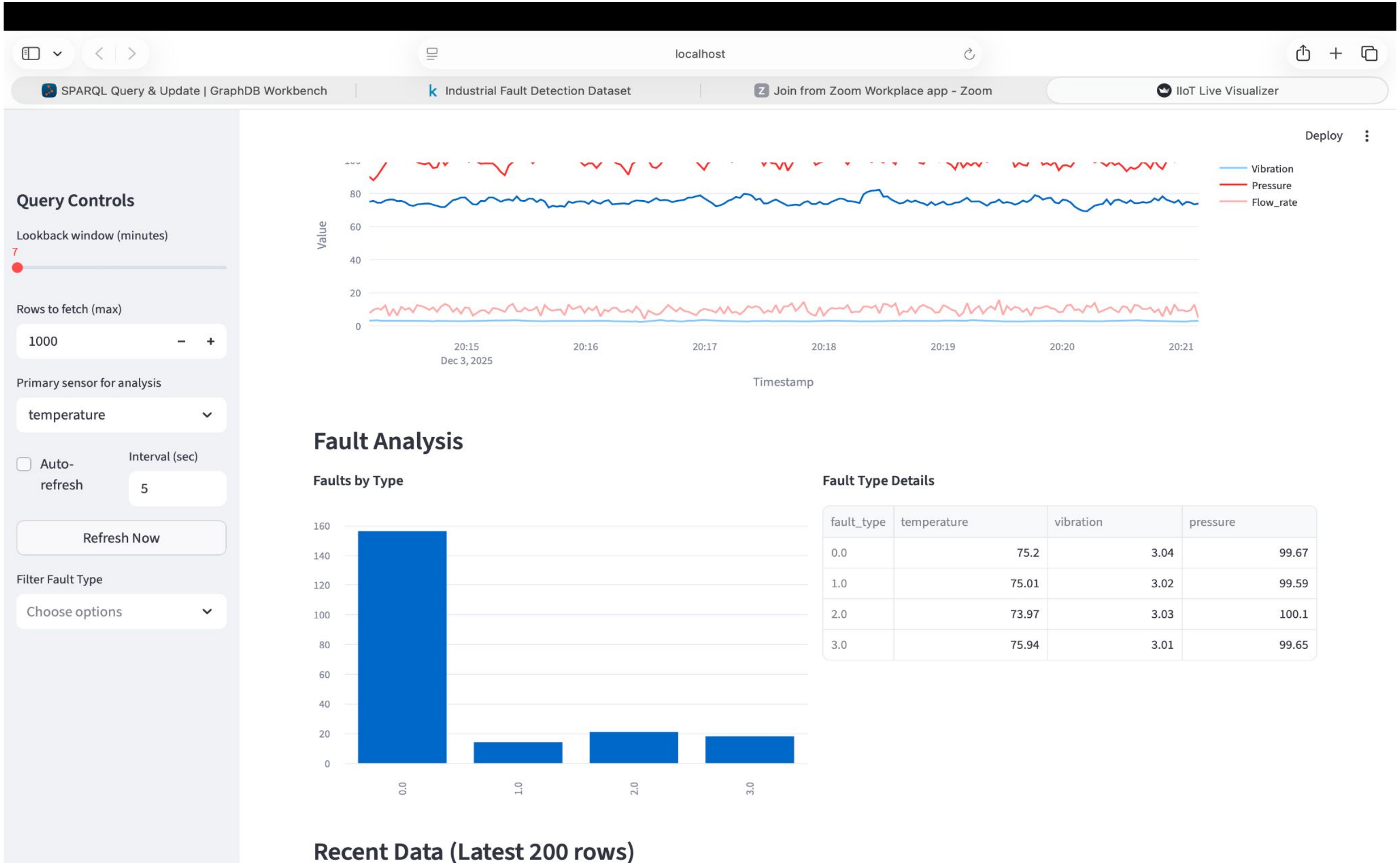
Application Layer/Dashboard
retrieves real-time & historical data
from PostgreSQL via SQL queries.

```python
def fetch_recent_rows(lookback_minutes: int, limit_rows: int, fault_filter_list=None) -> pd.DataFrame:
    """Fetch recent data from database with optional fault filtering"""
    engine = get_engine()


    if fault_filter_list and len(fault_filter_list) > 0:

        placeholders = ",".join([f"'{ft}'" for ft in fault_filter_list])
        sql = f"""
        SELECT * FROM {TABLE}
        WHERE ts >= NOW() - INTERVAL '{lookback_minutes} minutes' AND fault_type IN ({placeholders})
        ORDER BY ts DESC
        LIMIT :limit
        """
    else:
        sql = f"""
        SELECT * FROM {TABLE}
        WHERE ts >= NOW() - INTERVAL '{lookback_minutes} minutes'
        ORDER BY ts DESC
        LIMIT :limit
        """


    with engine.connect() as conn:
        df = pd.read_sql(text(sql), conn, params={"limit": limit_rows})


    df.columns = [c.lower() for c in df.columns]
    if "ts" in df.columns:
        df["ts"] = pd.to_datetime(df["ts"])
    return df
```


