

```

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import itertools
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error, r2_score
import xgboost as xgb
import os

files = [
    "ETTh1.csv",
    "ETTh2.csv",
    "ETTm1.csv",
    "ETTm2.csv"
]

dfs_raw = [pd.read_csv(f) for f in files]

```

## Verify Time Consistency

First, check if all datasets follow the same time frequency (e.g., every second or every minute). If they're not perfectly aligned or have missing timestamps, resample them to a unified scale using:

```

for fname, df in zip(files, dfs_raw):
    df['date'] = pd.to_datetime(df['date'])
    diffs = df['date'].diff().dropna()
    print(f"\nFile: {fname}")
    print(diffs.value_counts().head())

```

```

File: ETTh1.csv
date
0 days 01:00:00    17419
Name: count, dtype: int64

```

```

File: ETTh2.csv
date
0 days 01:00:00    17419
Name: count, dtype: int64

```

```

File: ETTm1.csv
date
0 days 00:15:00    69679
Name: count, dtype: int64

```

```
File: ETTm2.csv
date
0 days 00:15:00    69679
Name: count, dtype: int64
```

---

```
df_main =
pd.read_csv("/kaggle/input/electricity-transformer-dataset/ETTm1.csv")
df_main.head()
```

	date	HUFL	HULL	MUFL	MULL	LUFL	LULL
OT							
0	2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340
30.531000							
1	2016-07-01 00:15:00	5.760	2.076	1.492	0.426	4.264	1.401
30.459999							
2	2016-07-01 00:30:00	5.760	1.942	1.492	0.391	4.234	1.310
30.038000							
3	2016-07-01 00:45:00	5.760	1.942	1.492	0.426	4.234	1.310
27.013000							
4	2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371
27.787001							

## handle outliers

### outlier detectors

```
def detect_outliers_iqr(series, factor=1.5):
    valid = series.dropna()
    if valid.empty:
        return pd.Series(False, index=series.index)
    q1, q3 = valid.quantile([0.25, 0.75])
    iqr = q3 - q1
    if pd.isna(iqr) or iqr == 0:
        return pd.Series(False, index=series.index)
    lower = q1 - factor * iqr
    upper = q3 + factor * iqr
    mask = ~series.between(lower, upper, inclusive='both')
    return mask.fillna(False)
```

```

def detect_outliers_zscore(series, threshold=3.0):
    valid = series.dropna()
    if valid.empty:
        return pd.Series(False, index=series.index)
    mean = valid.mean()
    std = valid.std(ddof=0)
    if pd.isna(std) or std == 0:
        return pd.Series(False, index=series.index)
    z = (series - mean) / std
    mask = z.abs() > threshold
    return mask.fillna(False)

#choose numeric columns
numeric_df = df_main.select_dtypes('number')

#per-column counts
outlier_rows = []
for col in numeric_df.columns:                                # iterate columns
    explicitly
    s = df_main[col]
    m_iqr = detect_outliers_iqr(s)
    m_z = detect_outliers_zscore(s)
    outlier_rows.append({
        'column': col,
        'iqr_count': int(m_iqr.sum()),
        'z_count': int(m_z.sum()),
        'iqr_pct': round(m_iqr.mean() * 100, 2),
        'z_pct': round(m_z.mean() * 100, 2),
    })

outlier_df = pd.DataFrame(outlier_rows).set_index('column')

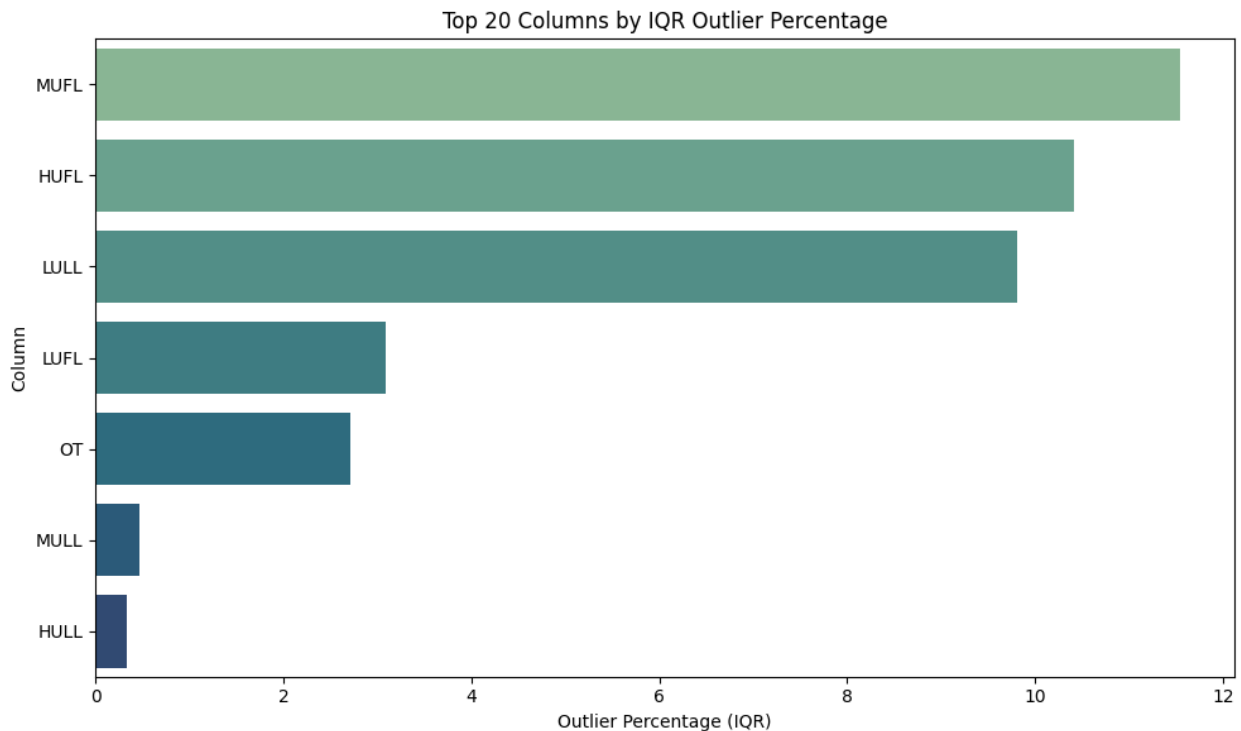
#top by IQR
top_iqr = outlier_df.sort_values('iqr_pct',
    ascending=False).head(20).reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(data=top_iqr, x='iqr_pct', y='column', orient='h',
    palette='crest')
plt.title('Top 20 Columns by IQR Outlier Percentage')
plt.xlabel('Outlier Percentage (IQR)')
plt.ylabel('Column')
plt.tight_layout()
plt.show()

```

/tmp/ipykernel\_55/3154530168.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=top_iqr, x='iqr_pct', y='column', orient='h',
palette='crest')
```



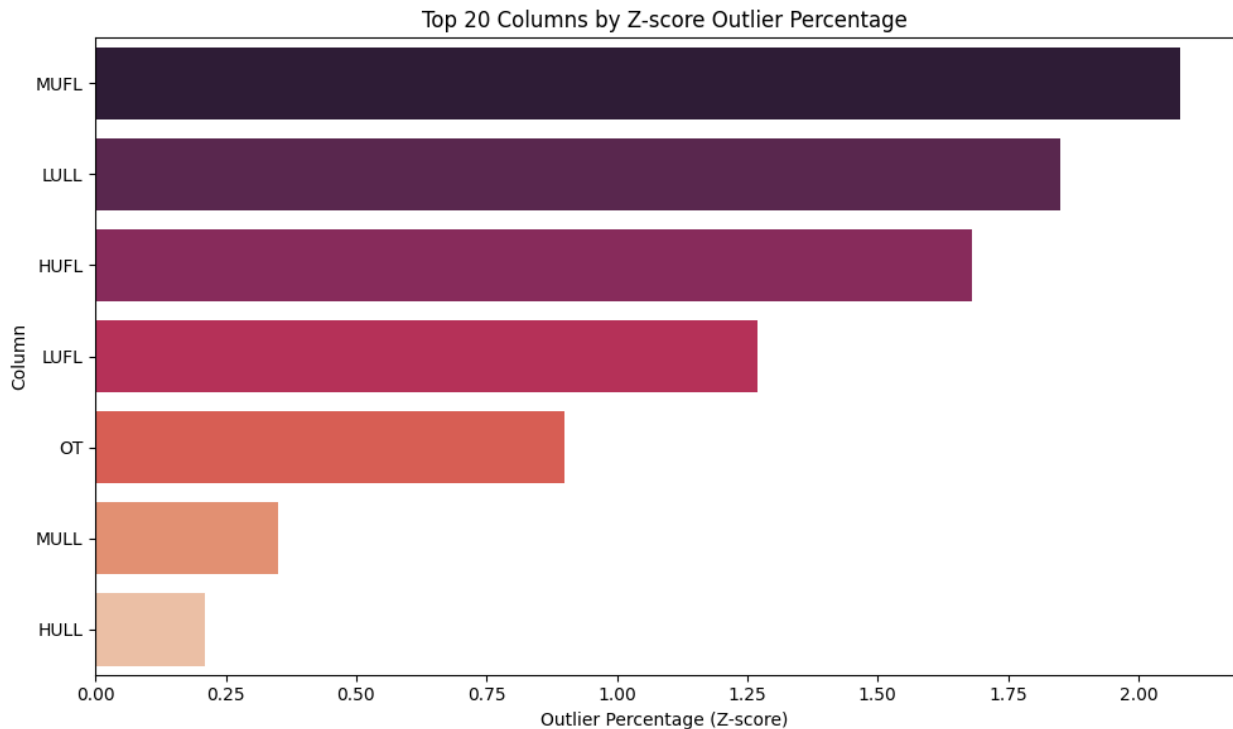
*#top by Z-score*

```
top_z = outlier_df.sort_values('z_pct',
ascending=False).head(20).reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(data=top_z, x='z_pct', y='column', orient='h',
palette='rocket')
plt.title('Top 20 Columns by Z-score Outlier Percentage')
plt.xlabel('Outlier Percentage (Z-score)')
plt.ylabel('Column')
plt.tight_layout()
plt.show()
```

/tmp/ipykernel\_55/805936563.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=top_z, x='z_pct', y='column', orient='h',
palette='rocket')
```



## Mark and replace outliers with NaN

```
nan_counts = df_main.isna().sum()
print(nan_counts)

date      0
HUFL      0
HULL      0
MUFL      0
MULL      0
LUFL      0
LULL      0
OT         0
dtype: int64

for col in df_main.select_dtypes('number'):
    mask = detect_outliers_iqr(df_main[col])
    df_main.loc[mask, col] = np.nan
#main df

nan_counts = df_main.isna().sum()
print(nan_counts)

date      0
HUFL     7258
HULL      235
MUFL     8048
MULL      328
```

```

LUFL      2153
LULL      6839
OT         1894
dtype: int64

# fill remaining gaps using time-based interpolation + fallback
df_main['date'] = pd.to_datetime(df_main['date'])
df_main = df_main.sort_values('date').set_index('date')

df_main = (
    df_main.interpolate(method='time', limit_direction='both')
        .ffill()
        .bfill()
)

# 4) quick check
missing_after = df_main.isna().mean() * 100
print("Remaining missing percentage per column:\n",
      missing_after.sort_values(ascending=False))

Remaining missing percentage per column:
HUFL      0.0
HULL      0.0
MUFL      0.0
MULL      0.0
LUFL      0.0
LULL      0.0
OT         0.0
dtype: float64

```

## checking

```

numeric_df = df_main.select_dtypes('number')
outlier_rows = []
for col in numeric_df.columns:                                # iterate columns
    explicitly
    s = df_main[col]
    m_iqr = detect_outliers_iqr(s)
    m_z = detect_outliers_zscore(s)
    outlier_rows.append({
        'column': col,
        'iqr_count': int(m_iqr.sum()),
        'z_count': int(m_z.sum()),
        'iqr_pct': round(m_iqr.mean() * 100, 2),
        'z_pct': round(m_z.mean() * 100, 2),
    })

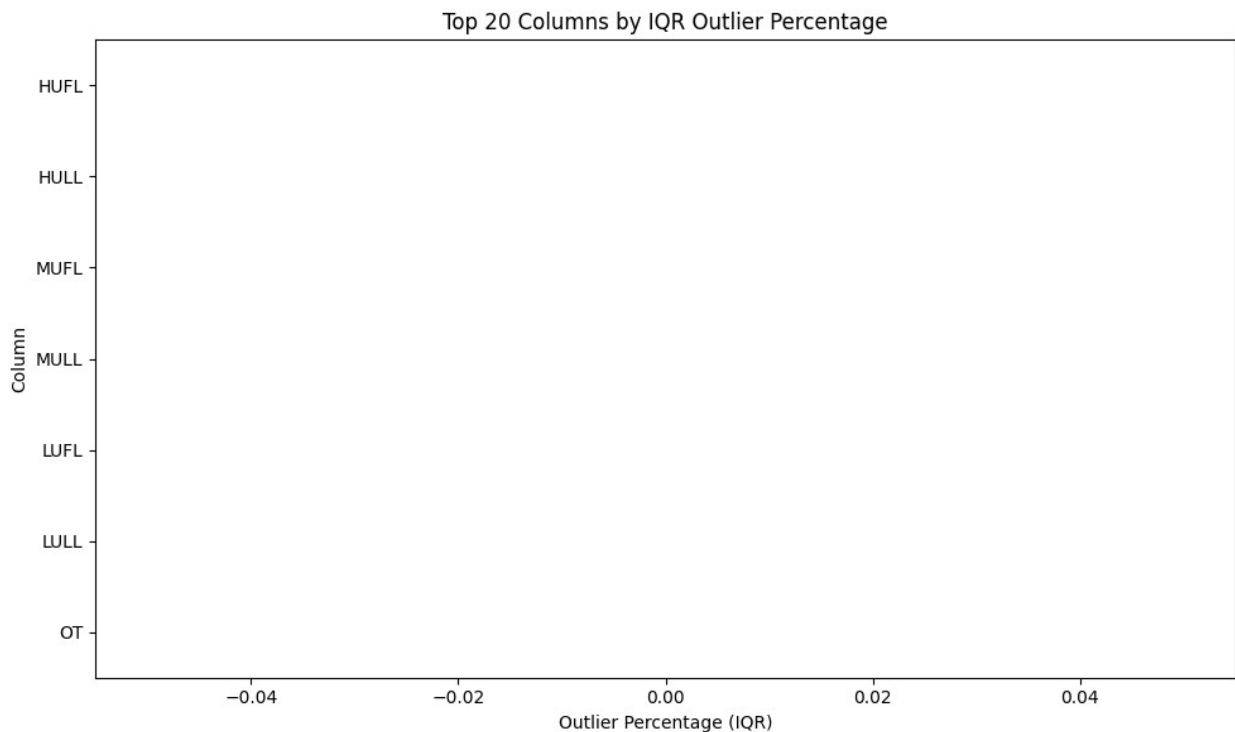
outlier_df = pd.DataFrame(outlier_rows).set_index('column')

```

```
# top by IQR %
top_iqr = outlier_df.sort_values('iqr_pct',
ascending=False).head(20).reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(data=top_iqr, x='iqr_pct', y='column', orient='h',
palette='crest')
plt.title('Top 20 Columns by IQR Outlier Percentage')
plt.xlabel('Outlier Percentage (IQR)')
plt.ylabel('Column')
plt.tight_layout()
plt.show()

/tmp/ipykernel_55/585245179.py:4: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

sns.barplot(data=top_iqr, x='iqr_pct', y='column', orient='h',
palette='crest')
```



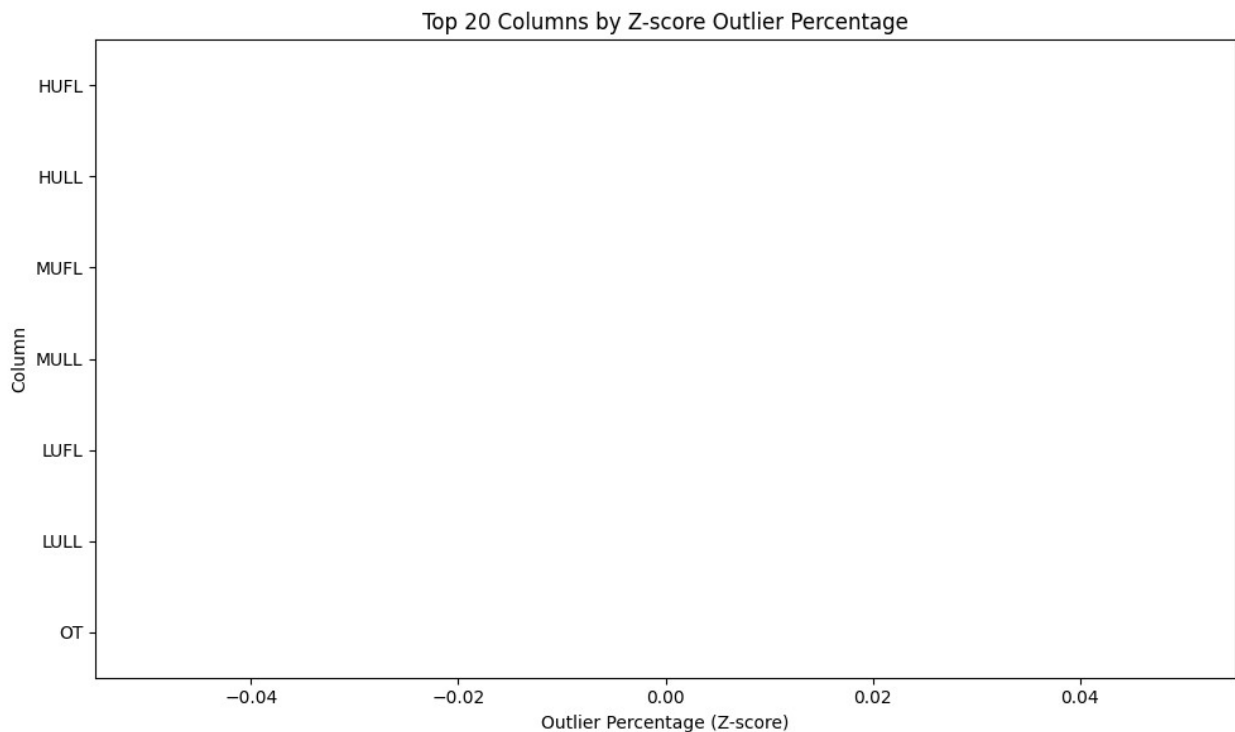
```
# --- top by Z-score %
top_z = outlier_df.sort_values('z_pct',
ascending=False).head(20).reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(data=top_z, x='z_pct', y='column', orient='h',
palette='rocket')
```

```
plt.title('Top 20 Columns by Z-score Outlier Percentage')
plt.xlabel('Outlier Percentage (Z-score)')
plt.ylabel('Column')
plt.tight_layout()
plt.show()
```

