```
In [1]: import torch
import torch.nn as nn
import torch.utils.data as Data
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

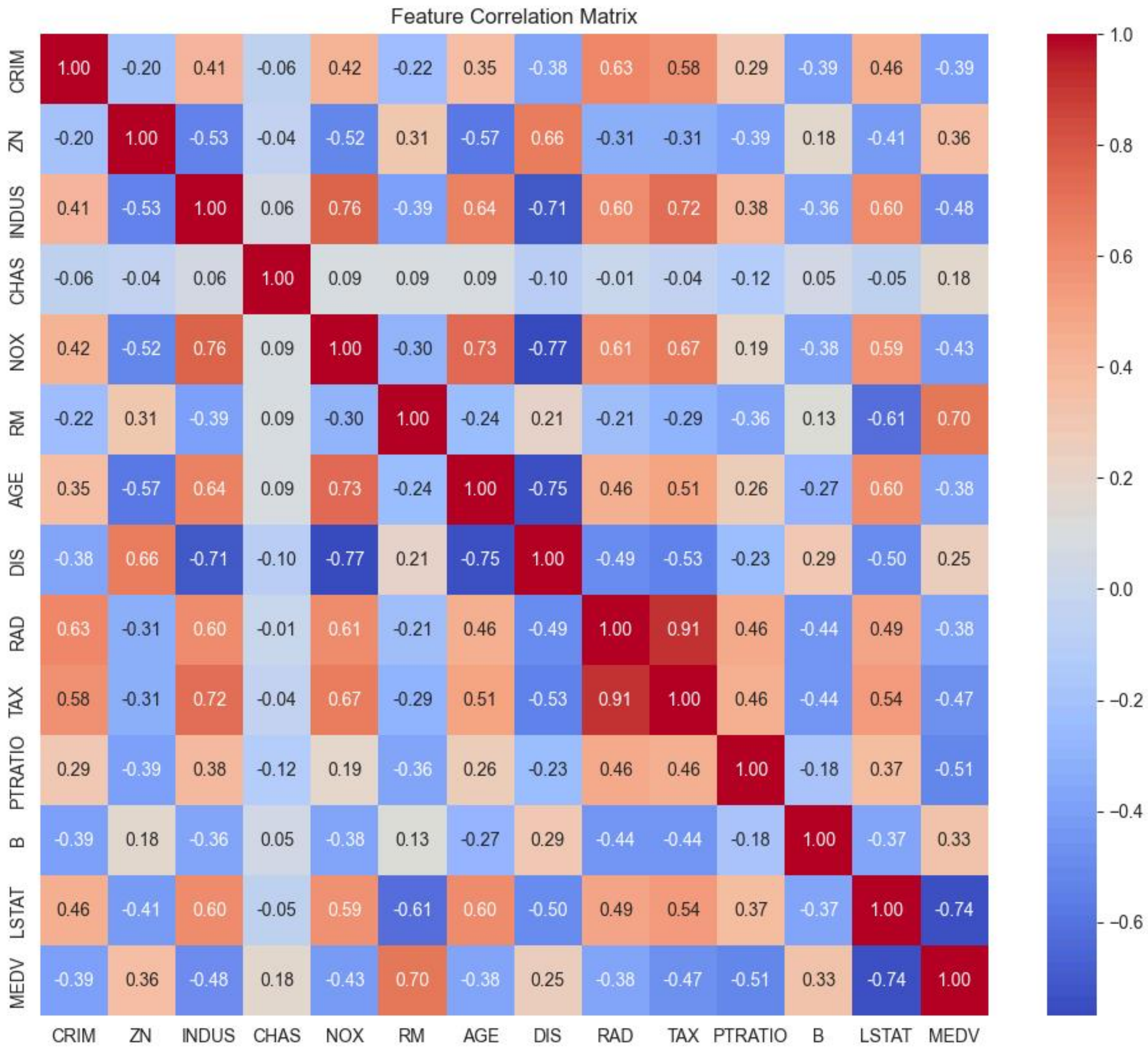
数据处理（包括相关性分析与PCA分析）