localhost
SPARQL Query & Update | GraphDB Workbench
Industrial Fault Detection Dataset
Join from Zoom Workplace app - Zoom
IIoT Live Visualizer
Deploy
Query Controls
Lookback window (minutes)
14
Rows to fetch (max)
1000
Primary sensor for analysis
temperature
Auto-refresh
Interval (sec)
5
Refresh Now
Filter Fault Type
Choose options
IIoT Real-time Data Visualizer
Last updated: 2025-12-03 13:20:22
Latest Sensor Readings
Temperature
76.33
Pressure
96.61
Vibration
3.06
Flow Rate
10.46
Current
11.64
Voltage
206.64
High temperature detected!
High vibration detected!
Fault detected: 0.0 (No Fault (Normal Operation))
Fault Type Distribution
Fault Type Distribution
10%
9.31%
8.35%
0.0
3.0
2.0
1.0


localhost
SPARQL Query & Update | GraphDB Workbench
Industrial Fault Detection Dataset
Join from Zoom Workplace app - Zoom
IIoT Live Visualizer
Deploy
Query Controls
Lookback window (minutes)
7
Rows to fetch (max)
1000
Primary sensor for analysis
temperature
Auto-refresh
Interval (sec)
5
Refresh Now
Filter Fault Type
Choose options
Time Series - TEMPERATURE (with Trend)
TEMPERATURE over time (with Trend)
Temperature
Trend (MA)
82
80
78
76
74
72
70
20:14
Dec 3, 2025
20:15
20:16
20:17
20:18
20:19
20:20
20:21
Timestamp
Multi-Sensor Comparison
Multiple Sensor Trends
Temperature


localhost
SPARQL Query & Update | GraphDB Workbench
Industrial Fault Detection Dataset
Join from Zoom Workplace app - Zoom
IIoT Live Visualizer
Deploy
Query Controls
Lookback window (minutes)
7
Rows to fetch (max)
1000
Primary sensor for analysis
temperature
Auto-refresh
Interval (sec)
5
Refresh Now
Filter Fault Type
Choose options
Timestamp
Multi-Sensor Comparison
Multiple Sensor Trends
Value
Temperature
Vibration
Pressure
Flow_rate
20:15
Dec 3, 2025
20:16
20:17
20:18
20:19
20:20
20:21
Timestamp
Fault Analysis
Faults by Type
Fault Type Details
fault_type
temperature
vibration
pressure
0.0 75.2 3.04 99.67
1.0 75.01 3.02 99.59
2.0 73.97 3.03 100.1
3.0 75.94 3.01 99.65



fault_type	temperature	vibration	pressure
0.0	75.2	3.04	99.67
1.0	75.01	3.02	99.59
2.0	73.97	3.03	100.1
3.0	75.94	3.01	99.65

localhost

SPARQL Query & Update | GraphDB Workbench | Industrial Fault Detection Dataset | Join from Zoom Workplace app - Zoom | IIoT Live Visualizer

Deploy

Query Controls

Lookback window (minutes)

7

Rows to fetch (max)

1000

Primary sensor for analysis

temperature

Auto-refresh

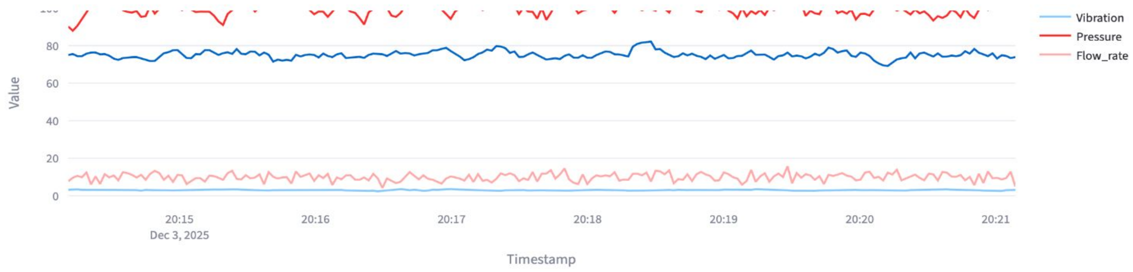
Interval (sec)

