In [9]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

In [16]:
```python
#  Dataset Loading
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist.data, mnist.target.astype(np.int8)
```

In [22]:
```python
X = X / 255.0  # Normalize pixel values
# Reduce dimensionality using PCA
pca = PCA(n_components=50)  # You can try 30–100 and tune this
X_pca = pca.fit_transform(X)
```

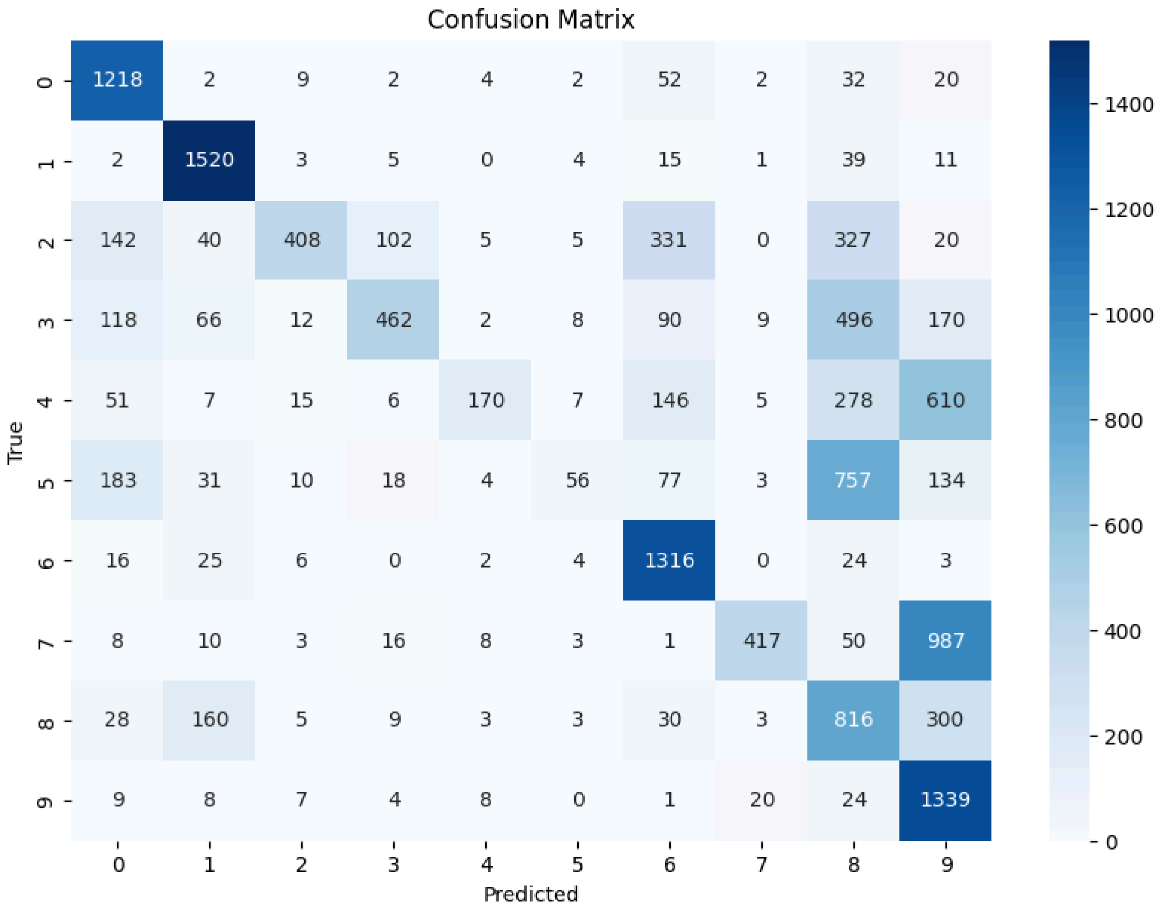
In [31]:
```python
# Step 3: Model Development (Using GaussianNB as approximation to Bayes' Decisio
model = GaussianNB()

# Step 4: Training and Testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [32]:
```python
# Step 5: Evaluation
print("\nEvaluation Metrics:")
print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Evaluation Metrics:
Accuracy: 55.16%

Classification Report:
               precision    recall  f1-score   support

           0       0.69      0.91      0.78      1343
           1       0.81      0.95      0.88      1600
           2       0.85      0.30      0.44      1380
           3       0.74      0.32      0.45      1433
           4       0.83      0.13      0.23      1295
           5       0.61      0.04      0.08      1273
           6       0.64      0.94      0.76      1396
           7       0.91      0.28      0.42      1503
           8       0.29      0.60      0.39      1357
           9       0.37      0.94      0.53      1420