```
In [2]: def load_and_analyze_data(test_size=0.1, random_state=42, use_pca=False, n_compo
    data = pd.read_excel("BostonHousingData.xlsx")
    # 相关性分析
    plt.figure(figsize=(12, 10))
    corr_matrix = data.corr()
    sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')#热力图
    plt.title('Feature Correlation Matrix')
    plt.show()

    # 数据处理
    x_data = data.iloc[:, :13].values
    y_data = data.MEDV.values.reshape(-1, 1)
    feature_names = data.columns[:13]
    scaler = StandardScaler()
    x_data = scaler.fit_transform(x_data)

    # 3. PCA分析与可视化
    if use_pca:
        pca = PCA(n_components=n_components)
        x_pca = pca.fit_transform(x_data)

        print("\n=== 主成分特征权重 ===")
        components_df = pd.DataFrame(
            pca.components_,
            columns=feature_names,
            index=[f'PC{i+1}' for i in range(pca.n_components_)]
        )
        print(components_df.round(3))

        print("\n=== 各主成分主要特征 ===")
        for i in range(pca.n_components_):
            print(f"\nPC{i+1}（解释方差：{pca.explained_variance_ratio_[i]:.2%}）
            top_features = components_df.iloc[i].abs().nlargest(3)
            print(top_features.to_string(float_format="%.3f"))

        # 解释方差比例（成分占比）
        plt.figure(figsize=(10, 6))
        plt.bar(range(pca.n_components_), pca.explained_variance_ratio_)
        plt.xlabel('Principal Component')
        plt.ylabel('Variance Explained')
        plt.title('PCA Explained Variance Ratio')
        plt.show()

        # 累计解释方差
```

```
        plt.figure(figsize=(10, 6))
        plt.plot(np.cumsum(pca.explained_variance_ratio_))
        plt.xlabel('Number of Components')
        plt.ylabel('Cumulative Explained Variance')
        plt.title('PCA Cumulative Explained Variance')
        plt.grid(True)
        plt.show()

        # 主成分散点图
        if n_components >= 2:
            plt.figure(figsize=(10, 6))
            plt.scatter(x_pca[:, 0], x_pca[:, 1], c=data['MEDV'], cmap='viridis'
            plt.colorbar(label='MEDV')
            plt.xlabel('First Principal Component')
            plt.ylabel('Second Principal Component')
            plt.title('PCA Components Colored by MEDV')
            plt.show()

        # 使用PCA转换后的数据
        x_data = x_pca

    # 划分数据集
    x_train, x_test, y_train, y_test = train_test_split(
        x_data, y_data, test_size=test_size, random_state=random_state
    )

    # 转换为PyTorch张量
    xt_train = torch.FloatTensor(x_train)
    xt_test = torch.FloatTensor(x_test)
    yt_train = torch.FloatTensor(y_train)
    yt_test = torch.FloatTensor(y_test)

