```
import torch
import torch.nn as nn
import pandas as pd
import zipfile
import os
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
import joblib
from torch.utils.data import DataLoader, TensorDataset, random_split
import numpy as np
import seaborn as sns
from google.colab import drive
```

```
# Mount Google Drive
drive.mount('/content/drive')

# Set path to dataset files in Google Drive
data_path = '/content/drive/MyDrive/ashrae-energy-prediction/'

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Load data
building = pd.read_csv(data_path + "building_metadata.csv")
meter = pd.read_csv(data_path + "train.csv")
weather = pd.read_csv(data_path + "weather_train.csv")
```

```
# Merge datasets
data = meter.merge(building, on="building_id")
data = data.merge(weather, on=["site_id", "timestamp"])
data['timestamp'] = pd.to_datetime(data['timestamp'])
data['hour'] = data['timestamp'].dt.hour
data['day'] = data['timestamp'].dt.dayofyear
```

```
# --- Define Model ---
class PINN(nn.Module):
    def __init__(self):
        super(PINN, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(4, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        return self.layers(x)

# --- Physics-Informed Loss Functions ---
def physics_loss(meter_type, pred, delta_T):
    c_p = 4.18  # kJ/kg°C
    m = 0.1     # kg/s
    L = 2260    # kJ/kg, for steam

    if meter_type in [1, 2]:  # Chilled Water or Hot Water
        q_phys = m * c_p * delta_T
    elif meter_type == 3:  # Steam
        q_phys = m * L * torch.ones_like(delta_T)
    else:  # Electricity (simple temp relationship)
        q_phys = 50 + 2 * delta_T

    return torch.mean((pred.squeeze() - q_phys)**2)
```

```
# Assume constant indoor temperature
T_inside = 22.0
for meter_type in range(4):
    print(f"\nTraining model for meter type {meter_type}...")
    subset = data[data['meter'] == meter_type].dropna()
```

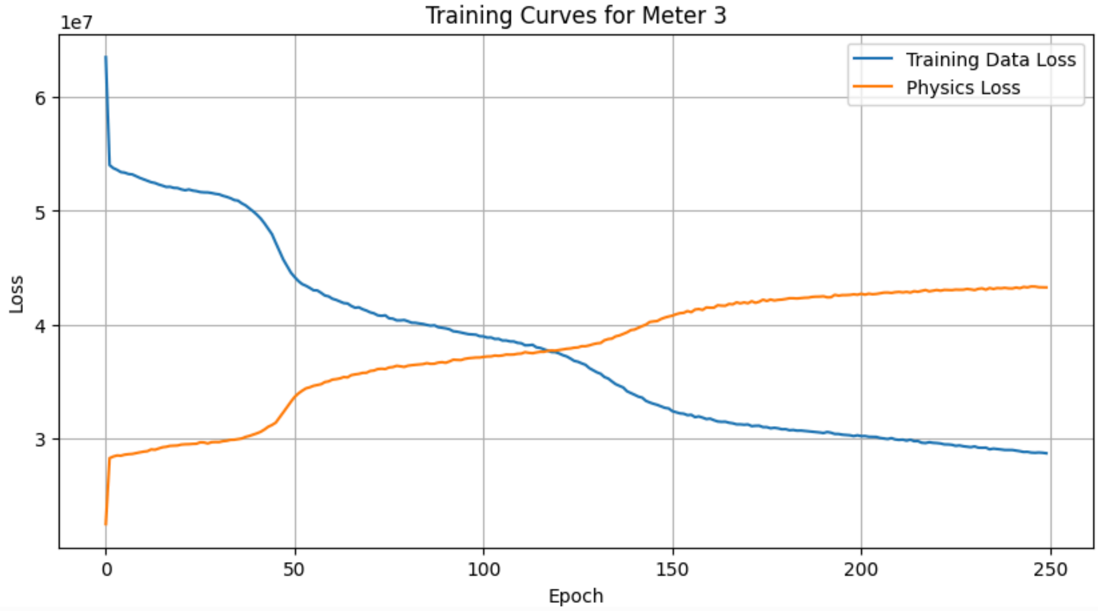
```
    if subset.empty:
        print("No data for this meter type.")
        continue

    subset['delta_T'] = T_inside - subset['air_temperature']
    if meter_type == 1:
        subset['delta_T'] = subset['air_temperature'] - T_inside

    features = subset[['square_feet', 'hour', 'day', 'delta_T']]
    target = subset['meter_reading']
    features = (features - features.mean()) / features.std()

    X = torch.tensor(features.values, dtype=torch.float32)
    y = torch.tensor(target.values.reshape(-1, 1), dtype=torch.float32)

    dataset = TensorDataset(X, y)
    train_size = int(0.8 * len(dataset))
    val_size = len(dataset) - train_size
    train_data, val_data = random_split(dataset, [train_size, val_size])

    train_loader = DataLoader(train_data, batch_size=256, shuffle=True)
    val_loader = DataLoader(val_data, batch_size=256)

    model = PINN()
    optimizer = torch.optim.AdamW(model.parameters(), lr=0.0003, weight_decay=1e-5)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=75, gamma=0.7)
    criterion = nn.MSELoss()

    loss_data_list, loss_phys_list, val_loss_list = [], [], []

    for epoch in range(250):
        model.train()
        epoch_loss_data, epoch_loss_phys = 0.0, 0.0

        for xb, yb in train_loader:
            pred = model(xb)
            delta_T = xb[:, -1]
            loss_data = criterion(pred, yb)
            loss_phys = physics_loss(meter_type, pred, delta_T)
            loss = loss_data + 0.3 * loss_phys

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_loss_data += loss_data.item()
            epoch_loss_phys += loss_phys.item()

        scheduler.step()
        loss_data_list.append(epoch_loss_data)
        loss_phys_list.append(epoch_loss_phys)

        # Validation loss
        model.eval()
        with torch.no_grad():
            val_preds, val_targets = [], []
            for xb, yb in val_loader:
                pred = model(xb)
                val_preds.extend(pred.squeeze().numpy())
                val_targets.extend(yb.squeeze().numpy())
            val_rmse = np.sqrt(mean_squared_error(val_targets, val_preds))
            val_loss_list.append(val_rmse)

        if epoch % 10 == 0:
            print(f"Epoch {epoch}: Train Loss = {loss.item():.4f}, Val RMSE = {val_rmse:.2f}")

    print(f"Finished training for meter {meter_type}.")
```

```
Training model for meter type 0...
Epoch 0: Train Loss = 79642.9375, Val RMSE = 232.31
Epoch 10: Train Loss = 41223.6719, Val RMSE = 229.34
Epoch 20: Train Loss = 53636.1406, Val RMSE = 227.39
Epoch 30: Train Loss = 54721.7539, Val RMSE = 224.84
Epoch 40: Train Loss = 28044.7012, Val RMSE = 219.39
Epoch 50: Train Loss = 76166.8828, Val RMSE = 204.47
Epoch 60: Train Loss = 49416.2344, Val RMSE = 203.15
Epoch 70: Train Loss = 38925.4805, Val RMSE = 200.08
Epoch 80: Train Loss = 22581.7109, Val RMSE = 198.14
Epoch 90: Train Loss = 34214.0859, Val RMSE = 194.51
Epoch 100: Train Loss = 45086.3750, Val RMSE = 193.43
Epoch 110: Train Loss = 32113.4258, Val RMSE = 192.27
Epoch 120: Train Loss = 49225.5859, Val RMSE = 187.39
Epoch 130: Train Loss = 35372.5156, Val RMSE = 185.39
```

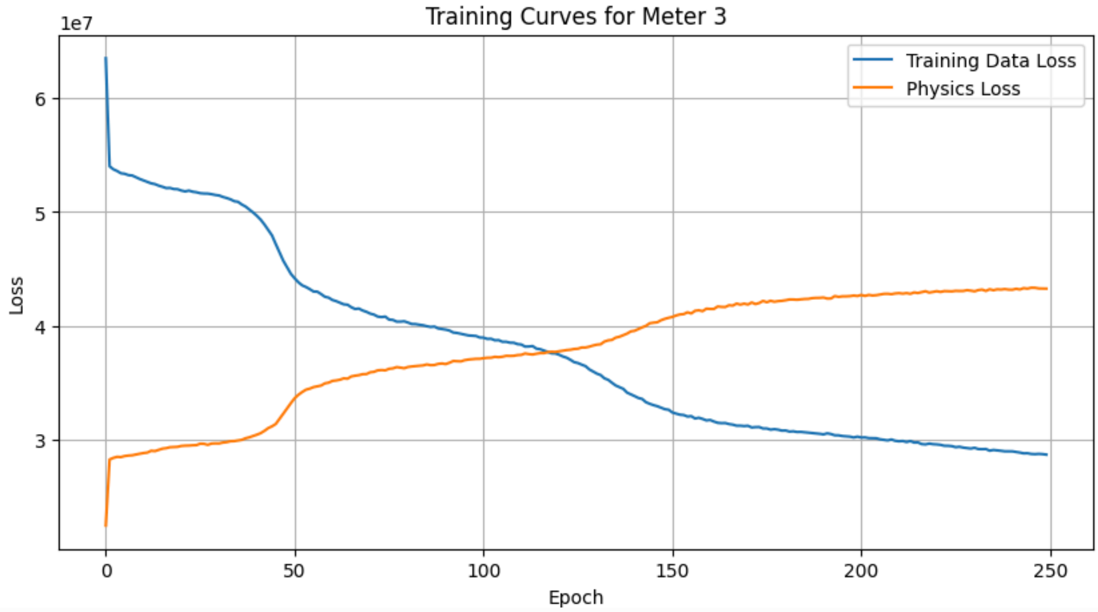
```
Epoch 140: Train Loss = 40301.0078, Val RMSE = 180.28
Epoch 150: Train Loss = 31318.9746, Val RMSE = 172.31
Epoch 160: Train Loss = 44355.6562, Val RMSE = 174.83
Epoch 170: Train Loss = 72496.8906, Val RMSE = 172.09
Epoch 180: Train Loss = 42203.9180, Val RMSE = 171.03
Epoch 190: Train Loss = 43947.4570, Val RMSE = 172.63
Epoch 200: Train Loss = 26167.9238, Val RMSE = 172.82
Epoch 210: Train Loss = 53689.0391, Val RMSE = 168.07
Epoch 220: Train Loss = 69582.8125, Val RMSE = 171.37
Epoch 230: Train Loss = 44526.4570, Val RMSE = 169.63
Epoch 240: Train Loss = 62259.2422, Val RMSE = 165.71
Finished training for meter 0.

Training model for meter type 1...
No data for this meter type.

Training model for meter type 2...
No data for this meter type.

Training model for meter type 3...
No data for this meter type.
```

```
subset = subset[subset['meter_reading'] > 0]  # Remove pure zeros
subset['meter_reading'] = np.log1p(subset['meter_reading'])  # log(1 + x)
```

```
    plt.figure(figsize=(10, 5))
    plt.plot(loss_data_list, label='Training Data Loss')
    plt.plot(loss_phys_list, label='Physics Loss')
    plt.title(f'Training Curves for Meter {meter_type}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.savefig(f"/content/meter_{meter_type}_loss_plot.png")
    plt.show()
```



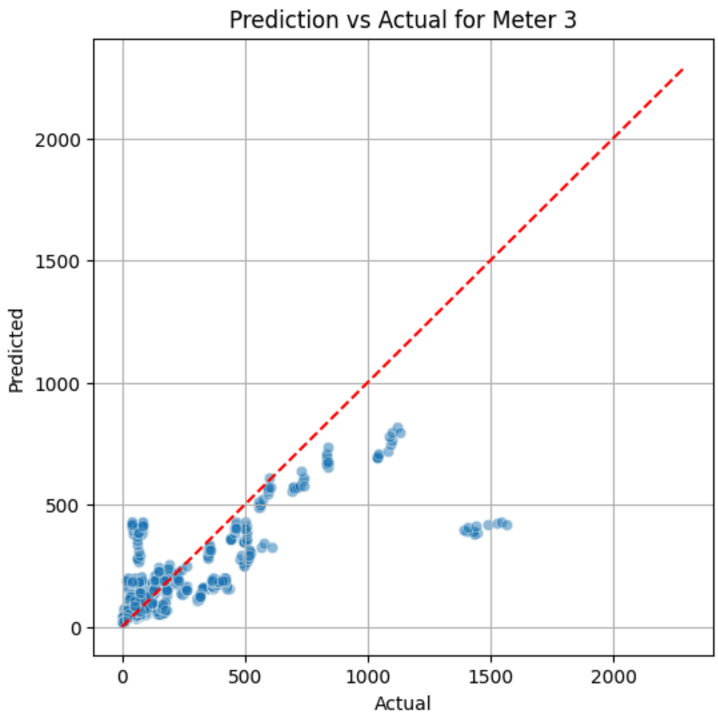

```
# --- Plot Training Curves ---
plt.figure(figsize=(10, 5))
plt.plot(loss_data_list, label='Training Data Loss')
plt.plot(loss_phys_list, label='Physics Loss')
# plt.plot(val_loss_list, label='Validation RMSE')  # Removed validation RMSE line
plt.title(f'Training Curves for Meter {meter_type}')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig(f"meter_{meter_type}_loss_plot.png")
plt.show()
```




```
model.eval()
with torch.no_grad():
    preds = model(X).numpy().squeeze()
    true = y.numpy().squeeze()

rmse = np.sqrt(mean_squared_error(true, preds))
r2 = r2_score(true, preds)
print(f"Evaluation for meter {meter_type} -> RMSE: {rmse:.2f}, R2: {r2:.2f}")
```

```
Evaluation for meter 3 -> RMSE: 167.54, R2: 0.66
```

```
plt.figure(figsize=(6, 6))
sns.scatterplot(x=true[:1000], y=preds[:1000], alpha=0.5)
plt.plot([true.min(), true.max()], [true.min(), true.max()], 'r--')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title(f"Prediction vs Actual for Meter {meter_type}")
plt.grid(True)
plt.savefig(f"/content/meter_{meter_type}_pred_vs_actual.png")
plt.show()

# --- Save Model and Features ---
torch.save(model.state_dict(), f"/content/pinn_model_meter_{meter_type}.pt")
joblib.dump(features.columns.tolist(), f"/content/features_meter_{meter_type}.pkl")
```



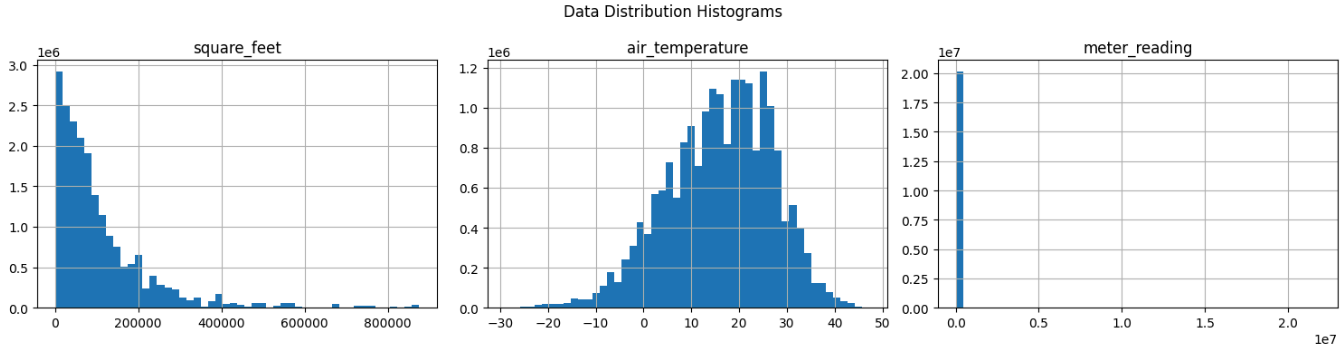

```
['/content/features_meter_3.pkl']
```

```
# Mean Absolute Percentage Error (MAPE)
mape = np.mean(np.abs((true - preds) / (true + 1e-5))) * 100  # +1e-5 to avoid division by zero

# Custom Accuracy: percentage of predictions within 20% of true values
tolerance = 0.2
within_tolerance = np.abs(true - preds) / (true + 1e-5) < tolerance
custom_accuracy = np.mean(within_tolerance) * 100

print(f"MAPE: {mape:.2f}%")
print(f"Custom Accuracy (within 20% of actual): {custom_accuracy:.2f}%")
```

