# Integrate machine learning (ML) models to enhance workload forecasting accuracy

This involves:

1. **Using Python-based regression models** (e.g., linear regression, decision trees, or deep learning) to analyze past performance metrics and predict future workload trends.
2. **Integrating ML models with PL/SQL** to allow seamless communication between Oracle and Python.
3. **Enhancing the chatbot** to provide more intelligent recommendations based on AI-driven forecasts.

**Step 1: Prepare Historical Data for ML Models**

First, we'll extract historical workload metrics from Oracle and store them in a format that Python can use for training the ML models.

**Create a Table for Historical Data Extraction**

create\_table\_workload\_history.sql

CREATE TABLE workload\_history AS

SELECT metric\_name, metric\_value, collection\_time

FROM system\_performance\_metrics

WHERE collection\_time >= SYSTIMESTAMP - INTERVAL '30' DAY;

This stores the last 30 days of performance data, which our ML models will use to train and predict workload trends.

**Step 2: Train a Machine Learning Model in Python**

We’ll train a regression model (e.g., XGBoost, LSTM, or Prophet) using this data.

**Python Code for Training an ML Model**

ml\_model\_trainer.py

import pandas as pd

import cx\_Oracle

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

import joblib

# Connect to Oracle Database

dsn = cx\_Oracle.makedsn("your\_db\_host", "your\_db\_port", "your\_db\_service")

conn = cx\_Oracle.connect("your\_username", "your\_password", dsn)

query = "SELECT metric\_name, metric\_value, collection\_time FROM workload\_history"

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

conn.close()

# Convert timestamps to numerical values

df['timestamp'] = pd.to\_datetime(df['collection\_time']).astype(int) / 10\*\*9  # Convert to UNIX time

# Prepare features and labels

X = df[['timestamp']]

y = df['metric\_value']

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a model (RandomForest in this case)

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Save the trained model

joblib.dump(model, "workload\_forecast\_model.pkl")

This script:  
Connects to Oracle, extracts historical data, and converts timestamps into numerical values.  
Trains a **Random Forest Regressor** on workload patterns.  
Saves the trained model for future predictions.

**Step 3: Deploy the Model for Predictions**

We now create a Python API that Oracle can call for real-time predictions.

**Flask API for Oracle Integration**

api\_oracle.py

from flask import Flask, request, jsonify

import joblib

import numpy as np

app = Flask(\_\_name\_\_)

# Load trained model

model = joblib.load("workload\_forecast\_model.pkl")

@app.route('/predict', methods=['POST'])

def predict():

    data = request.json

    timestamp = np.array(data['timestamp']).reshape(-1, 1)

    prediction = model.predict(timestamp)

    return jsonify({'predicted\_value': prediction.tolist()})

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5000)

This sets up an API endpoint (/predict) where Oracle can send a timestamp, and the trained model will return a forecasted workload value.

**Step 4: Integrate ML Predictions into Oracle**

Now, we create a PL/SQL function that calls this Python API.

call\_api\_function.sql

CREATE OR REPLACE FUNCTION ml\_forecast\_metric (

    p\_metric\_name VARCHAR2,

    p\_timeframe INTERVAL DAY TO SECOND

) RETURN NUMBER IS

    v\_prediction NUMBER;

    v\_api\_url VARCHAR2(500) := 'http://your\_python\_server:5000/predict';

    v\_timestamp NUMBER;

    v\_response CLOB;

BEGIN

    -- Get current UNIX timestamp

    SELECT EXTRACT(EPOCH FROM SYSTIMESTAMP + p\_timeframe) INTO v\_timestamp FROM DUAL;

    -- Call Python API to get the prediction

    v\_response := http\_request(v\_api\_url, 'POST', '{"timestamp": ' || v\_timestamp || '}');

    -- Extract predicted value

    v\_prediction := JSON\_VALUE(v\_response, '$.predicted\_value');

    RETURN v\_prediction;

END ml\_forecast\_metric;

/

This function:  
Sends a request to the Python API with the future timestamp.  
Retrieves the AI-generated workload forecast.  
Returns the predicted value to Oracle.

**Step 5: Enhance the Chatbot with ML Predictions**

Now, the chatbot will use AI-driven forecasts instead of simple moving averages.

chatbot\_ml\_forecast\_response\_procedure.sql

CREATE OR REPLACE PROCEDURE chatbot\_ml\_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

    -- Get AI-powered workload forecasts

    v\_predicted\_cpu := ml\_forecast\_metric('CPU Usage (%)', INTERVAL '1' HOUR);

    v\_predicted\_sessions := ml\_forecast\_metric('Active Sessions', INTERVAL '1' HOUR);

    -- Check for overload risks

    IF v\_predicted\_cpu > 80 THEN

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

        v\_advice := 'Consider scaling up resources or optimizing SQL execution plans.';

    ELSIF v\_predicted\_sessions > 200 THEN

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

        v\_advice := 'Check application load balancing and tune long-running queries.';

    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\_ml\_forecast\_response;

/

Now, the chatbot can intelligently predict and respond to workload concerns using AI forecasts.

**Step 6: Automate AI-Based Forecasting**

Schedule AI-driven workload forecasting every **30 minutes**.

ai\_workload\_scheduler.sql

BEGIN

    DBMS\_SCHEDULER.create\_job (

        job\_name        => 'GENERATE\_AI\_FORECAST\_JOB',

        job\_type        => 'PLSQL\_BLOCK',

        job\_action      => 'BEGIN generate\_workload\_forecast; END;',

        start\_date      => SYSTIMESTAMP,

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

        enabled         => TRUE

    );

END;

/

**Final Outcome**

**AI-powered workload forecasting** with Python-based models.  
**More accurate predictions** using machine learning instead of simple averages.  
**Enhanced chatbot intelligence** that warns about future database slowdowns.  
**Proactive alerting system** that notifies DBAs before performance issues occur.