# Integrate Machine Learning into PL/SQL Forecasting

Instead of relying solely on **PL/SQL-based moving averages**, we can use a **Python-based ML model** to analyze historical workload data and make more **accurate workload predictions**.

**New Approach: Machine Learning-Driven Workload Forecasting**

1. **Train an ML model (Python) on historical system performance data.**
2. **Store the predictions in Oracle.**
3. **Modify PL/SQL procedures to fetch and use the ML-based predictions.**
4. **Use Python and Oracle Machine Learning (OML) or REST APIs to integrate predictions into the chatbot.**

**Step 1: Train the Machine Learning Model (Python)**

We will use **Python with Scikit-learn** to train a **time-series forecasting model** using **past CPU, memory, and session usage data**.

**Python Code for Training the Model**

model\_trainer.py

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

import cx\_Oracle

# Connect to Oracle and fetch historical workload data

dsn = cx\_Oracle.makedsn("DB\_HOST", 1521, service\_name="DB\_SERVICE")

conn = cx\_Oracle.connect(user="DB\_USER", password="DB\_PASS", dsn=dsn)

query = """

SELECT collection\_time, cpu\_usage, active\_sessions, memory\_usage

FROM system\_performance\_metrics

WHERE collection\_time >= SYSDATE - 30

ORDER BY collection\_time

"""

df = pd.read\_sql(query, conn)

conn.close()

# Convert timestamps to numeric values for ML training

df['timestamp'] = (df['collection\_time'] - df['collection\_time'].min()).dt.total\_seconds()

# Train a Linear Regression Model for CPU Usage

X = df[['timestamp']]

y\_cpu = df['cpu\_usage']

cpu\_model = LinearRegression()

cpu\_model.fit(X, y\_cpu)

# Train a Linear Regression Model for Sessions

y\_sessions = df['active\_sessions']

sessions\_model = LinearRegression()

sessions\_model.fit(X, y\_sessions)

# Train a Linear Regression Model for Memory Usage

y\_memory = df['memory\_usage']

memory\_model = LinearRegression()

memory\_model.fit(X, y\_memory)

print("ML Models Trained Successfully!")

**Step 2: Generate Predictions and Store in Oracle**

We now use the trained models to **predict future workload values** and insert them into the workload\_forecast table.

model\_predict.py

from datetime import datetime, timedelta

# Predict future workload 1 hour ahead

future\_time = (datetime.now() - df['collection\_time'].min()).total\_seconds() + 3600

predicted\_cpu = cpu\_model.predict([[future\_time]])[0]

predicted\_sessions = sessions\_model.predict([[future\_time]])[0]

predicted\_memory = memory\_model.predict([[future\_time]])[0]

# Insert the predictions into the Oracle database

conn = cx\_Oracle.connect(user="DB\_USER", password="DB\_PASS", dsn=dsn)

cursor = conn.cursor()

insert\_query = """

INSERT INTO workload\_forecast (metric\_name, predicted\_value, forecast\_time)

VALUES (:1, :2, SYSTIMESTAMP)

"""

cursor.execute(insert\_query, ('CPU Usage (%)', predicted\_cpu))

cursor.execute(insert\_query, ('Active Sessions', predicted\_sessions))

cursor.execute(insert\_query, ('SGA Memory (MB)', predicted\_memory))

conn.commit()

conn.close()

print("Predictions Inserted into Oracle Successfully!")

**Step 3: Modify PL/SQL Procedures to Use ML-Based Predictions**

Now that **Python-based predictions** are stored in workload\_forecast, we modify the chatbot’s logic to **fetch ML-based values** instead of using moving averages.

fetch\_chatbot\_forecast\_response.sql

CREATE OR REPLACE PROCEDURE chatbot\_forecast\_response(

    p\_user\_query IN VARCHAR2,

    p\_response OUT CLOB

) IS

    v\_predicted\_cpu NUMBER;

    v\_predicted\_sessions NUMBER;

    v\_alert\_msg VARCHAR2(500);

    v\_advice VARCHAR2(500);

BEGIN

    -- Fetch latest ML-based predictions

    SELECT predicted\_value INTO v\_predicted\_cpu

    FROM workload\_forecast

    WHERE metric\_name = 'CPU Usage (%)'

    ORDER BY forecast\_time DESC FETCH FIRST 1 ROW ONLY;

    SELECT predicted\_value INTO v\_predicted\_sessions

    FROM workload\_forecast

    WHERE metric\_name = 'Active Sessions'

    ORDER BY forecast\_time DESC FETCH FIRST 1 ROW ONLY;

    -- Check for future overload risk

    IF v\_predicted\_cpu > 85 THEN

        v\_alert\_msg := 'High CPU usage predicted: ' || v\_predicted\_cpu || '%.';

        v\_advice := 'Consider increasing CPU capacity or tuning resource-intensive queries.';

    ELSIF v\_predicted\_sessions > 220 THEN

        v\_alert\_msg := 'Surge in active sessions expected: ' || v\_predicted\_sessions || '.';

        v\_advice := 'Check connection pooling and optimize long-running transactions.';

    ELSE

        v\_alert\_msg := 'No critical workload spikes predicted.';

        v\_advice := 'Keep monitoring system performance.';

    END IF;

    -- Generate chatbot response

    p\_response := v\_alert\_msg || ' Suggested action: ' || v\_advice;

END chatbot\_forecast\_response;

/

**Step 4: Automate ML-Based Forecasting Using DBMS\_SCHEDULER**

Now, we schedule a **Python script execution** every **30 minutes** using Oracle **DBMS\_SCHEDULER**.

forecast\_scheduler.sql

BEGIN

    DBMS\_SCHEDULER.create\_job (

        job\_name        => 'ML\_FORECAST\_JOB',

        job\_type        => 'EXECUTABLE',

        job\_action      => '/usr/bin/python3 /path/to/ml\_forecast.py',

        start\_date      => SYSTIMESTAMP,

        repeat\_interval => 'FREQ=MINUTELY; INTERVAL=30',

        enabled         => TRUE

    );

END;

/

**Final Outcome**

**Machine learning-based predictions replace simple moving averages.**  
**More accurate workload forecasts prevent performance bottlenecks.**  
**The chatbot now provides AI-driven insights to DBAs.**  
**Automated Python scripts insert ML-based predictions into Oracle every 30 minutes.**  
**DBAs receive real-time alerts based on advanced forecasting.**