    return (xt_train, yt_train), (xt_test, yt_test), scaler, feature_names
```

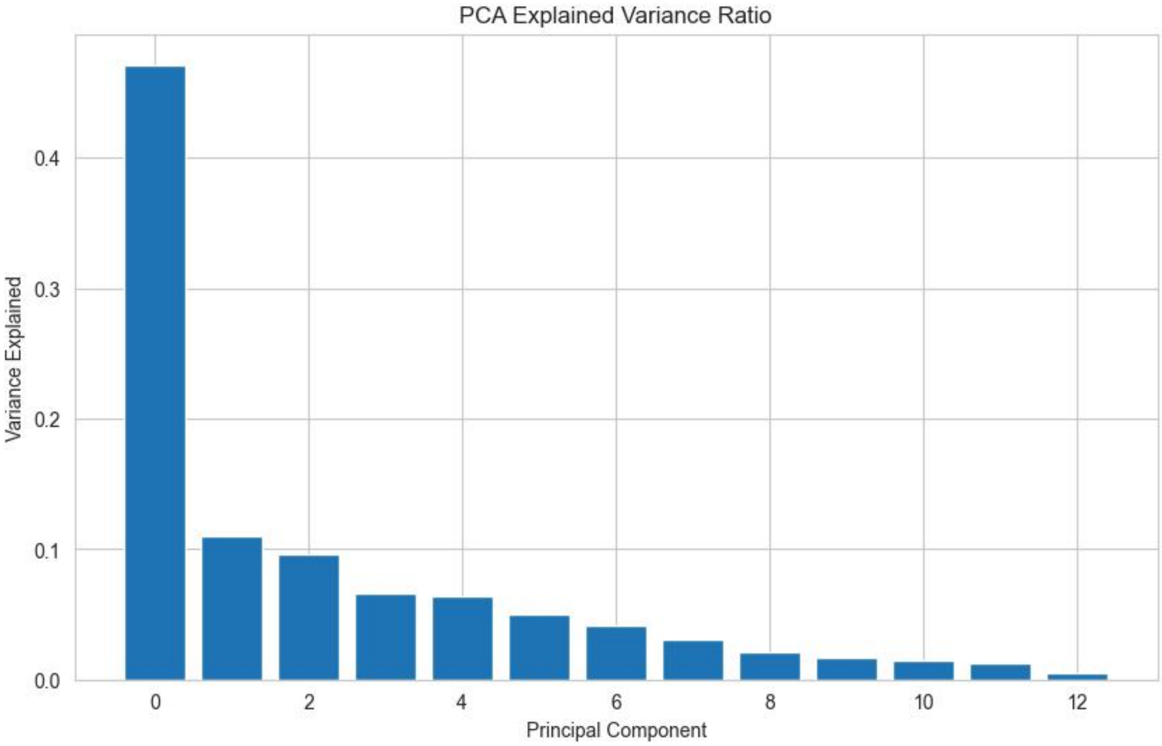
数据加载

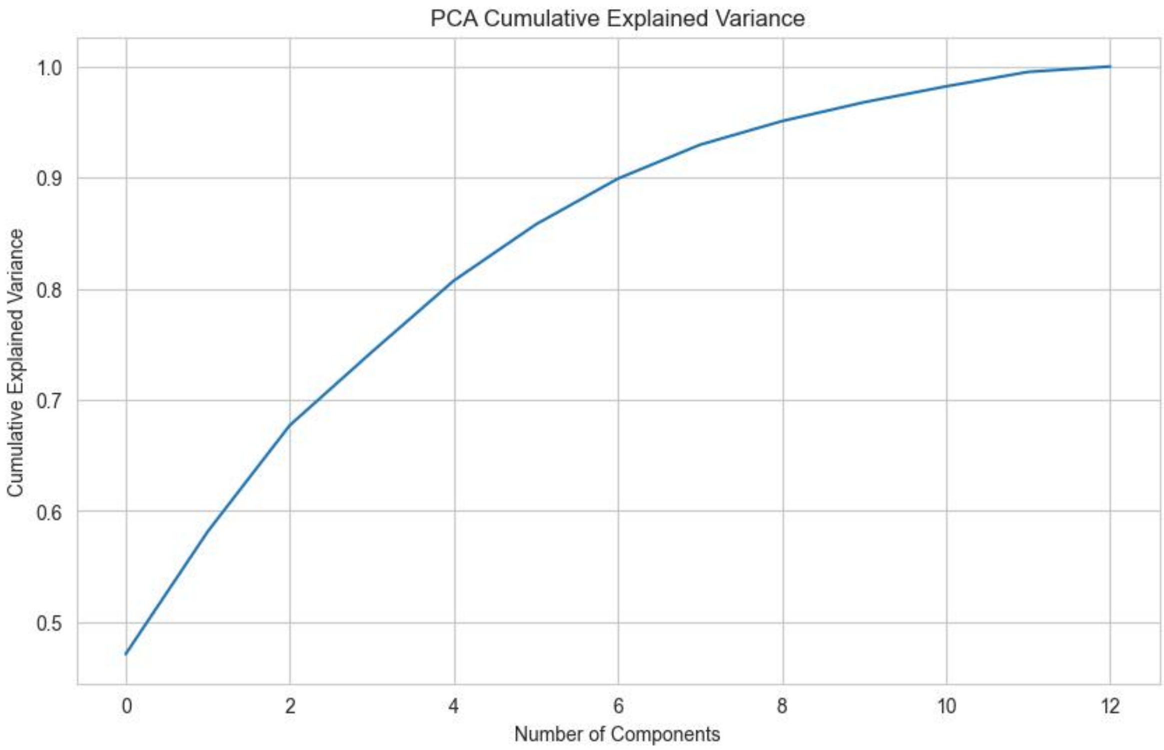
In [3]:
```
(xt_train, yt_train), (xt_test, yt_test), scaler, feature_names = load_and_analy