5

Refresh Now

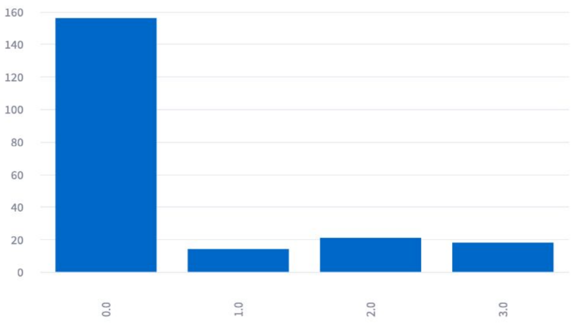
Filter Fault Type

Choose options



Fault Analysis

Faults by Type



Fault Type Details

fault_type	temperature	vibration	pressure
0.0	75.2	3.04	99.67
1.0	75.01	3.02	99.59
2.0	73.97	3.03	100.1
3.0	75.94	3.01	99.65

Recent Data (Latest 200 rows)

localhost

SPARQL Query & Update | GraphDB Workbench | Industrial Fault Detection Dataset | Join from Zoom Workplace app - Zoom | IIoT Live Visualizer

Deploy

Query Controls

Lookback window (minutes)

7

Rows to fetch (max)

1000

Primary sensor for analysis

temperature

Auto-refresh

Interval (sec)

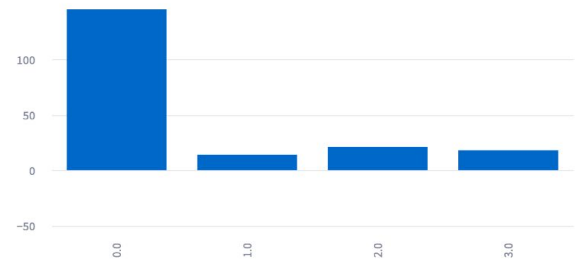
5

Refresh Now

Filter Fault Type

Choose options

- Leakage Fault
- No Fault (Normal Operation)
- Overheating Fault
- Power Fluctuation Fault

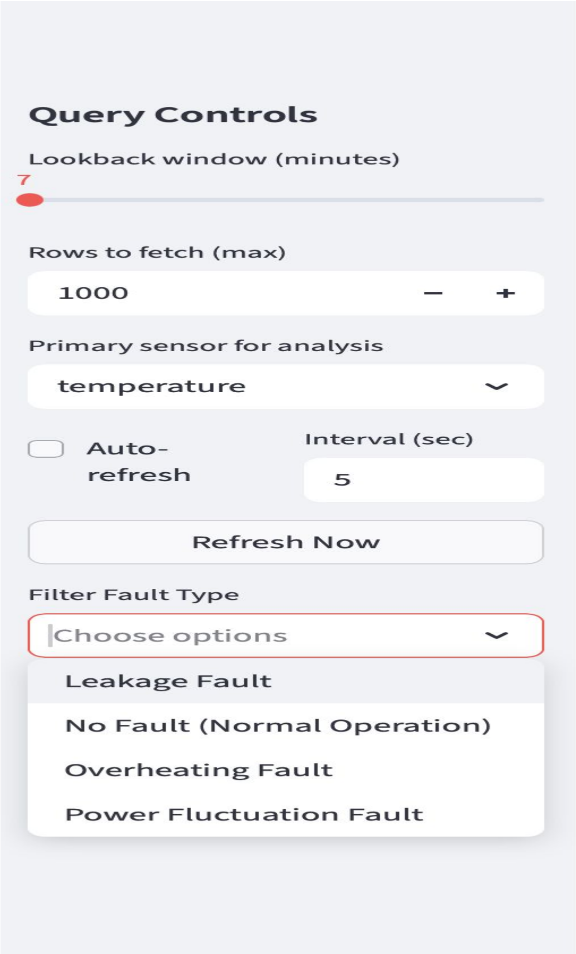


1.0	75.01	3.02	99.59
2.0	73.97	3.03	100.1
3.0	75.94	3.01	99.65

Recent Data (Latest 200 rows)

	id	ts	temperature	vibration	pressure	flow_rate	current	voltage	fft_temp_0	fft_vib_0	fft_pres_0	fft_temp_1	fft_vib_1	fft_
0	7460	2025-12-03 20:21:08+00:00	73.94	3.16	106.83	5.02	13.71	224.57	756.2587	31.9291	1019.6729	11.1785	0.8955	
1	7459	2025-12-03 20:21:06+00:00	73.37	2.96	107.26	12.84	16.53	235.74	759.3631	32.4574	1026.5425	9.6587	1.2283	2
2	7458	2025-12-03 20:21:04+00:00	74.55	3.01	106.58	9.42	16.82	229.10	753.1014	31.2201	1055.797	10.1157	1.5276	5
3	7457	2025-12-03 20:21:02+00:00	74.89	2.58	106.19	8.74	19.53	213.95	750.4421	30.5347	1043.3174	9.8354	2.1958	3
4	7456	2025-12-03 20:21:00+00:00	73.14	2.41	103.64	9.58	15.56	233.63	751.8471	30.9339	1026.786	10.6919	1.8429	3
5	7455	2025-12-03 20:20:58+00:00	75.90	2.55	99.86	9.34	13.63	207.83	746.5902	29.5562	1030.4359	5.7093	2.7034	3.
6	7454	2025-12-03 20:20:56+00:00	74.38	2.75	99.10	13.01	15.61	224.16	740.2943	28.9401	1029.8314	1.2248	2.9265	3.
7	7453	2025-12-03 20:20:54+00:00	75.38	2.77	101.99	6.88	12.66	236.19	749.1744	28.5062	1033.4918	7.7436	2.8905	3
8	7452	2025-12-03 20:20:52+00:00	76.19	2.94	99.07	11.75	17.92	226.00	738.7373	28.5653	1031.7628	6.5996	2.926	3.
9	7451	2025-12-03 20:20:50+00:00	78.26	3.09	94.69	5.70	18.48	230.27	749.6142	28.9223	1042.8507	4.6188	3.2698	3.

Download Data as CSV


Query Controls
Lookback window (minutes)
7
Rows to fetch (max)
1000
Primary sensor for analysis
temperature
Auto-refresh
Interval (sec)
5
Refresh Now
Filter Fault Type
Choose options
Leakage Fault
No Fault (Normal Operation)
Overheating Fault
Power Fluctuation Fault

Key Outcomes, Challenges, and Insights

Key Outcomes:

- Successfully built an end-to-end IIoT pipeline using sensors → edge preprocessing → TCP/IP transfer → cloud database → real-time dashboard.
- Real-time data streaming achieved with reliable ingestion into PostgreSQL.
- Dashboard visualizations clearly show trends, fault distribution, and recent data with actionable alerts.
- Prototype demonstrates how predictive maintenance can detect overheating, vibration spikes, and abnormal pressure patterns early and required actions can be taken to improvise the process.

Challenges:

- Ensuring smooth timestamp synchronization during real-time simulation on the Raspberry Pi.
- Handling noisy sensor values and formatting issues before insertion into PostgreSQL.
- Maintaining stable, low-latency TCP/IP communication without data drops.
- Dashboard query optimization to avoid lag during large lookback windows or multiple sensor comparisons.

Key Outcomes, Challenges, and Insights

Insights:

- Edge preprocessing (filtering + cleaning) significantly improves data quality before storage.
- Polling-based TCP/IP transfer is simple and reliable for a single-device prototype, avoiding unnecessary complexity.
- Clear visualization helps catch operational anomalies faster than manual inspection.
- Industrial datasets with FFT features and Fault_Type labels greatly enhance predictive capabilities.

Conclusion

- This project demonstrates how IIoT can transform a traditional thermal fluid heating system into a smart, continuously monitored, and fault-adaptive system.

- By combining edge processing, efficient data transfer, cloud storage, and real-time dashboards, the system enables early detection of overheating, vibration abnormalities, and energy inefficiencies.

- The prototype proves that predictive maintenance can reduce downtime, improve safety, and optimize industrial operations.

- Future extensions include ML-based anomaly detection, automated control actions, energy optimization insights, and scaling the system across multiple industrial units.

Thank you!