/tmp/ipykernel\_55/1015225668.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=top_z, x='z_pct', y='column', orient='h',
palette='rocket')
```



## computes the correlation

### Pearson correlations

```
#compute Pearson correlations
corr = df_main.corr()['OT'].sort_values(ascending=False)

#display top correlations
print("Top correlations with active_power:\n")
print(corr)
```

Top correlations with active\_power:

OT	1.000000
MULL	0.231388
HULL	0.229805
LULL	0.119918
LUFL	0.068222
HUFL	0.048561
MUFL	0.044240

Name: OT, dtype: float64

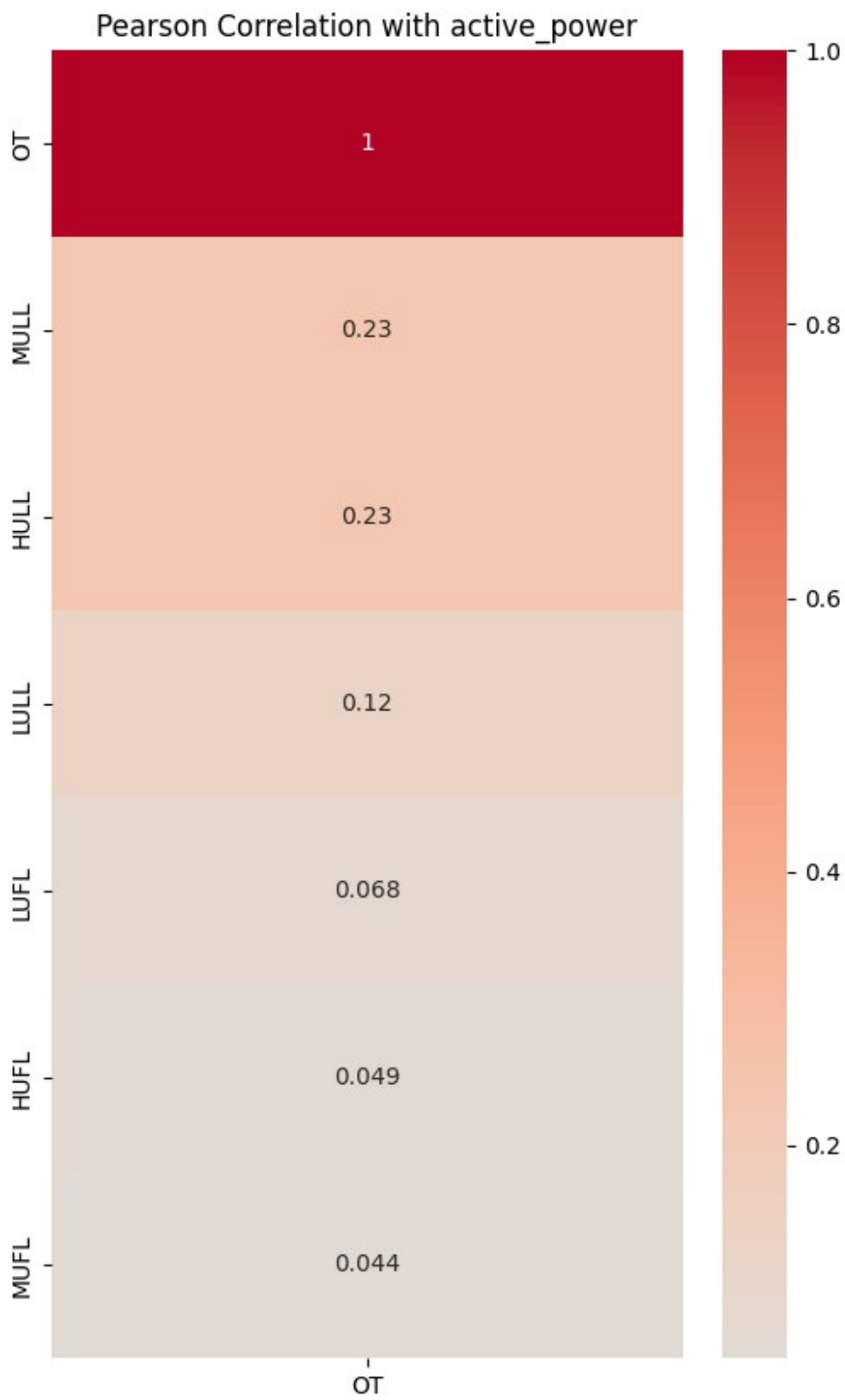
*# visualize as heatmap*

```
plt.figure(figsize=(6, 10))
```

```
sns.heatmap(corr.to_frame(), annot=True, cmap='coolwarm', center=0)
```

```
plt.title("Pearson Correlation with active_power")
```

```
plt.show()
```



```

corr = df_main.corr()['OT'].drop('OT')

# sort by absolute correlation (|r|)
corr_sorted = corr.abs().sort_values()

# show the weakest correlations (closest to 0)
weakest = corr_sorted.head(20).index.tolist()
print("Columns with weakest correlation to OT:\n", weakest)

Columns with weakest correlation to OT:
['MUFL', 'HUFL', 'LUFL', 'LULL', 'HULL', 'MULL']

print("\nActual correlation values for these columns:\n")
print(corr.loc[weakest])

```

Actual correlation values for these columns:

MUFL	0.044240
HUFL	0.048561
LUFL	0.068222
LULL	0.119918
HULL	0.229805
MULL	0.231388

Name: OT, dtype: float64

## information gain

```

from sklearn.feature_selection import mutual_info_regression
import pandas as pd

# prepare data
X = df_main.drop(columns=['OT'])
y = df_main['OT']

# compute mutual information (Information Gain)
mi = mutual_info_regression(X, y, random_state=42)

mi_df = pd.DataFrame({'Feature': X.columns, 'Information_Gain': mi})
mi_df = mi_df.sort_values('Information_Gain', ascending=False)

