

```
In [1]: # Step 1: Mount Google Drive
# This step connects Google Drive to access the saved models and datasets.

from google.colab import drive

drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [2]: # Step 2: Install and Import Required Libraries
# This step installs timm and imports all libraries required for the final model comparison.

!pip install -q timm

import os
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
import torch
import timm

from torchvision import datasets, transforms
from torch.utils.data import DataLoader

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications.resnet50 import preprocess_input as resnet_preprocess
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as mobilenet_preprocess

print("Libraries imported successfully.")
```

Libraries imported successfully.

```
In [3]: # Step 3: Define Model and Dataset Paths
# This step defines the saved model paths and the correct test dataset path for each model.

# Test dataset paths
RESNET_TEST_PATH = "/content/drive/MyDrive/ASL_Split_Dataset_ResNet/test"
MOBILENET_TEST_PATH = "/content/drive/MyDrive/ASL_Split_Dataset_MobileNet/test"
EFFICIENTNET_TEST_PATH = "/content/drive/MyDrive/ASL_Split_Dataset_EfficientNet/test"
SWIN_TEST_PATH = "/content/drive/MyDrive/ASL_Split_Dataset_Swin/test"

# TensorFlow model paths
RESNET_MODEL_PATH = "/content/drive/MyDrive/ASL_Project/final_resnet50_asl_model.keras"
MOBILENET_MODEL_PATH = "/content/drive/MyDrive/ASL_MobileNet_Project/final_mobilenetv2_asl_model.keras"
EFFICIENTNET_MODEL_PATH = "/content/drive/MyDrive/ASL_EfficientNet_Project/final_efficientnet_asl_model.keras"

# Swin Transformer model paths
SWIN_MODEL_PATH = "/content/drive/MyDrive/ASL_Transformer_Project/final_swin_asl_model.pth"
SWIN_CLASS_NAMES_PATH = "/content/drive/MyDrive/ASL_Transformer_Project/swin_class_names.txt"

print("All paths are ready.")
```

All paths are ready.

```
In [4]: # Step 4: Load All Saved Models
# This step loads the three TensorFlow models and the Swin Transformer model.

# Load TensorFlow models
resnet_model = tf.keras.models.load_model(RESNET_MODEL_PATH)
mobilenet_model = tf.keras.models.load_model(MOBILENET_MODEL_PATH)
efficientnet_model = tf.keras.models.load_model(EFFICIENTNET_MODEL_PATH)

print("TensorFlow models loaded successfully.")

# Load Swin Transformer class names
with open(SWIN_CLASS_NAMES_PATH, "r") as file:
    SWIN_CLASS_NAMES = [line.strip() for line in file.readlines()]

NUM_CLASSES = len(SWIN_CLASS_NAMES)

# Load Swin Transformer model
swin_model = timm.create_model(
    "swin_tiny_patch4_window7_224",
    pretrained=False,
    num_classes=NUM_CLASSES
)

swin_model.load_state_dict(
    torch.load(SWIN_MODEL_PATH, map_location="cpu")
)

swin_model.eval()

print("Swin Transformer model loaded successfully.")
print("All models loaded successfully.")
```

TensorFlow models loaded successfully.
Swin Transformer model loaded successfully.
All models loaded successfully.

In [5]: # Step 5: Create Test Data Generators
This step creates the correct test data generator for each TensorFlow model.

```
IMG_SIZE = 224
BATCH_SIZE = 16

# ResNet50 Test Generator
resnet_test_generator = ImageDataGenerator(
    preprocessing_function=resnet_preprocess
).flow_from_directory(
    RESNET_TEST_PATH,
    target_size=(IMG_SIZE, IMG_SIZE),
    classes=[
        '0','1','2','3','4','5','6','7','8','9',
        'a','b','c','d','e','f','g','h','i','j',
        'k','l','m','n','o','p','q','r','s','t',
        'u','v','w','x','y','z'
    ],
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

# MobileNetV2 Test Generator
mobilenet_test_generator = ImageDataGenerator(
    preprocessing_function=mobilenet_preprocess
).flow_from_directory(
    MOBILENET_TEST_PATH,
    target_size=(IMG_SIZE, IMG_SIZE),
    classes=[
        '0','1','2','3','4','5','6','7','8','9',
        'a','b','c','d','e','f','g','h','i','j',
        'k','l','m','n','o','p','q','r','s','t',
        'u','v','w','x','y','z'
    ],
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

# EfficientNet Test Generator
efficientnet_test_generator = ImageDataGenerator().flow_from_directory(
    EFFICIENTNET_TEST_PATH,
    target_size=(IMG_SIZE, IMG_SIZE),
    classes=[
        '0','1','2','3','4','5','6','7','8','9',
        'a','b','c','d','e','f','g','h','i','j',
        'k','l','m','n','o','p','q','r','s','t',
        'u','v','w','x','y','z'
    ],
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("All TensorFlow test generators are ready.")
```

Found 505 images belonging to 36 classes.
Found 505 images belonging to 36 classes.
Found 539 images belonging to 36 classes.
All TensorFlow test generators are ready.

