# Implementing Transformer-Based Oracle Workload Forecasting

To enhance workload forecasting, we will use **Transformer-based models**, which are superior to LSTMs for capturing **long-range dependencies** in time-series data.

We will:  
Use **Temporal Fusion Transformers (TFT)** or **GPT-like models** for workload prediction.  
Train a **Transformer model on Oracle workload data**.  
Deploy it via an **API for real-time forecasting**.  
**Integrate it into Oracle** for **proactive performance management**.

**Step 1: Prepare Time-Series Data for Transformer Training**

Transformers require **sequential input** with **positional encoding** to understand time-based patterns.

**Modify SQL for Feature-Rich Dataset**

We extract workload metrics **with timestamps, day-of-week, and hour-of-day** for richer context.

create\_table\_workload\_time\_series.sql

CREATE TABLE workload\_time\_series AS

SELECT metric\_name,

       metric\_value,

       collection\_time,

       EXTRACT(DAY FROM collection\_time) AS day\_of\_month,

       TO\_CHAR(collection\_time, 'D') AS day\_of\_week,

       EXTRACT(HOUR FROM collection\_time) AS hour\_of\_day

FROM system\_performance\_metrics

WHERE collection\_time >= SYSTIMESTAMP - INTERVAL '90' DAY

ORDER BY collection\_time;

This ensures:  
**Chronologically ordered** dataset.  
**Temporal features** (day, hour) for better predictions.

**Step 2: Train a Transformer Model for Oracle Workload Prediction**

**Install Required Libraries**

pip install torch transformers datasets scikit-learn joblib cx\_Oracle pandas

**Python Code for Transformer Model Training**

transformer\_model\_trainer.py

import pandas as pd

import numpy as np

import cx\_Oracle

import torch

from torch import nn, optim

from transformers import TimeSeriesTransformerModel, TimeSeriesTransformerConfig

from sklearn.preprocessing import MinMaxScaler

import joblib

# Connect to Oracle & Load Data

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\_value, day\_of\_week, hour\_of\_day FROM workload\_time\_series"

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

conn.close()

# Normalize values for stability

scaler = MinMaxScaler()

df[['metric\_value', 'day\_of\_week', 'hour\_of\_day']] = scaler.fit\_transform(df[['metric\_value', 'day\_of\_week', 'hour\_of\_day']])

# Convert into sequences

sequence\_length = 24  # Use past 24 readings for forecasting

X, y = [], []

for i in range(len(df) - sequence\_length):

    X.append(df[['metric\_value', 'day\_of\_week', 'hour\_of\_day']].iloc[i:i+sequence\_length].values)

    y.append(df['metric\_value'].iloc[i+sequence\_length])

X, y = np.array(X), np.array(y)

# Convert to PyTorch tensors

X\_train, y\_train = torch.tensor(X[:-1000], dtype=torch.float32), torch.tensor(y[:-1000], dtype=torch.float32)

X\_test, y\_test = torch.tensor(X[-1000:], dtype=torch.float32), torch.tensor(y[-1000:], dtype=torch.float32)

# Transformer Model Configuration

config = TimeSeriesTransformerConfig(

    d\_model=64,  # Hidden layer size

    n\_heads=4,   # Number of attention heads

    num\_encoder\_layers=3,

    num\_decoder\_layers=3,

    dropout=0.1

)

model = TimeSeriesTransformerModel(config)

# Training

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(50):

    model.train()

    optimizer.zero\_grad()

    output = model(X\_train).squeeze()

    loss = criterion(output, y\_train)

    loss.backward()

    optimizer.step()

    print(f'Epoch {epoch+1}, Loss: {loss.item()}')

# Save Model & Scaler

torch.save(model.state\_dict(), "transformer\_workload.pth")

joblib.dump(scaler, "scaler.pkl")

This:  
Uses a **Transformer-based model**.  
Normalizes data & converts it into **sequential format**.  
Uses **24 past readings** for forecasting.  
Saves the trained **Transformer model & scaler**.

**Step 3: Deploy Transformer API**

Now, we expose the trained Transformer model via a **Flask API** for real-time Oracle predictions.

transformer\_api.py

from flask import Flask, request, jsonify

import torch

import numpy as np

import joblib

from transformers import TimeSeriesTransformerModel, TimeSeriesTransformerConfig

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

# Load Model & Scaler

config = TimeSeriesTransformerConfig(d\_model=64, n\_heads=4, num\_encoder\_layers=3, num\_decoder\_layers=3, dropout=0.1)

model = TimeSeriesTransformerModel(config)

model.load\_state\_dict(torch.load("transformer\_workload.pth"))

model.eval()

scaler = joblib.load("scaler.pkl")

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

def predict():

    data = request.json['sequence']

    sequence = np.array(data).reshape(1, len(data), 3)  # 3 features: metric\_value, day\_of\_week, hour\_of\_day

    sequence = torch.tensor(sequence, dtype=torch.float32)

    # Predict

    with torch.no\_grad():

        prediction = model(sequence).squeeze().numpy()

    # Convert back to original scale

    predicted\_value = scaler.inverse\_transform([[prediction]])[0][0]

    return jsonify({'predicted\_value': predicted\_value})

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

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

Accepts **time-series workload data**.  
Uses **Transformers** for multi-step forecasting.  
Returns **denormalized workload prediction**.

**Step 4: Oracle PL/SQL Integration**

Now, we call the **Transformer API from Oracle**.

api\_call\_function.sql

CREATE OR REPLACE FUNCTION ml\_forecast\_transformer (

    p\_metric\_name VARCHAR2

) RETURN NUMBER IS

    v\_response CLOB;

    v\_sequence CLOB;

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

    v\_prediction NUMBER;

BEGIN

    -- Fetch last 24 workload values with day & hour

    SELECT JSON\_ARRAYAGG(JSON\_OBJECT('metric\_value' VALUE metric\_value, 'day\_of\_week' VALUE day\_of\_week, 'hour\_of\_day' VALUE hour\_of\_day) ORDER BY collection\_time)

    INTO v\_sequence

    FROM (SELECT metric\_value, day\_of\_week, hour\_of\_day

          FROM workload\_time\_series

          WHERE metric\_name = p\_metric\_name

          ORDER BY collection\_time DESC FETCH FIRST 24 ROWS ONLY);

    -- Call Transformer API

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

    -- Extract predicted value

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

    RETURN v\_prediction;

END ml\_forecast\_transformer;

/

Sends **last 24 readings (with time context)**.  
Gets **Transformer-based workload prediction**.  
Returns **forecasted value for Oracle analytics**.

**Step 5: Automating Transformer AI Predictions**

We schedule **Transformer-based workload predictions** every **30 minutes**.

transformer\_ai\_workload\_predictions\_scheduler.sql

BEGIN

    DBMS\_SCHEDULER.create\_job (

        job\_name        => 'TRANSFORMER\_AI\_WORKLOAD\_JOB',

        job\_type        => 'PLSQL\_BLOCK',

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

        start\_date      => SYSTIMESTAMP,

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

        enabled         => TRUE

    );

END;

/

**Step 6: Enhancing Chatbot with Transformer AI**

chatbot\_transformer\_forecast\_response\_procedure.sql

CREATE OR REPLACE PROCEDURE chatbot\_transformer\_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

    v\_predicted\_cpu := ml\_forecast\_transformer('CPU Usage (%)');

    v\_predicted\_sessions := ml\_forecast\_transformer('Active Sessions');

    IF v\_predicted\_cpu > 85 THEN

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

        v\_advice := 'Optimize queries and consider autoscaling.';

    ELSIF v\_predicted\_sessions > 250 THEN

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

        v\_advice := 'Increase connection pool size.';

    ELSE

        v\_alert\_msg := 'No major workload spikes expected.';

        v\_advice := 'Monitor system normally.';

    END IF;

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

END chatbot\_transformer\_forecast\_response;

/