# print top features
print(mi_df)

```

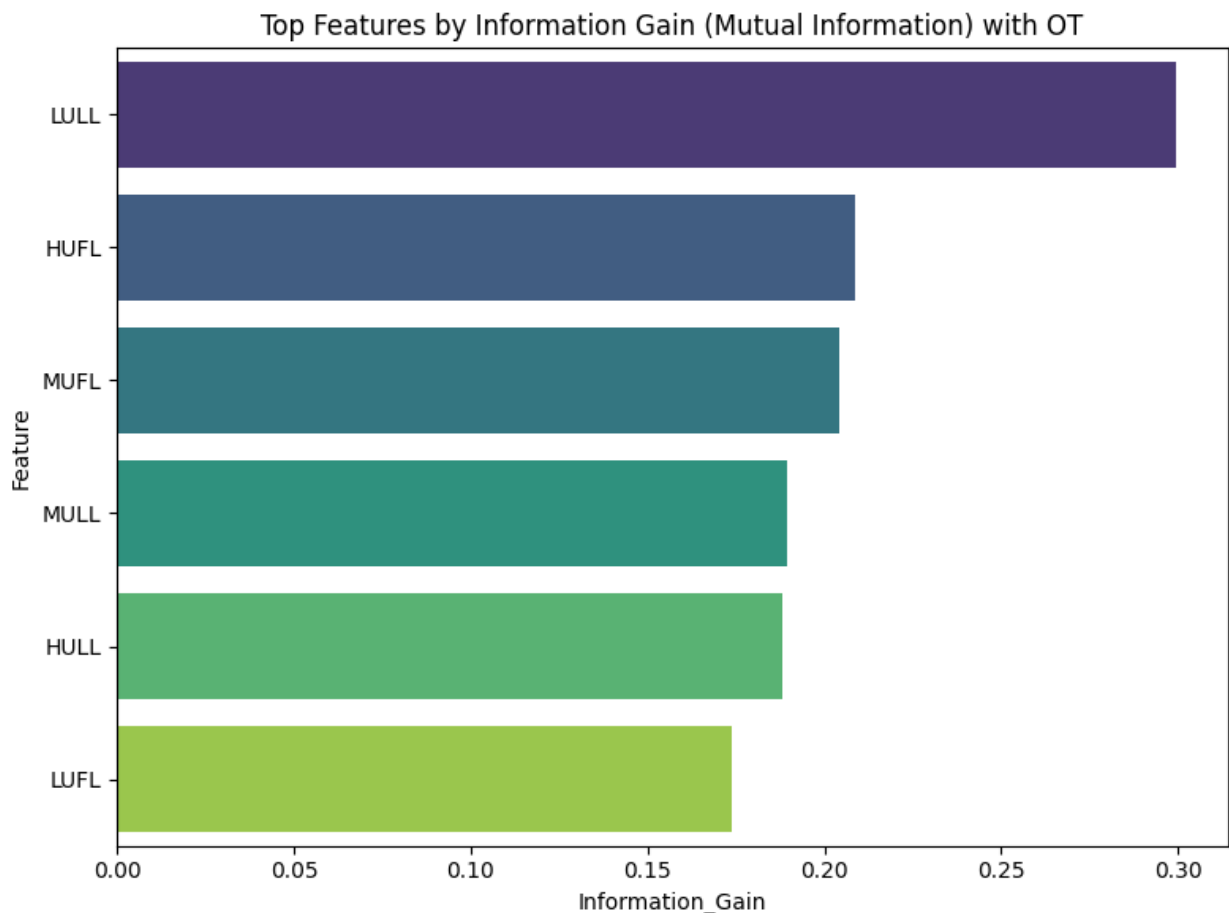
	Feature	Information_Gain
5	LULL	0.299215
0	HUFL	0.208853
2	MUFL	0.204417
3	MULL	0.189269
1	HULL	0.188047
4	LUFL	0.173861

```
# visualize
plt.figure(figsize=(8, 6))
sns.barplot(data=mi_df.head(15), x='Information_Gain', y='Feature',
palette='viridis')
plt.title("Top Features by Information Gain (Mutual Information) with
OT")
plt.tight_layout()
plt.show()
```

/tmp/ipykernel\_55/809312862.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=mi_df.head(15), x='Information_Gain', y='Feature',
palette='viridis')
```



```
# strong correlation features
print(mi_df.sort_values('Information_Gain', ascending=False))
```

	Feature	Information_Gain
5	LULL	0.299215
0	HUFL	0.208853
2	MUFL	0.204417
3	MULL	0.189269
1	HULL	0.188047
4	LUFL	0.173861

```
import math, time
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import StandardScaler
import joblib
from tqdm import tqdm
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import random
```

```
TARGET = "OT"
FEATS = [col for col in df_main.columns if col != TARGET]
WINDOW_SIZES = [30, 60, 120]
HORIZON = 1
VAL_FRAC = 0.10
TEST_FRAC = 0.20
BATCH_SIZE = 256
EPOCHS = 200
PATIENCE = 20
SEED = 42
```

#### *#Reproducibility*

```
def seed_everything(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
```

```
seed_everything(SEED)
```

```
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

#### *#Split chronologically*

```
df = df_main.sort_index().copy()
n = len(df)
test_cut = int(n * (1 - TEST_FRAC))
trainval, test = df.iloc[:test_cut], df.iloc[test_cut:]
```

```

val_cut = int(len(trainval) * (1 - VAL_FRAC))
train, val = trainval.iloc[:val_cut], trainval.iloc[val_cut:]

X_train, X_val, X_test = train[FEATS].values, val[FEATS].values,
test[FEATS].values
y_train, y_val, y_test = train[TARGET].values.reshape(-1,1),
val[TARGET].values.reshape(-1,1), test[TARGET].values.reshape(-1,1)

#Scale features
x_scaler = StandardScaler().fit(X_train)
y_scaler = StandardScaler().fit(y_train)
X_train, X_val, X_test = x_scaler.transform(X_train),
x_scaler.transform(X_val), x_scaler.transform(X_test)
y_train, y_val, y_test = y_scaler.transform(y_train),
y_scaler.transform(y_val), y_scaler.transform(y_test)

```

## Data loader

```

class SeqDataset(Dataset):
    def __init__(self, X, y):
        # X: (N, W, F) float32, y: (N,1) float32
        self.X = torch.from_numpy(X).float()
        self.y = torch.from_numpy(y).float()
    def __len__(self): return self.X.shape[0]
    def __getitem__(self, i): return self.X[i], self.y[i]

```

## Vanilla LSTM

```

# Vanilla LSTM
class VanillaLSTMReg(nn.Module):
    def __init__(self, n_features, hidden=64, dropout=0.1):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size=n_features,
            hidden_size=hidden,
            num_layers=1,
            batch_first=True
        )
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden, 1)

    def forward(self, x):
        # x: (B, W, F)
        out, _ = self.lstm(x)
        # (B, W, H)
        last = out[:, -1, :]
        # (B, H)
        last = self.dropout(last)
        return self.fc(last).squeeze(-1)

```