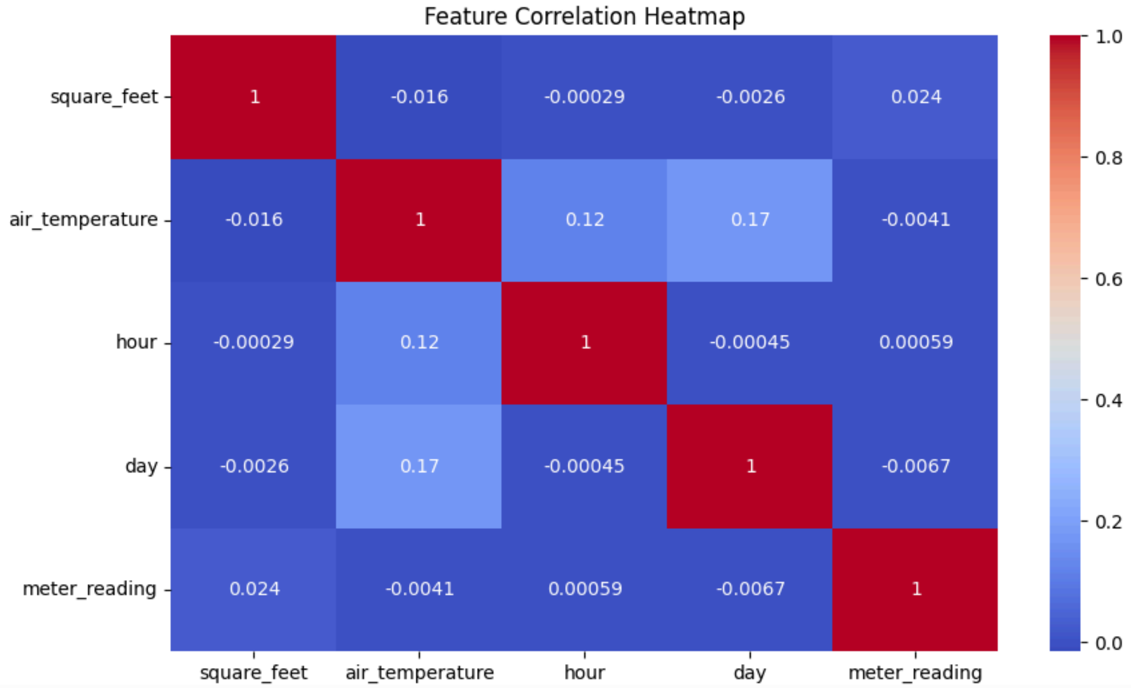
```
MAPE: 1525785.88%
Custom Accuracy (within 20% of actual): 17.09%
```

```
# Plot data distribution histograms
plt.figure(figsize=(12, 8))
data[['square_feet', 'air_temperature', 'meter_reading']].hist(bins=50, layout=(1, 3), figsize=(15, 4))
plt.suptitle('Data Distribution Histograms')
plt.tight_layout()
plt.savefig("/content/data_distribution_histograms.png")
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

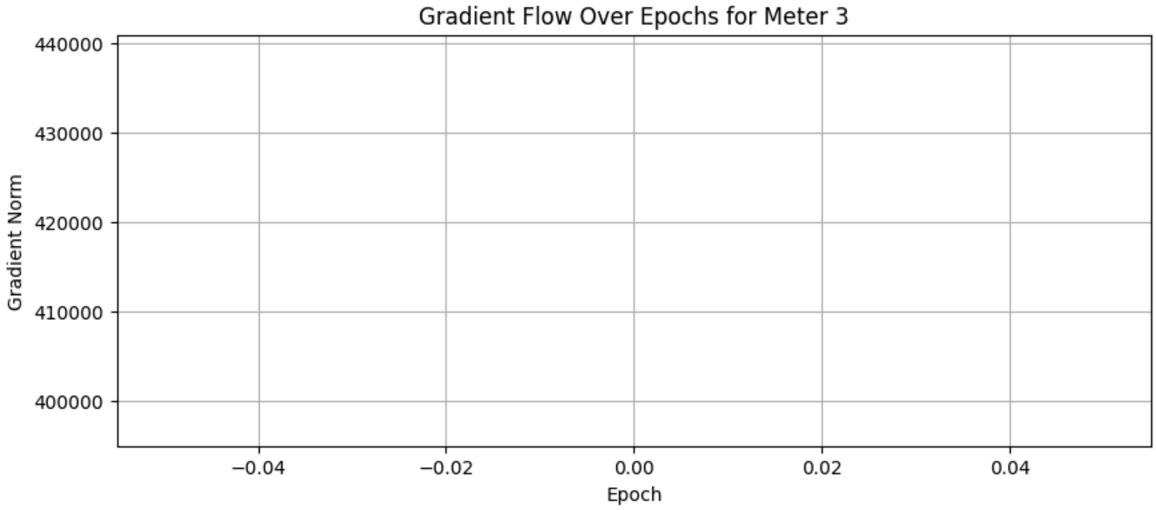



```
# Feature Correlation Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(data[['square_feet', 'air_temperature', 'hour', 'day', 'meter_reading']].corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.savefig("/content/feature_correlation_heatmap.png")
plt.show()
```

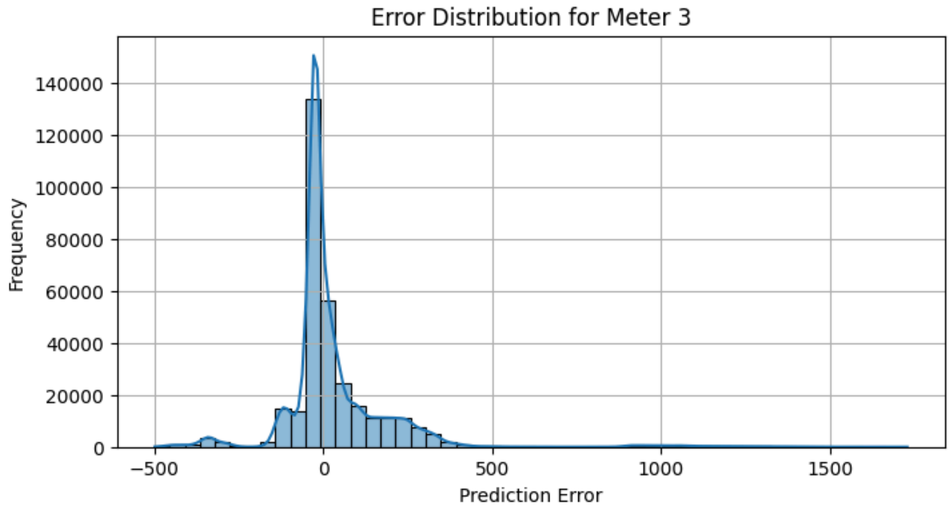



```
grad_flow_list=[]
total_norm = 0
for p in model.parameters():
            if p.grad is not None:
                param_norm = p.grad.data.norm(2)
                total_norm += param_norm.item() ** 2
grad_flow_list.append(total_norm ** 0.5)
```

```
    plt.figure(figsize=(10, 4))
    plt.plot(grad_flow_list, label='Gradient Flow Magnitude')
    plt.xlabel("Epoch")
    plt.ylabel("Gradient Norm")
    plt.title(f"Gradient Flow Over Epochs for Meter {meter_type}")
    plt.grid(True)
    plt.savefig(f"/content/meter_{meter_type}_gradient_flow.png")
    plt.show()

    # --- Final Evaluation ---
    model.eval()
    with torch.no_grad():
        preds = model(X).numpy().squeeze()
        true = y.numpy().squeeze()

    rmse = np.sqrt(mean_squared_error(true, preds))
    r2 = r2_score(true, preds)
    print(f"Evaluation for meter {meter_type} -> RMSE: {rmse:.2f}, R2: {r2:.2f}")

    # --- Error Analysis ---
    errors = true - preds
    plt.figure(figsize=(8, 4))
    sns.histplot(errors, bins=50, kde=True)
    plt.title(f"Error Distribution for Meter {meter_type}")
    plt.xlabel("Prediction Error")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.savefig(f"/content/meter_{meter_type}_error_analysis.png")
    plt.show()

    # --- Save Model and Features ---
    torch.save(model.state_dict(), f"/content/pinn_model_meter_{meter_type}.pt")
    joblib.dump(features.columns.tolist(), f"/content/features_meter_{meter_type}.pkl")
```




```
Evaluation for meter 3 -> RMSE: 167.54, R2: 0.66
```

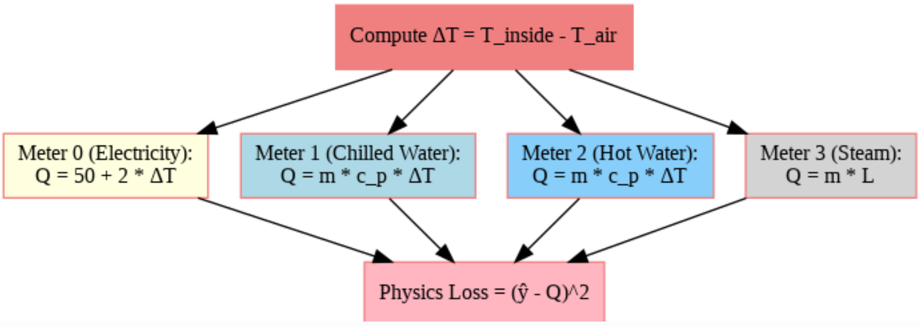



```
['/content/features_meter_3.pkl']
```

```
pip install graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (0.20.3)
```

```
from graphviz import Digraph

dot = Digraph(comment='Physics Loss Breakdown')
dot.attr(rankdir='TB', size='8,10')
dot.attr('node', shape='box', style='filled', color='lightcoral', fontsize='12')

# Base computation
dot.node('A', 'Compute ΔT = T_inside - T_air')

# Meter-specific branches
dot.node('B0', 'Meter 0 (Electricity):\nQ = 50 + 2 * ΔT', fillcolor='lightyellow')
dot.node('B1', 'Meter 1 (Chilled Water):\nQ = m * c_p * ΔT', fillcolor='lightblue')
dot.node('B2', 'Meter 2 (Hot Water):\nQ = m * c_p * ΔT', fillcolor='lightskyblue')
dot.node('B3', 'Meter 3 (Steam):\nQ = m * L', fillcolor='lightgray')

dot.node('C', 'Physics Loss = (ŷ - Q)^2', fillcolor='lightpink')

# Edges
dot.edge('A', 'B0')
dot.edge('A', 'B1')
dot.edge('A', 'B2')
dot.edge('A', 'B3')
dot.edge('B0', 'C')
dot.edge('B1', 'C')
dot.edge('B2', 'C')
dot.edge('B3', 'C')

# Render
dot.render('physics_loss_breakdown', format='png', cleanup=True)
from IPython.display import Image
Image('physics_loss_breakdown.png')
```




```
from graphviz import Digraph

dot = Digraph(comment='Evaluation Process')
dot.attr(rankdir='TB', size='8,10')
dot.attr('node', shape='box', style='filled', color='lightgray', fontsize='12')

# Nodes
dot.node('A', 'Trained PINN Model', fillcolor='lightblue')
dot.node('B', 'Input Test Data\n(features only)', fillcolor='lightyellow')
dot.node('C', 'Generate Predictions ŷ', fillcolor='lightgreen')
dot.node('D', 'Compare with True Values\n(if available)', fillcolor='lightcoral')
dot.node('E', 'Calculate Metrics:\nRMSE, R², MAPE', fillcolor='lightpink')
dot.node('F', 'Plot Results:\nPrediction vs Actual, Error Histograms', fillcolor='lightskyblue')
dot.node('G', 'Export Results:\npredictions.csv, plots', fillcolor='wheat')

# Edges
dot.edges(['AB', 'BC', 'CD', 'DE', 'EF', 'FG'])

# Render
dot.render('/content/evaluation_process_diagram', format='png', cleanup=True)
from IPython.display import Image
Image('/content/evaluation_process_diagram.png')
```