train_dataset = Data.TensorDataset(xt_train, yt_train)
train_loader = Data.DataLoader(dataset=train_dataset, batch_size=32, shuffle=Tru
```

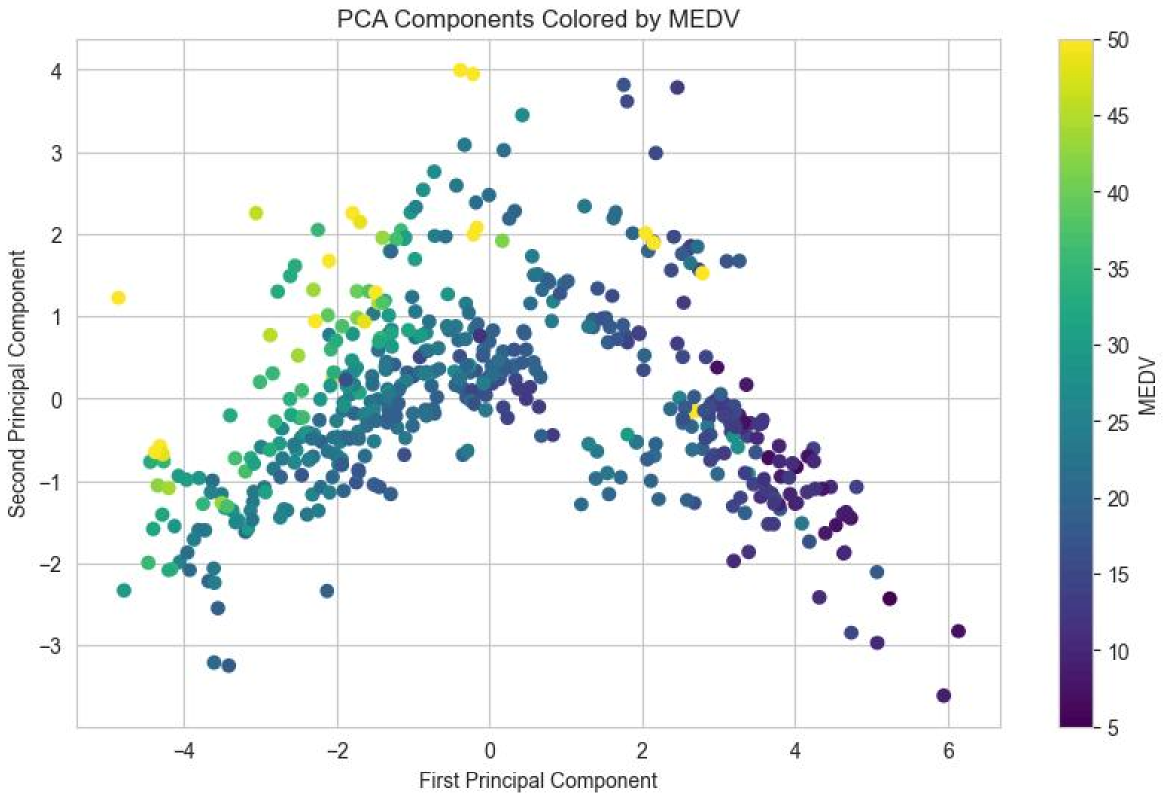

Feature Correlation Matrix
CRIM 1.00 -0.20 0.41 -0.06 0.42 -0.22 0.35 -0.38 0.63 0.58 0.29 -0.39 0.46 -0.39
ZN -0.20 1.00 -0.53 -0.04 -0.52 0.31 -0.57 0.66 -0.31 -0.31 -0.39 0.18 -0.41 0.36
INDUS 0.41 -0.53 1.00 0.06 0.76 -0.39 0.64 -0.71 0.60 0.72 0.38 -0.36 0.60 -0.48
CHAS -0.06 -0.04 0.06 1.00 0.09 0.09 0.09 -0.10 -0.01 -0.04 -0.12 0.05 -0.05 0.18
NOX 0.42 -0.52 0.76 0.09 1.00 -0.30 0.73 -0.77 0.61 0.67 0.19 -0.38 0.59 -0.43
RM -0.22 0.31 -0.39 0.09 -0.30 1.00 -0.24 0.21 -0.21 -0.29 -0.36 0.13 -0.61 0.70
AGE 0.35 -0.57 0.64 0.09 0.73 -0.24 1.00 -0.75 0.46 0.51 0.26 -0.27 0.60 -0.38
DIS -0.38 0.66 -0.71 -0.10 -0.77 0.21 -0.75 1.00 -0.49 -0.53 -0.23 0.29 -0.50 0.25
RAD 0.63 -0.31 0.60 -0.01 0.61 -0.21 0.46 -0.49 1.00 0.91 0.46 -0.44 0.49 -0.38
TAX 0.58 -0.31 0.72 -0.04 0.67 -0.29 0.51 -0.53 0.91 1.00 0.46 -0.44 0.54 -0.47
PTRATIO 0.29 -0.39 0.38 -0.12 0.19 -0.36 0.26 -0.23 0.46 0.46 1.00 -0.18 0.37 -0.51
B -0.39 0.18 -0.36 0.05 -0.38 0.13 -0.27 0.29 -0.44 -0.44 -0.18 1.00 -0.37 0.33
LSTAT 0.46 -0.41 0.60 -0.05 0.59 -0.61 0.60 -0.50 0.49 0.54 0.37 -0.37 1.00 -0.74
MEDV -0.39 0.36 -0.48 0.18 -0.43 0.70 -0.38 0.25 -0.38 -0.47 -0.51 0.33 -0.74 1.00
CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT MEDV
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6