    accuracy                           0.55     14000
   macro avg       0.67      0.54      0.50     14000
weighted avg       0.68      0.55      0.51     14000
```

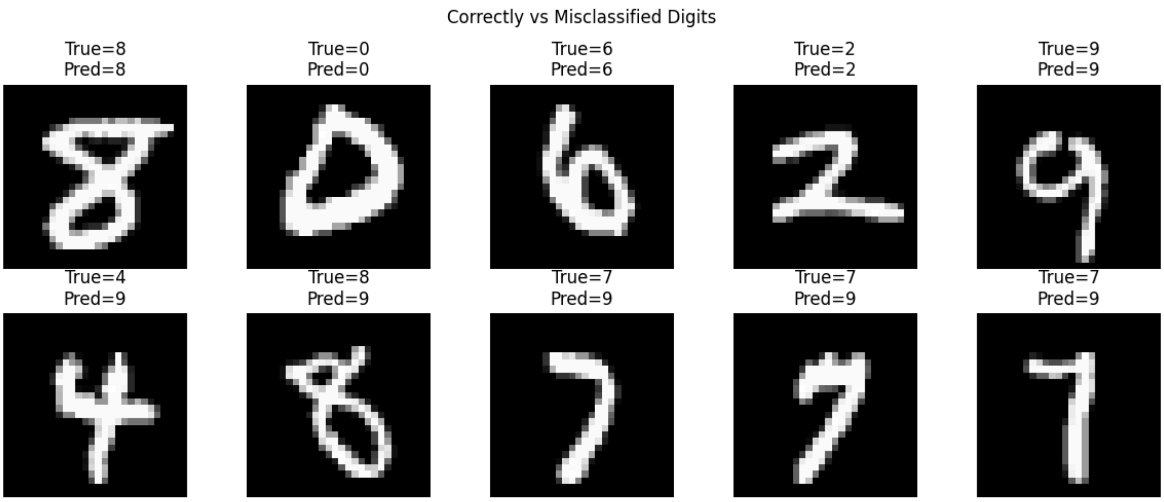
In [25]:
```python
conf_mat = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
```

```
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```



In [29]:

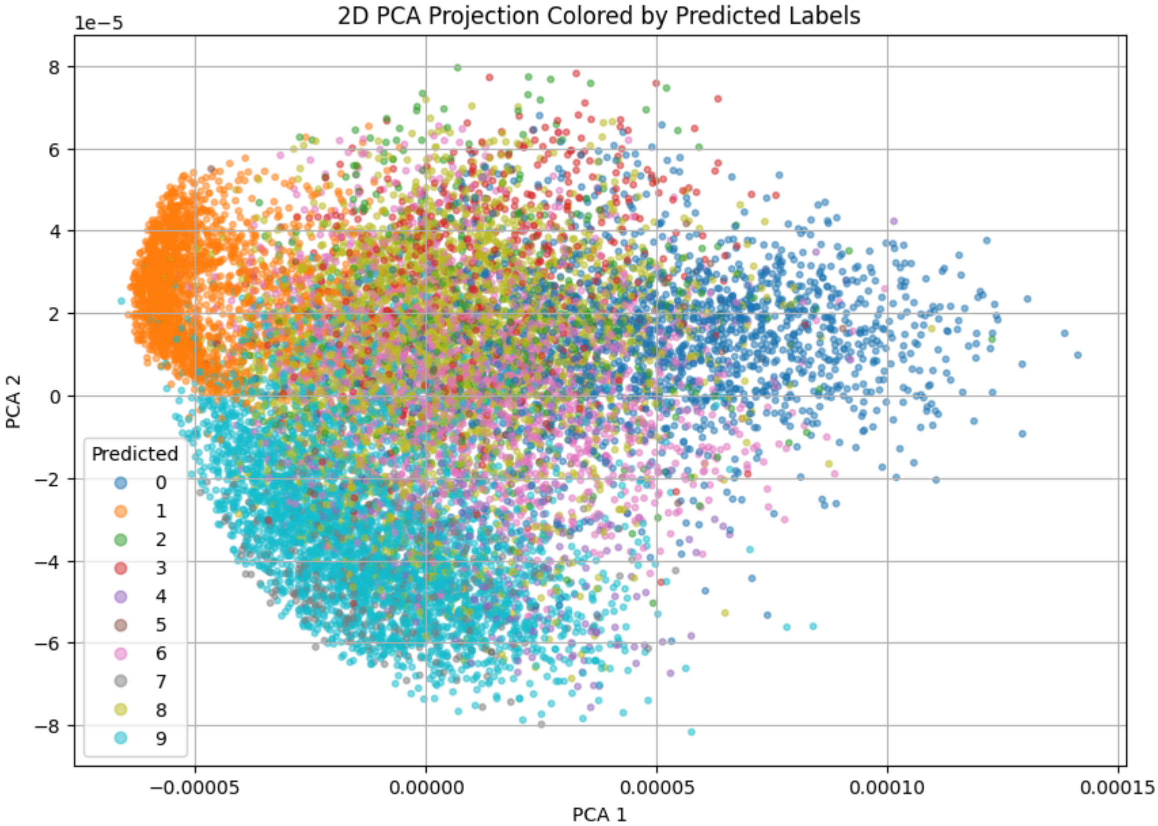
```
# Visualization - Correct vs Misclassified
correct = np.where(y_pred == y_test)[0]
incorrect = np.where(y_pred != y_test)[0]

plt.figure(figsize=(12, 5))
for i, idx in enumerate(correct[:5]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True={y_test[idx]}\nPred={y_pred[idx]}")
    plt.axis('off')

for i, idx in enumerate(incorrect[:5]):
    plt.subplot(2, 5, i + 6)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True={y_test[idx]}\nPred={y_pred[idx]}")
    plt.axis('off')
plt.suptitle("Correctly vs Misclassified Digits")
plt.tight_layout()
plt.show()
```



In [27]:

```
# Visualize Decision Boundaries with PCA (2D)
print("Reducing dimensionality for visualization...")
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_test)

plt.figure(figsize=(10, 7))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_pred, cmap='tab10', alpha=0.
plt.legend(*scatter.legend_elements(), title="Predicted")
plt.title("2D PCA Projection Colored by Predicted Labels")
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.grid(True)
plt.show()
```

```
Reducing dimensionality for visualization...
```



In [ ]: