```python
# Imports
import pandas as pd
import torch
from torch_geometric.data import Data, DataLoader as
GeometricDataLoader
from torch_geometric.nn import GCNConv, global_mean_pool
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
import shap
import numpy as np
import re
import gc
import os

# Set CUDA_LAUNCH_BLOCKING for synchronous error reporting
os.environ["CUDA_LAUNCH_BLOCKING"] = "1"

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if not torch.cuda.is_available():
    raise RuntimeError("CUDA is not available")
print(f"Using device: {device}")
```

```
Using device: cuda
```

```python
def load_and_preprocess_data(csv_path):
    """Load and preprocess the CSV data for battery-relevant
materials."""
    df = pd.read_csv(csv_path)
    df['elements'] = df['elements'].apply(eval)
    df['contains_li'] = df['elements'].apply(lambda x: 'Li' in x)

    filtered_df = df[
        (df['contains_li']) &
        (df['contains_transition_metal']) &
        (df['is_semiconductor']) &
        (df['band_gap'].between(1, 3)) &
        (df['formation_energy_per_atom'] < 0)
    ].copy()

    if filtered_df.empty:
        raise ValueError("Filtered dataset is empty. Adjust filtering
conditions.")

    def parse_formula(formula):
```

```
        elements = re.findall(r'([A-Z][a-z]?)(\d*)', formula)
        total_atoms = sum(int(n) if n else 1 for _, n in elements)
        li_count = next((int(n) if n else 1 for e, n in elements if e
== 'Li'), 0)
        return li_count / total_atoms if total_atoms > 0 else 0

    filtered_df.loc[:, 'li_fraction'] =
filtered_df['formula_pretty'].apply(parse_formula)

    features = ['formation_energy_per_atom', 'energy_per_atom',
'band_gap',
                'density', 'volume', 'n_elements', 'li_fraction']
    X =
filtered_df[features].fillna(filtered_df[features].mean()).values
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    return filtered_df, X_scaled, scaler, features

def create_graph_data(df, X_scaled, targets=None):
    """Convert scaled features into graph data for the GNN."""
    if np.any(np.isnan(X_scaled)) or np.any(np.isinf(X_scaled)):
        raise ValueError("X_scaled contains NaN or infinite values")

    data_list = []
    for i in range(len(df)):
        x = torch.tensor(X_scaled[i],
dtype=torch.float).to(device).reshape(-1, 1)
        num_nodes = x.shape[0]
        edge_index = torch.tensor([[i, j] for i in range(num_nodes)
for j in range(num_nodes) if i != j],
                                  dtype=torch.long,
device=device).t().contiguous()
        data = Data(x=x, edge_index=edge_index)
        if targets is not None:
            data.y = torch.tensor([targets[i]], dtype=torch.float,
device=device)
        data_list.append(data)
    loader = GeometricDataLoader(data_list, batch_size=32,
shuffle=False)
    return loader

class GNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GNN, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, data):
```

```python
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))
        x = global_mean_pool(x, batch)
        return self.fc(x)

class VAE(nn.Module):
    def __init__(self, input_dim, hidden_dim=128, latent_dim=32):
        super(VAE, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc_mu = nn.Linear(hidden_dim, latent_dim)
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)
        self.fc3 = nn.Linear(latent_dim, hidden_dim)
        self.fc4 = nn.Linear(hidden_dim, input_dim)

    def encode(self, x):
        h = F.relu(self.fc1(x))
        return self.fc_mu(h), self.fc_logvar(h)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h = F.relu(self.fc3(z))
        return self.fc4(h)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

def train_gnn(model, loader, optimizer, device):
    model.train()
    total_loss = 0
    for data in loader:
        data = data.to(device)
        optimizer.zero_grad()
        out = model(data).squeeze()  # Remove extra dimension
        loss = F.mse_loss(out, data.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * data.num_graphs
        del data, out, loss
        torch.cuda.empty_cache()
    return total_loss / len(loader.dataset)

def evaluate_gnn(model, loader, device):
    model.eval()
```

```
    total_loss = 0
    predictions = []
    true_values = []
    with torch.no_grad():
        for data in loader:
            data = data.to(device)
            out = model(data).squeeze()  # Remove extra dimension
            loss = F.mse_loss(out, data.y)
            total_loss += loss.item() * data.num_graphs
            predictions.extend(out.cpu().numpy())
            true_values.extend(data.y.cpu().numpy())
            del data, out, loss
            torch.cuda.empty_cache()
    return total_loss / len(loader.dataset), predictions, true_values

def train_vae(model, data, optimizer, device):
    model.train()
    optimizer.zero_grad()
    recon, mu, logvar = model(data)
    loss = F.mse_loss(recon, data, reduction='sum') + (-0.5 *
torch.sum(1 + logvar - mu.pow(2) - logvar.exp()))
    loss.backward()
    optimizer.step()
    loss_value = loss.item()
    del recon, mu, logvar, loss
    torch.cuda.empty_cache()
    return loss_value

def generate_structures(vae, num_samples, latent_dim, device):
    vae.eval()
    with torch.no_grad():
        z = torch.randn(num_samples, latent_dim, device=device)
        samples = vae.decode(z)
    samples_cpu = samples.cpu().numpy()
    del z, samples
    torch.cuda.empty_cache()
    return samples_cpu

def main(csv_path):
    # Preprocess data
    df, X_scaled, scaler, features =
load_and_preprocess_data(csv_path)

    # Split data
    X_train, X_temp, y_train, y_temp = train_test_split(X_scaled,
df['band_gap'].values, test_size=0.3, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

    # Validate data consistency
```

```
    assert len(df.iloc[:len(X_train)]) == len(X_train), "Train data
length mismatch"
    assert len(df.iloc[len(X_train):len(X_train)+len(X_val)]) ==
len(X_val), "Val data length mismatch"
    assert len(df.iloc[len(X_train)+len(X_val):]) == len(X_test),
"Test data length mismatch"

    # Create data loaders
    train_loader = create_graph_data(df.iloc[:len(X_train)], X_train,
targets=y_train)
    val_loader = create_graph_data(df.iloc[len(X_train):len(X_train)
+len(X_val)], X_val, targets=y_val)
    test_loader = create_graph_data(df.iloc[len(X_train)+len(X_val):],
X_test, targets=y_test)

    # Define model parameters
    input_dim = 1
    hidden_dim = 64
    output_dim = 1
    vae_input_dim = X_scaled.shape[1]
    latent_dim = 32

    # Initialize models
    gnn = GNN(input_dim=input_dim, hidden_dim=hidden_dim,
output_dim=output_dim).to(device)
    vae = VAE(input_dim=vae_input_dim).to(device)

    # Train GNN
    optimizer_gnn = optim.Adam(gnn.parameters(), lr=0.001)
    train_losses, val_losses = [], []
    for epoch in range(50):
        train_loss = train_gnn(gnn, train_loader, optimizer_gnn,
device)
        val_loss, _, _ = evaluate_gnn(gnn, val_loader, device)
        train_losses.append(train_loss)
        val_losses.append(val_loss)
        if epoch % 10 == 0:
            print(f"Epoch {epoch}, GNN Train Loss: {train_loss:.4f},
Val Loss: {val_loss:.4f}")
        gc.collect()
        torch.cuda.empty_cache()

    # Plot GNN Training and Validation Loss
    plt.figure(figsize=(8, 6))
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('MSE Loss')
    plt.title('GNN Training and Validation Loss')
```

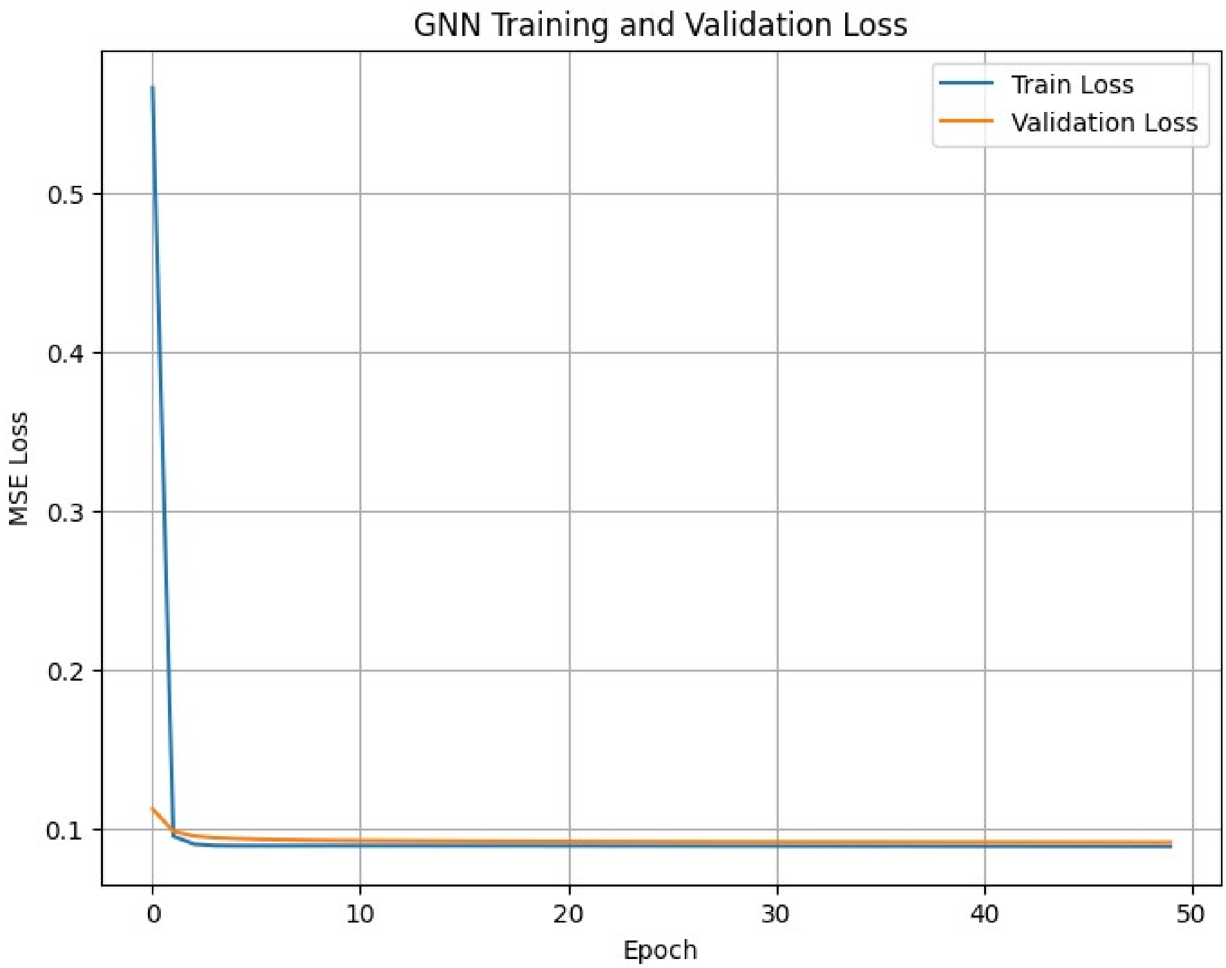
```
    plt.legend()
    plt.grid(True)
    plt.show()

    # Train VAE
    optimizer_vae = optim.Adam(vae.parameters(), lr=0.001)
    X_train_tensor = torch.tensor(X_train, dtype=torch.float,
device=device)
    vae_losses = []
    for epoch in range(50):
        loss = train_vae(vae, X_train_tensor, optimizer_vae, device)
        vae_losses.append(loss)
        if epoch % 10 == 0:
            print(f"Epoch {epoch}, VAE Loss: {loss:.4f}")
        gc.collect()
        torch.cuda.empty_cache()

    # Plot VAE Training Loss
    plt.figure(figsize=(8, 6))
    plt.plot(vae_losses, label='VAE Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('VAE Training Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Inference: Generate new structures
    num_samples = 100
    new_structures_scaled = generate_structures(vae,
num_samples=num_samples, latent_dim=latent_dim, device=device)
    new_structures_unscaled =
scaler.inverse_transform(new_structures_scaled)
    new_df = pd.DataFrame(new_structures_unscaled, columns=features)
    new_loader = create_graph_data(new_df, new_structures_scaled)

    # Predict band gaps for new structures
    predictions_new = []
    with torch.no_grad():
        for data in new_loader:
            data = data.to(device)
            pred = gnn(data).squeeze()
            predictions_new.extend(pred.cpu().numpy())
            del data, pred
            torch.cuda.empty_cache()
    new_df.loc[:, 'predicted_band_gap'] = predictions_new

    # Plot histogram of predicted band gaps
    plt.figure(figsize=(8, 6))
```

```
    plt.hist(new_df['predicted_band_gap'], bins=20, color='skyblue',
edgecolor='black')
    plt.xlabel('Predicted Band Gap (eV)')
    plt.ylabel('Frequency')
    plt.title('Histogram of Predicted Band Gaps for Generated
Structures')
    plt.grid(True)
    plt.show()

    # Evaluations on Test Set
    test_loss, predictions_test, true_values = evaluate_gnn(gnn,
test_loader, device)
    mae = mean_absolute_error(true_values, predictions_test)
    mse = mean_squared_error(true_values, predictions_test)
    print(f"Test Set Evaluation - Loss: {test_loss:.4f}, MAE:
{mae:.4f}, MSE: {mse:.4f}")

    # Plot true vs predicted band gaps
    plt.figure(figsize=(8, 6))
    plt.scatter(true_values, predictions_test, alpha=0.5)
    plt.plot([min(true_values), max(true_values)], [min(true_values),
max(true_values)], 'r--')
    plt.xlabel('True Band Gap (eV)')
    plt.ylabel('Predicted Band Gap (eV)')
    plt.title('True vs Predicted Band Gaps on Test Set')
    plt.grid(True)
    plt.show()

    # XAI with SHAP (using KernelExplainer)
    background_loader = create_graph_data(df.iloc[:len(X_train)],
X_train, targets=y_train)
    background_batch = next(iter(background_loader)).to(device)

    def gnn_predict(features):
        # Ensure features is 2D and has enough nodes
        if len(features.shape) == 1:
            features = features.reshape(1, -1)
        num_graphs = features.shape[0]
        num_nodes_per_graph = max(len(features[0]), 1)  # Ensure at
least 1 node
        total_nodes = num_graphs * num_nodes_per_graph

        # Reshape features to match expected node structure
        features_tensor = torch.tensor(features, dtype=torch.float,
device=device).reshape(total_nodes, 1)

        # Generate edge_index, ensuring non-empty even for single node
        if total_nodes > 1:
            edge_index = torch.tensor([[i, j] for i in
```

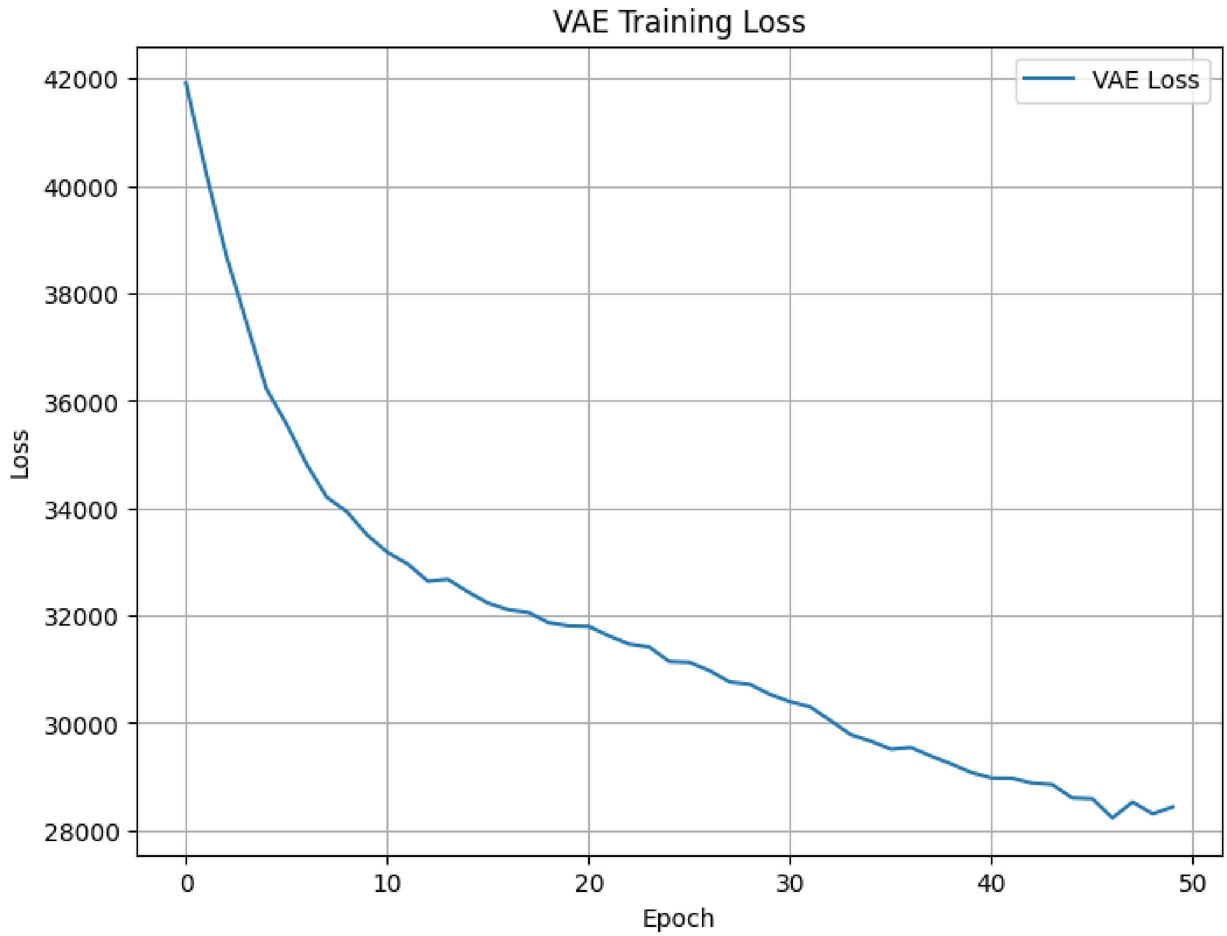
```
range(total_nodes) for j in range(total_nodes) if i != j],
                                      dtype=torch.long,
device=device).t().contiguous()
        else:
            # For a single node, create a self-loop
            edge_index = torch.tensor([[0, 0]], dtype=torch.long,
device=device).t().contiguous()

        # Batch tensor
        batch = torch.cat([torch.full((num_nodes_per_graph,), i,
dtype=torch.long, device=device)
                           for i in range(num_graphs)], dim=0)

        data = Data(x=features_tensor, edge_index=edge_index,
batch=batch)
        with torch.no_grad():
            out = gnn(data).cpu().numpy()
        del data, features_tensor, edge_index, batch
        torch.cuda.empty_cache()
        return out

    new_batch = next(iter(new_loader)).to(device)
    background_data = shap.sample(background_batch.x.cpu().numpy(),
50)
    explainer = shap.KernelExplainer(gnn_predict, background_data)
    shap_values = explainer.shap_values(new_batch.x.cpu().numpy(),
nsamples=50)

    shap.summary_plot(shap_values, features=new_batch.x.cpu().numpy(),

                      feature_names=features, plot_type="bar")
    print("SHAP summary plot generated for feature importance in GNN
predictions")

    # Final cleanup
    del train_loader, val_loader, test_loader, new_loader,
X_train_tensor, background_batch, new_batch
    gc.collect()
    torch.cuda.empty_cache()

    # Save outputs
    torch.save(gnn.state_dict(), 'gnn_model.pt')
    torch.save(vae.state_dict(), 'vae_model.pt')
    new_df.to_csv('generated_structures.csv', index=False)
    print("Models saved to 'gnn_model.pt' and 'vae_model.pt',
structures saved to 'generated_structures.csv'")

if __name__ == "__main__":
    csv_path =
```

```
"/kaggle/input/material-science/lithium_battery_materials.csv"
    main(csv_path)
```

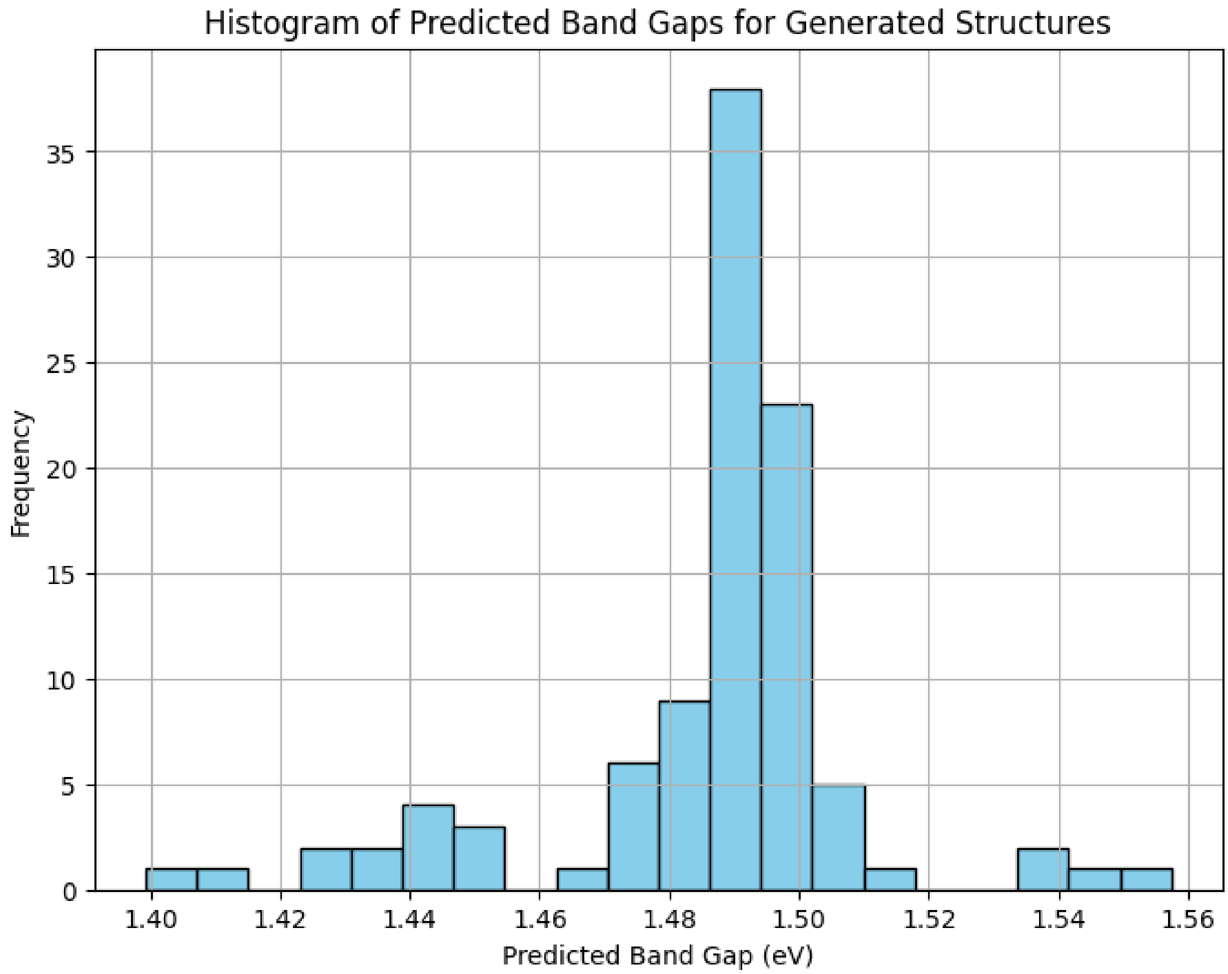
```
'data.DataLoader' is deprecated, use 'loader.DataLoader' instead

Epoch 0, GNN Train Loss: 0.5663, Val Loss: 0.1129
Epoch 10, GNN Train Loss: 0.0898, Val Loss: 0.0929
Epoch 20, GNN Train Loss: 0.0898, Val Loss: 0.0924
Epoch 30, GNN Train Loss: 0.0896, Val Loss: 0.0922
Epoch 40, GNN Train Loss: 0.0895, Val Loss: 0.0920
```



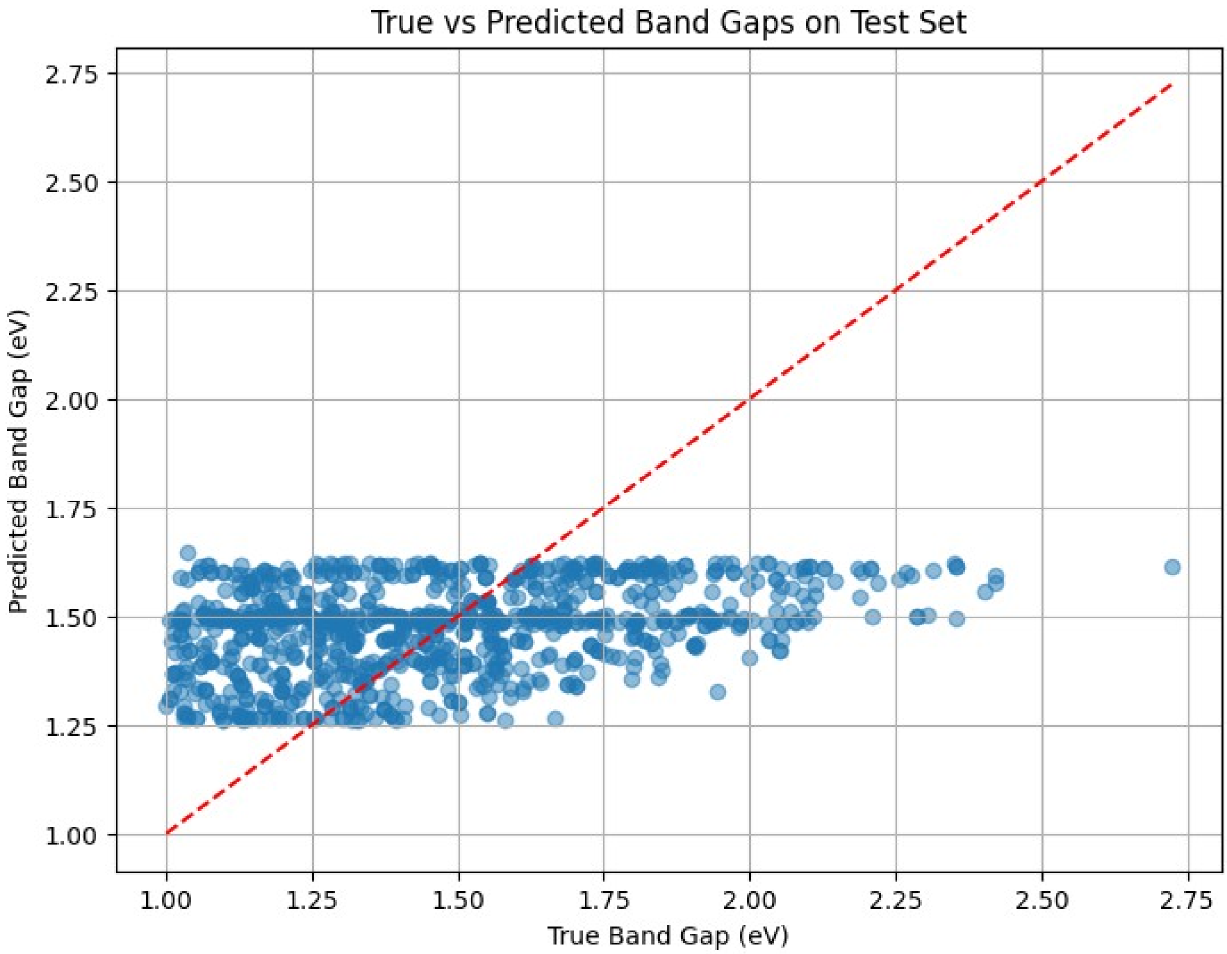

```
Epoch 0, VAE Loss: 41922.2031
Epoch 10, VAE Loss: 33179.4414
Epoch 20, VAE Loss: 31798.9023
Epoch 30, VAE Loss: 30397.9375
Epoch 40, VAE Loss: 28975.4492
```




```
'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
```

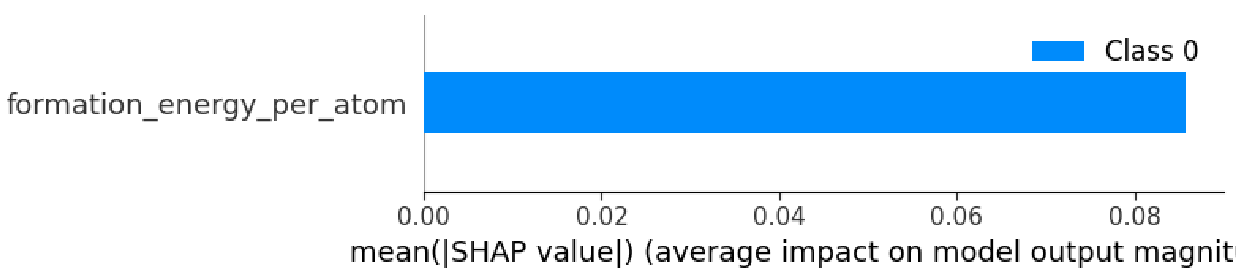



```
Test Set Evaluation - Loss: 0.0845, MAE: 0.2374, MSE: 0.0845
```




```
'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
```

{"model_id":"43170b19b988456c95c9a67aad80b8e7","version_major":2,"version_minor":0}




```
SHAP summary plot generated for feature importance in GNN predictions
Models saved to 'gnn_model.pt' and 'vae_model.pt', structures saved to
'generated_structures.csv'
```