```

def make_sequences(X, y, window, horizon=1):
    xs, ys = [], []
    for i in range(len(X) - window - horizon + 1):
        xs.append(X[i:i+window])
        ys.append(y[i+window+horizon-1, 0])
    return np.asarray(xs, np.float32), np.asarray(ys, np.float32)

def make_sequences_delta(df, feats, target="OT", window=120,
horizon=1):
    X = df[feats].to_numpy(np.float32)
    ot = df[target].to_numpy(np.float32)

    Xs, y_delta, ot_last = [], [], []

    for i in range(window, len(df) - horizon):
        Xs.append(X[i-window:i])                # (W,F)
        ot_last.append(ot[i-1])                  # OT(t)
        y_delta.append(ot[i+horizon-1] - ot[i-1])# delta

    return (
        np.array(Xs),
        np.array(y_delta),
        np.array(ot_last)
    )

def make_sequences_delta_xy(X, y, window=120, horizon=1):
    X = np.asarray(X, dtype=np.float32)
    y = np.asarray(y, dtype=np.float32).ravel()

    Xs, y_delta, y_last = [], [], []
    for i in range(window, len(y) - horizon):
        Xs.append(X[i-window:i])
        y_last.append(y[i-1])
        y_delta.append(y[i+horizon-1] - y[i-1])

    return np.array(Xs), np.array(y_delta), np.array(y_last)

def run_epoch(model, loader, criterion, optimizer=None):
    train = optimizer is not None
    model.train() if train else model.eval()
    losses, preds, trues = [], [], []
    with torch.set_grad_enabled(train):
        for xb, yb in loader:
            xb, yb = xb.to(DEVICE), yb.to(DEVICE)
            yhat = model(xb)
            loss = criterion(yhat, yb)
            if train:
                optimizer.zero_grad()
                loss.backward()
                nn.utils.clip_grad_norm_(model.parameters(),

```

```

max_norm=1.0)
        optimizer.step()
        losses.append(loss.item())
        preds.append(yhat.detach().cpu())
        trues.append(yb.detach().cpu())
    if not preds:
        return float("nan"), None, None
    preds = torch.cat(preds).numpy()
    trues = torch.cat(trues).numpy()
    return float(sum(losses) / len(losses)), trues, preds

```

## Training & Evaluation

```

def train_one_window( Xtr, ytr, Xva, yva, W,
                      batch=BATCH_SIZE, epochs=EPOCHS,
                      patience=PATIENCE):
    n_features = Xtr.shape[-1]

    model = VanillaLSTMReg(n_features)
    model = model.to(DEVICE)

    crit = nn.MSELoss()
    opt = torch.optim.AdamW(model.parameters(), lr=5e-4)
    sched = torch.optim.lr_scheduler.ReduceLROnPlateau(
        opt, mode="min", factor=0.5, patience=4, min_lr=1e-5
    )

    # ensure dtypes/shapes are good: X: (N,W,F) float32, y: (N,) float32
    dtr = DataLoader(SeqDataset(Xtr, ytr), batch_size=batch,
                        shuffle=True)
    dva = DataLoader(SeqDataset(Xva, yva), batch_size=batch,
                        shuffle=False)

    history = {"loss": [], "val_loss": [], "mae": [], "val_mae": []}
    best_val = float("inf"); best_state = None; wait = 0

    for ep in range(1, epochs + 1):
        tr_loss, yt_tr, yp_tr = run_epoch(model, dtr, crit,
                                           optimizer=opt)
        va_loss, yt_va, yp_va = run_epoch(model, dva, crit,
                                           optimizer=None)

        # scaled-domain metrics
        tr_mae = float(np.mean(np.abs(yt_tr - yp_tr))) if yt_tr is not
        None else float("nan")
        va_mae_sc = float(np.mean(np.abs(yt_va - yp_va)))

```

```

        history["loss"].append(tr_loss);
    history["val_loss"].append(va_loss)
        history["mae"].append(tr_mae);
    history["val_mae"].append(va_mae_sc)

    sched.step(va_loss)
    print(f"Epoch {ep:03d} | loss {tr_loss:.4f} val_loss
{va_loss:.4f} | mae(sc) {tr_mae:.4f} val_mae(sc) {va_mae_sc:.4f}")

    if va_loss < best_val - 1e-6:
        best_val = va_loss
        best_state = {k: v.detach().cpu().clone() for k, v in
model.state_dict().items()}
        wait = 0
    else:
        wait += 1
        if wait >= patience:
            print("Early stopping")
            break

    if best_state is not None:
        model.load_state_dict(best_state)

    return model, history

```

## Training loop

```

def inverse_metrics(y_true_sc, y_pred_sc, y_scaler):
    # inverse scaling
    yt = y_scaler.inverse_transform(y_true_sc.reshape(-1, 1)).ravel()
    yp = y_scaler.inverse_transform(y_pred_sc.reshape(-1, 1)).ravel()

    mse = mean_squared_error(yt, yp)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(yt, yp)
    r2 = r2_score(yt, yp)

    return mse, mae, rmse, r2, yt, yp

results_pt = {}

for W in WINDOW_SIZES:
    Xtr, ytr_d, ot_tr = make_sequences_delta_xy(X_train, y_train,
window=W, horizon=HORIZON)
    Xva, yva_d, ot_va = make_sequences_delta_xy(X_val, y_val,
window=W, horizon=HORIZON)
    Xte, yte_d, ot_te = make_sequences_delta_xy(X_test, y_test,

```

```

window=W, horizon=HORIZON)

    # train (NOTE: inside train_one_window, make train loader
    shuffle=True)
    model, hist = train_one_window(Xtr, ytr_d.ravel(), Xva,
    yva_d.ravel(), W)

    # test inference (delta domain)
    dte = DataLoader(SeqDataset(Xte, yte_d.ravel()),
    batch_size=BATCH_SIZE, shuffle=False)
    te_loss, delta_true, delta_pred = run_epoch(model, dte,
    nn.MSELoss(), optimizer=None)

    # convert deltas -> OT (original domain)
    y_true_ot = ot_te + delta_true.ravel()
    y_pred_ot = ot_te + delta_pred.ravel()