```
=== 主成分特征权重 ===
       CRIM     ZN  INDUS   CHAS    NOX     RM    AGE    DIS    RAD    TAX  \
PC1   0.251 -0.256  0.347  0.005  0.343 -0.189  0.314 -0.322  0.320  0.338
PC2  -0.315 -0.323  0.112  0.455  0.219  0.149  0.312 -0.349 -0.272 -0.239
PC3   0.247  0.296 -0.016  0.290  0.121  0.594 -0.018 -0.050  0.287  0.221
PC4   0.062  0.129  0.017  0.816 -0.128 -0.281 -0.175  0.215  0.132  0.103
PC5  -0.082 -0.321  0.008 -0.087 -0.137  0.423 -0.017 -0.099  0.204  0.130
PC6   0.220  0.323  0.076 -0.167  0.153 -0.059  0.072 -0.023  0.143  0.193
PC7   0.778 -0.275 -0.340  0.074 -0.200  0.064  0.116 -0.104 -0.138 -0.315
PC8  -0.153  0.403 -0.174  0.025 -0.080  0.327  0.601  0.122 -0.080 -0.083
PC9   0.260  0.358  0.644 -0.014 -0.019  0.048 -0.068 -0.153 -0.471 -0.177
PC10 -0.019 -0.268  0.364  0.006 -0.231  0.431 -0.363  0.171 -0.022  0.035
PC11  0.110 -0.263  0.303 -0.014 -0.111 -0.053  0.459  0.696 -0.037  0.105
PC12  0.087 -0.071 -0.113 -0.004  0.804  0.153 -0.212  0.391 -0.107 -0.215
PC13  0.046 -0.081 -0.251  0.036  0.044  0.046 -0.039 -0.018 -0.633  0.720

      PTRATIO      B  LSTAT
PC1     0.205 -0.203  0.310
PC2    -0.306  0.239 -0.074
PC3    -0.323 -0.300 -0.267
PC4     0.283  0.168  0.069
PC5     0.584  0.346 -0.395
PC6    -0.273  0.803  0.053
PC7     0.002  0.070  0.087
PC8     0.318  0.005  0.424
PC9     0.254 -0.045 -0.195
PC10   -0.153  0.097  0.601
PC11   -0.175 -0.019 -0.271
PC12    0.210  0.042  0.055
PC13    0.023 -0.004  0.024

=== 各主成分主要特征 ===

PC1（解释方差：47.13%）
INDUS   0.347
NOX     0.343
TAX     0.338

PC2（解释方差：11.03%）
CHAS   0.455
DIS    0.349
ZN     0.323

PC3（解释方差：9.56%）
RM         0.594
PTRATIO    0.323
B          0.300

PC4（解释方差：6.60%）
CHAS       0.816
PTRATIO    0.283
RM         0.281

PC5（解释方差：6.42%）
PTRATIO    0.584
RM         0.423
LSTAT      0.395

PC6（解释方差：5.06%）
B          0.803
```

```
ZN         0.323
PTRATIO    0.273

PC7（解释方差：4.12%）
CRIM     0.778
INDUS    0.340
TAX      0.315

PC8（解释方差：3.05%）
AGE      0.601
LSTAT    0.424
ZN       0.403

PC9（解释方差：2.13%）
INDUS    0.644
RAD      0.471
ZN       0.358

PC10（解释方差：1.69%）
LSTAT    0.601
RM       0.431
INDUS    0.364

PC11（解释方差：1.43%）
DIS      0.696
AGE      0.459
INDUS    0.303

PC12（解释方差：1.30%）
NOX    0.804
DIS    0.391
TAX    0.215

PC13（解释方差：0.49%）
TAX      0.720
RAD      0.633
INDUS    0.251
```







模型定义（三层全连接网络）

In [4]:
```
class Model(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(0.03),

            nn.Linear(128, 64),
```

```
            nn.BatchNorm1d(64),
            nn.ReLU(),

            nn.Linear(64, 32),
            nn.ReLU(),

            nn.Linear(32, 1)
        )

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

#模型初始化
model = Model(input_dim=xt_train.shape[1])
print(model)
```

```
Model(
  (net): Sequential(
    (0): Linear(in_features=13, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (2): ReLU()
    (3): Dropout(p=0.03, inplace=False)
    (4): Linear(in_features=128, out_features=64, bias=True)
    (5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (6): ReLU()
    (7): Linear(in_features=64, out_features=32, bias=True)
    (8): ReLU()
    (9): Linear(in_features=32, out_features=1, bias=True)
  )
)
```

模型结构

训练函数

In [5]:
```
def train_model(model, train_loader, epochs):
    optimizer = torch.optim.Adam(model.parameters(), lr= 0.0005, weight_decay= 0
    criterion = nn.MSELoss()#适用于回归问题
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', pat

    model.train()
    loss_history = []

    for epoch in range(epochs):
        epoch_loss = 0
        for x, y in train_loader:
            optimizer.zero_grad()
            outputs = model(x)
            loss = criterion(outputs, y)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()

        avg_loss = epoch_loss / len(train_loader)
        loss_history.append(avg_loss)
        scheduler.step(avg_loss)

        if epoch % 15 == 0:
```

```
            print(f"Epoch {epoch}: Loss = {avg_loss:.4f}, LR = {optimizer.param_
    return model, loss_history
```

评估函数

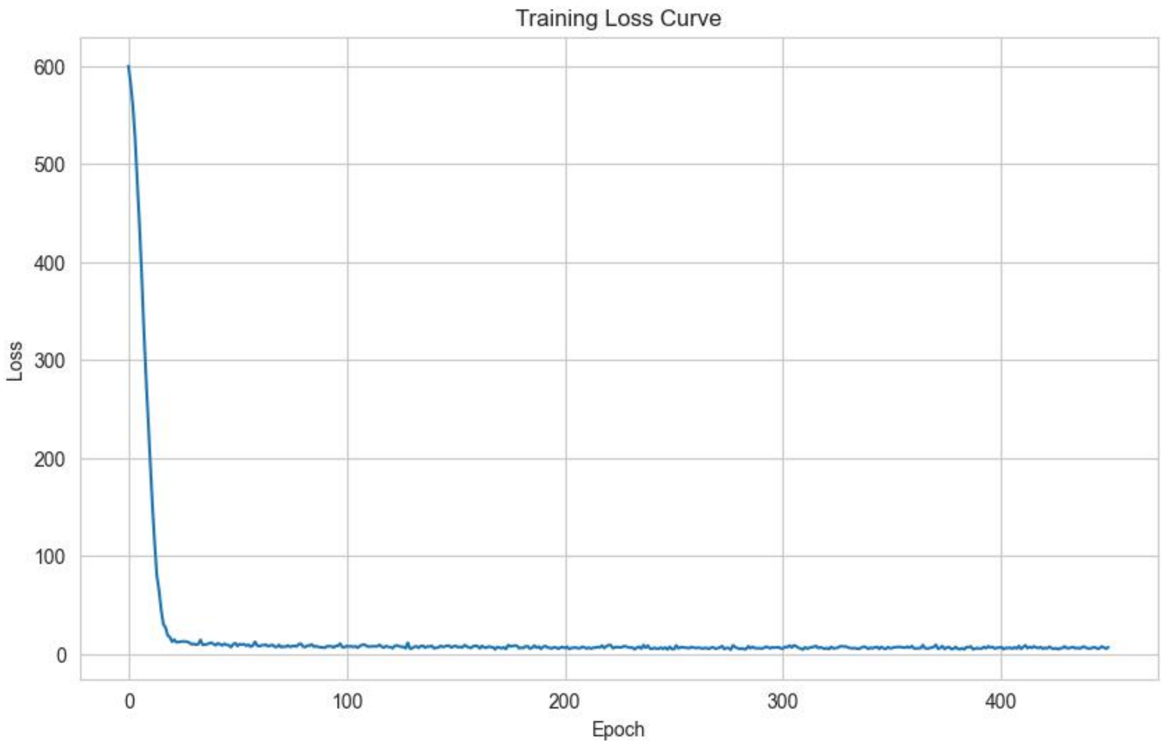
In [6]:
```
def evaluate_model(model, x_test, y_test):
    model.eval()
    with torch.no_grad():
        predictions = model(x_test).numpy()
        y_test = y_test.numpy()

        # 计算评估指标
        mse = mean_squared_error(y_test, predictions)#
        rmse = np.sqrt(mse)
        mae = mean_absolute_error(y_test, predictions)#
        r2 = r2_score(y_test, predictions)#

        # 打印评估结果
        print("\n" + "—"*50)
        print("Model Evaluation Metrics:")
        print(f"MSE: {mse:.4f}")
        print(f"RMSE: {rmse:.4f}")
        print(f"MAE: {mae:.4f}")
        print(f"R² Score: {r2:.4f}")
        print("—"*50 + "\n")

        # 绘制预测结果
        plt.figure(figsize=(12, 6))

        # 实际vs预测散点图
        plt.subplot(1, 2, 1)
        plt.scatter(y_test, predictions, alpha=0.6)
        plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k-
        plt.xlabel('Actual Prices')
        plt.ylabel('Predicted Prices')
        plt.title('Actual vs Predicted Prices')
        plt.grid(True)

        # 残差图
        plt.subplot(1, 2, 2)
        residuals = y_test - predictions
        plt.scatter(predictions, residuals, alpha=0.6)
        plt.hlines(0, predictions.min(), predictions.max(), colors='k', linestyl
        plt.xlabel('Predicted Values')
        plt.ylabel('Residuals')
        plt.title('Residual Analysis')
        plt.grid(True)

        plt.tight_layout()
        plt.show()

        return {'mse': mse, 'rmse': rmse, 'mae': mae, 'r2': r2}
```

训练模型

In [7]:
```
trained_model, loss_history = train_model(model, train_loader, epochs=450)
#绘制损失函数
plt.figure(figsize=(10, 6))
plt.plot(loss_history)
```

```
plt.title('Training Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.show()

evaluation_results = evaluate_model(trained_model, xt_test, yt_test)

# 保存模型
torch.save({
    'model_state_dict': trained_model.state_dict(), 'scaler': scaler, 'feature_n
}, "boston_housing_model.pth")
```

```
Epoch 0: Loss = 599.8332, LR = 0.000500
Epoch 15: Loss = 44.9147, LR = 0.000500
Epoch 30: Loss = 10.1640, LR = 0.000500
Epoch 45: Loss = 9.1359, LR = 0.000500
Epoch 60: Loss = 7.8110, LR = 0.000405
Epoch 75: Loss = 7.8752, LR = 0.000365
Epoch 90: Loss = 6.1547, LR = 0.000295
Epoch 105: Loss = 6.2006, LR = 0.000239
Epoch 120: Loss = 6.9160, LR = 0.000174
Epoch 135: Loss = 8.1215, LR = 0.000141
Epoch 150: Loss = 8.0786, LR = 0.000114
Epoch 165: Loss = 7.5247, LR = 0.000083
Epoch 180: Loss = 6.2021, LR = 0.000068
Epoch 195: Loss = 6.6784, LR = 0.000055
Epoch 210: Loss = 5.4881, LR = 0.000040
Epoch 225: Loss = 6.8057, LR = 0.000032
Epoch 240: Loss = 5.1893, LR = 0.000024
Epoch 255: Loss = 6.5747, LR = 0.000019
Epoch 270: Loss = 5.7311, LR = 0.000015
Epoch 285: Loss = 5.7005, LR = 0.000013
Epoch 300: Loss = 5.6183, LR = 0.000009
Epoch 315: Loss = 8.0326, LR = 0.000007
Epoch 330: Loss = 6.2050, LR = 0.000005
Epoch 345: Loss = 7.1189, LR = 0.000004
Epoch 360: Loss = 5.5505, LR = 0.000003
Epoch 375: Loss = 6.4194, LR = 0.000003
Epoch 390: Loss = 5.4920, LR = 0.000002
Epoch 405: Loss = 5.3582, LR = 0.000002
Epoch 420: Loss = 6.1924, LR = 0.000001
Epoch 435: Loss = 6.1144, LR = 0.000001
```



Model Evaluation Metrics:
MSE: 5.6475
RMSE: 2.3764
MAE: 1.8727
R² Score: 0.9095