    # metrics on OT
    mse = mean_squared_error(y_true_ot, y_pred_ot)
    rmse = float(np.sqrt(mse))
    mae = mean_absolute_error(y_true_ot, y_pred_ot)
    r2 = r2_score(y_true_ot, y_pred_ot)

    results_pt[W] = {
        "history": hist,
        "metrics": (mse, mae, rmse, r2),
        "y_true": y_true_ot,
        "y_pred": y_pred_ot,
        "test_loss_delta_mse": te_loss, # this is MSE on delta
(scaled domain if you scaled)
    }

    print(f"[W={W}] TEST: MSE={mse:.3f} MAE={mae:.3f} RMSE={rmse:.3f}
R2={r2:.3f}")

    # plots
    plt.figure(figsize=(6,3))
    plt.plot(hist["loss"], label="train_loss")
    plt.plot(hist["val_loss"], label="val_loss")
    plt.title(f"PyTorch LSTM (window={W}) - Loss (delta MSE)")
    plt.xlabel("Epoch"); plt.ylabel("MSE"); plt.legend();
plt.tight_layout(); plt.show()

    steps = min(len(y_true_ot), 4*24*60)
    plt.figure(figsize=(10,3))
    plt.plot(y_true_ot[-steps:], label="Actual OT")
    plt.plot(y_pred_ot[-steps:], label="Predicted OT")
    plt.title(f"Test Window - OT (window={W})")
    plt.xlabel("Minute (last ~7 days)"); plt.ylabel("OT")

```

```

plt.legend(); plt.tight_layout(); plt.show()

# save model (minimal)
torch.save(
    {"state_dict": model.state_dict(), "window": W, "n_features":
Xtr.shape[-1]},
    f"VanillaLSTM{W}.pt"
)

```

```

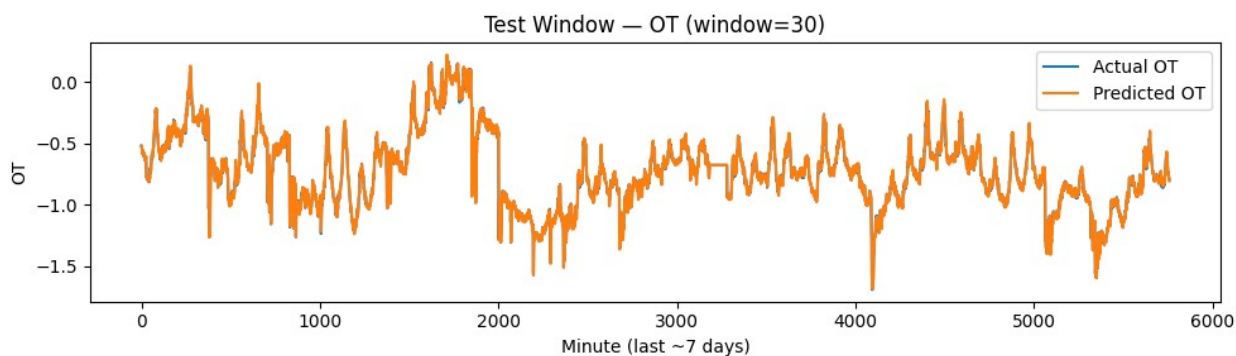
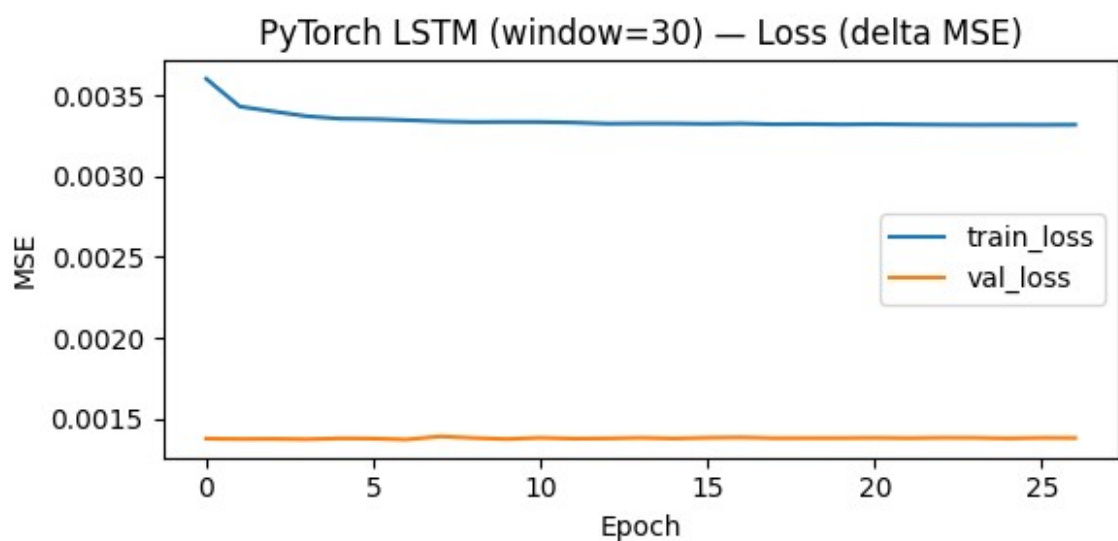
Epoch 001 | loss 0.0036 val_loss 0.0014 | mae(sc) 0.0408 val_mae(sc)
0.0267
Epoch 002 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0392 val_mae(sc)
0.0267
Epoch 003 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0387 val_mae(sc)
0.0267
Epoch 004 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0384 val_mae(sc)
0.0266
Epoch 005 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0383 val_mae(sc)
0.0266
Epoch 006 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0382 val_mae(sc)
0.0266
Epoch 007 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0382 val_mae(sc)
0.0265
Epoch 008 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc)
0.0271
Epoch 009 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0267
Epoch 010 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0266
Epoch 011 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0268
Epoch 012 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 013 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 014 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 015 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 016 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0268
Epoch 017 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0268
Epoch 018 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0266
Epoch 019 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0266
Epoch 020 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0267
Epoch 021 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)

```

```

0.0266
Epoch 022 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0266
Epoch 023 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0267
Epoch 024 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0266
Epoch 025 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0266
Epoch 026 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0267
Epoch 027 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0378 val_mae(sc)
0.0266
Early stopping
[W=30] TEST: MSE=0.002 MAE=0.027 RMSE=0.040 R2=0.991

```

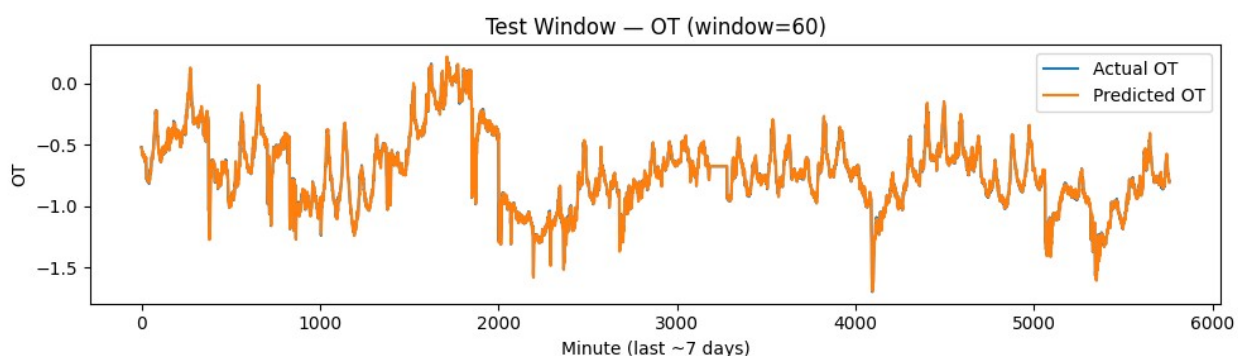
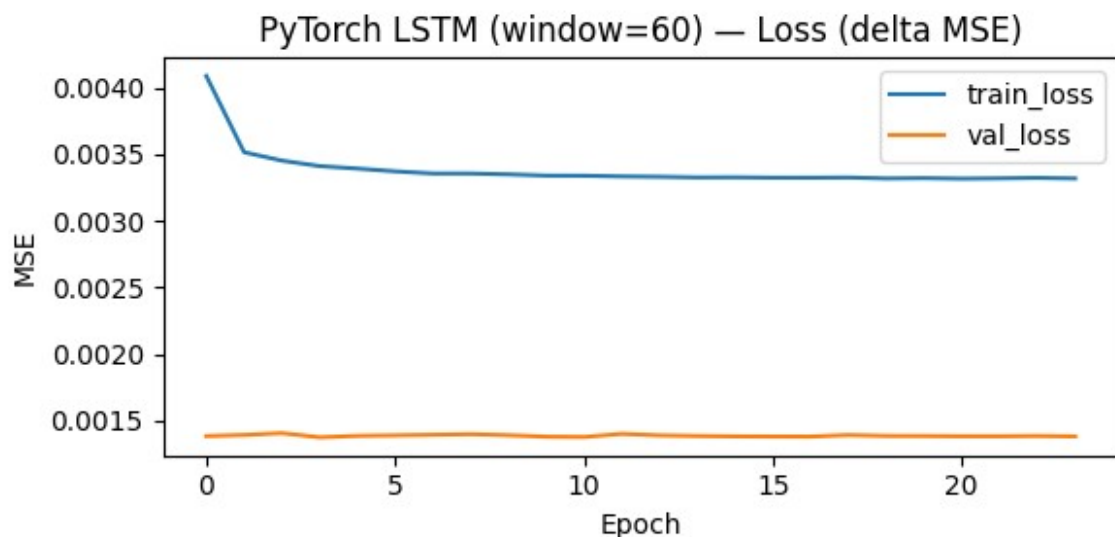


```

Epoch 001 | loss 0.0041 val_loss 0.0014 | mae(sc) 0.0444 val_mae(sc)
0.0268
Epoch 002 | loss 0.0035 val_loss 0.0014 | mae(sc) 0.0402 val_mae(sc)
0.0271

```

```
Epoch 003 | loss 0.0035 val_loss 0.0014 | mae(sc) 0.0395 val_mae(sc) 0.0273
Epoch 004 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0391 val_mae(sc) 0.0266
Epoch 005 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0388 val_mae(sc) 0.0268
Epoch 006 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0386 val_mae(sc) 0.0270
Epoch 007 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0384 val_mae(sc) 0.0270
Epoch 008 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0384 val_mae(sc) 0.0272
Epoch 009 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0383 val_mae(sc) 0.0269
Epoch 010 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0382 val_mae(sc) 0.0267
Epoch 011 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0382 val_mae(sc) 0.0266
Epoch 012 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0382 val_mae(sc) 0.0273
Epoch 013 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc) 0.0268
Epoch 014 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc) 0.0268
Epoch 015 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc) 0.0267
Epoch 016 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc) 0.0267
Epoch 017 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0267
Epoch 018 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0269
Epoch 019 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0267
Epoch 020 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0268
Epoch 021 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0267
Epoch 022 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0268
Epoch 023 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0268
Epoch 024 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc) 0.0267
Early stopping
[W=60] TEST: MSE=0.002 MAE=0.027 RMSE=0.040 R2=0.991
```



```
Epoch 001 | loss 0.0041 val_loss 0.0014 | mae(sc) 0.0444 val_mae(sc) 0.0271
Epoch 002 | loss 0.0035 val_loss 0.0014 | mae(sc) 0.0405 val_mae(sc) 0.0272
Epoch 003 | loss 0.0035 val_loss 0.0014 | mae(sc) 0.0396 val_mae(sc) 0.0270
Epoch 004 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0391 val_mae(sc) 0.0269
Epoch 005 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0388 val_mae(sc) 0.0271
Epoch 006 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0386 val_mae(sc) 0.0267
Epoch 007 | loss 0.0034 val_loss 0.0014 | mae(sc) 0.0385 val_mae(sc) 0.0270
Epoch 008 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0384 val_mae(sc) 0.0271
Epoch 009 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0383 val_mae(sc) 0.0270
Epoch 010 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0383 val_mae(sc) 0.0270
Epoch 011 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0382 val_mae(sc)
```

```
0.0275
Epoch 012 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc)
0.0268
Epoch 013 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0381 val_mae(sc)
0.0269
Epoch 014 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0271
Epoch 015 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0268
Epoch 016 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0270
Epoch 017 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0268
Epoch 018 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0268
Epoch 019 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 020 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0268
Epoch 021 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0380 val_mae(sc)
0.0267
Epoch 022 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0269
Epoch 023 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 024 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0268
Epoch 025 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0267
Epoch 026 | loss 0.0033 val_loss 0.0014 | mae(sc) 0.0379 val_mae(sc)
0.0268
Early stopping
[W=120] TEST: MSE=0.002 MAE=0.027 RMSE=0.040 R2=0.991
```