In [6]: # Step 6: Create Swin Transformer Test DataLoader
This step creates the correct PyTorch DataLoader for the Swin Transformer model.

```
swin_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

swin_test_dataset = datasets.ImageFolder(
    root=SWIN_TEST_PATH,
    transform=swin_transform
)

swin_test_loader = DataLoader(
    swin_test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False
)

print("Swin Transformer test DataLoader is ready.")
print("Swin Test Images:", len(swin_test_dataset))
print("Swin Classes:", swin_test_dataset.classes)
```

Swin Transformer test DataLoader is ready.
Swin Test Images: 505
Swin Classes: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

In [7]: # Step 7: Define Evaluation Function for TensorFlow Models
This step evaluates TensorFlow models using accuracy, precision, recall, F1-score, and inference time.

```
def evaluate_tensorflow_model(model, test_generator, model_name):

    test_generator.reset()

    start_time = time.time()

    predictions = model.predict(
        test_generator,
        verbose=0
    )

    inference_time = time.time() - start_time

    y_pred = np.argmax(predictions, axis=1)
    y_true = test_generator.classes

    accuracy = accuracy_score(y_true, y_pred)

    precision = precision_score(
        y_true,
        y_pred,
        average="weighted",
        zero_division=0
    )

    recall = recall_score(
        y_true,
        y_pred,
        average="weighted",
        zero_division=0
    )

    f1 = f1_score(
        y_true,
        y_pred,
        average="weighted",
        zero_division=0
    )

    return {
        "Model": model_name,
        "Test Images": test_generator.n,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1-Score": f1,
        "Inference Time (s)": inference_time
    }
```

```
In [8]: # Step 8: Define Evaluation Function for Swin Transformer
# This step evaluates the Swin Transformer model using accuracy, precision, recall, F1-score, and inference time.

def evaluate_swin_model(model, test_loader, model_name):

    model.eval()

    all_predictions = []
    all_labels = []

    start_time = time.time()

    with torch.no_grad():

        for images, labels in test_loader:

            outputs = model(images)

            _, predicted = torch.max(outputs, 1)

            all_predictions.extend(predicted.numpy())
            all_labels.extend(labels.numpy())

    inference_time = time.time() - start_time

    accuracy = accuracy_score(all_labels, all_predictions)

    precision = precision_score(
        all_labels,
        all_predictions,
        average="weighted",
        zero_division=0
    )

    recall = recall_score(
        all_labels,
        all_predictions,
        average="weighted",
        zero_division=0
    )

    f1 = f1_score(
        all_labels,
        all_predictions,
        average="weighted",
        zero_division=0
    )

    return {
        "Model": model_name,
        "Test Images": len(all_labels),
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1-Score": f1,
        "Inference Time (s)": inference_time
    }
```

```
In [9]: # Step 9: Run Final Model Comparison
# This step evaluates all four models using their correct test datasets and preprocessing methods.

results = []

results.append(
    evaluate_tensorflow_model(
        resnet_model,
        resnet_test_generator,
        "ResNet50"
    )
)

results.append(
    evaluate_tensorflow_model(
        mobilenet_model,
        mobilenet_test_generator,
        "MobileNetV2"
    )
)

results.append(
    evaluate_tensorflow_model(
        efficientnet_model,
        efficientnet_test_generator,
        "EfficientNetB0"
    )
)

results.append(
    evaluate_swin_model(
        swin_model,
        swin_test_loader,
        "Swin Transformer"
    )
)

comparison_df = pd.DataFrame(results)

comparison_df = comparison_df.sort_values(
    by="Accuracy",
    ascending=False
).reset_index(drop=True)

comparison_df
```

Out[9]:

	Model	Test Images	Accuracy	Precision	Recall	F1-Score	Inference Time (s)
0	Swin Transformer	505	0.992079	0.992838	0.992079	0.992054	275.576531
1	MobileNetV2	505	0.879208	0.890759	0.879208	0.877554	143.833358
2	ResNet50	505	0.873267	0.888986	0.873267	0.873152	208.556154
3	EfficientNetB0	539	0.736549	0.815643	0.736549	0.731494	166.182902

```
In [10]: # Step 10: Save Comparison Results
# This step saves the comparison table as CSV and Excel files.

comparison_df.to_csv(
    "/content/drive/MyDrive/ASL_Model_Comparison_Results.csv",
    index=False
)

comparison_df.to_excel(
    "/content/drive/MyDrive/ASL_Model_Comparison_Results.xlsx",
    index=False
)

print("Comparison results saved successfully.")

Comparison results saved successfully.
```

```
In [11]: # Step 11: Plot Accuracy Comparison
# This step visualizes the test accuracy comparison among all models.

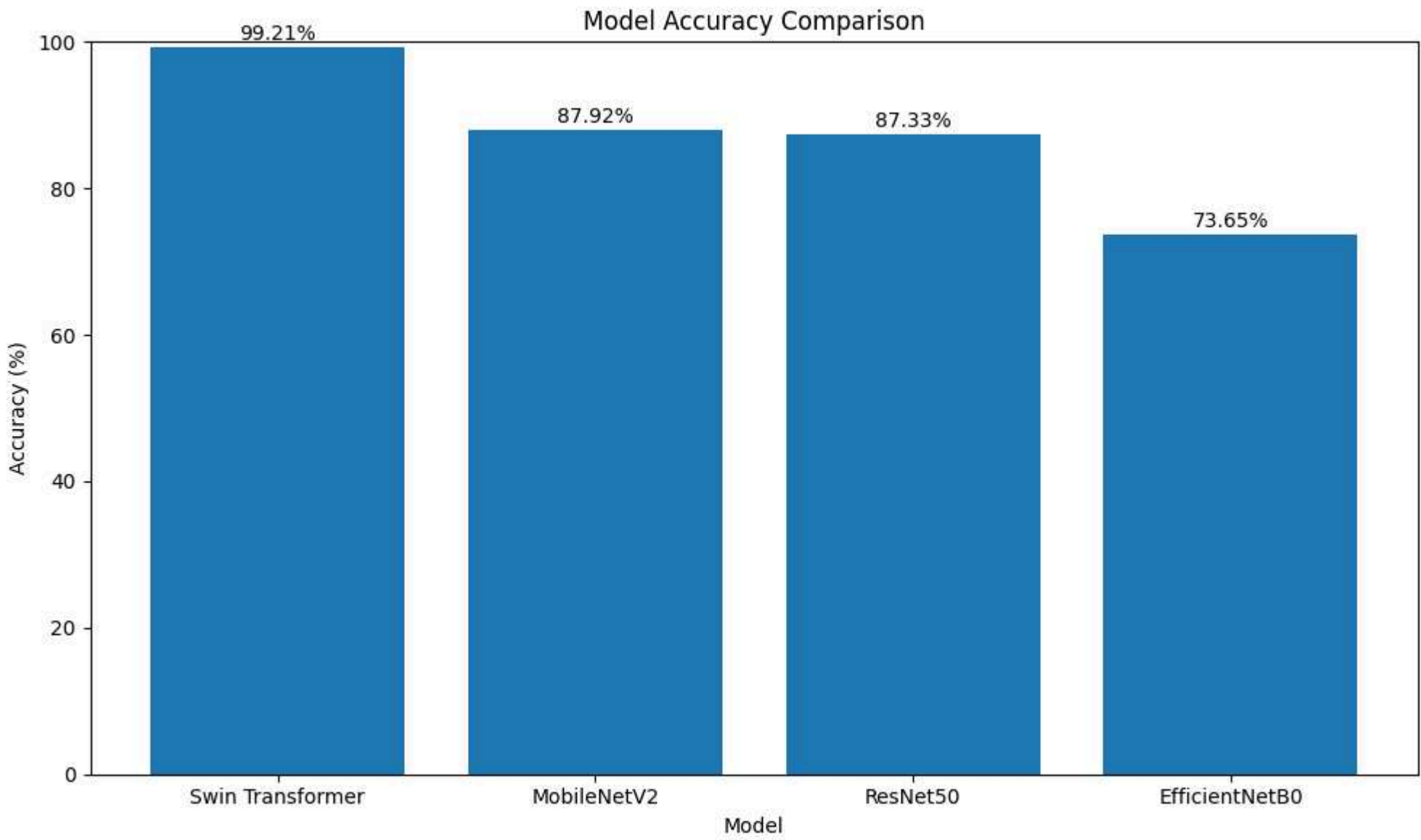
plt.figure(figsize=(10, 6))

plt.bar(
    comparison_df["Model"],
    comparison_df["Accuracy"] * 100
)

plt.title("Model Accuracy Comparison")
plt.xlabel("Model")
plt.ylabel("Accuracy (%)")
plt.ylim(0, 100)

for index, value in enumerate(comparison_df["Accuracy"] * 100):
    plt.text(
        index,
        value + 1,
        f"{value:.2f}%",
        ha="center"
    )

plt.tight_layout()
plt.show()
```



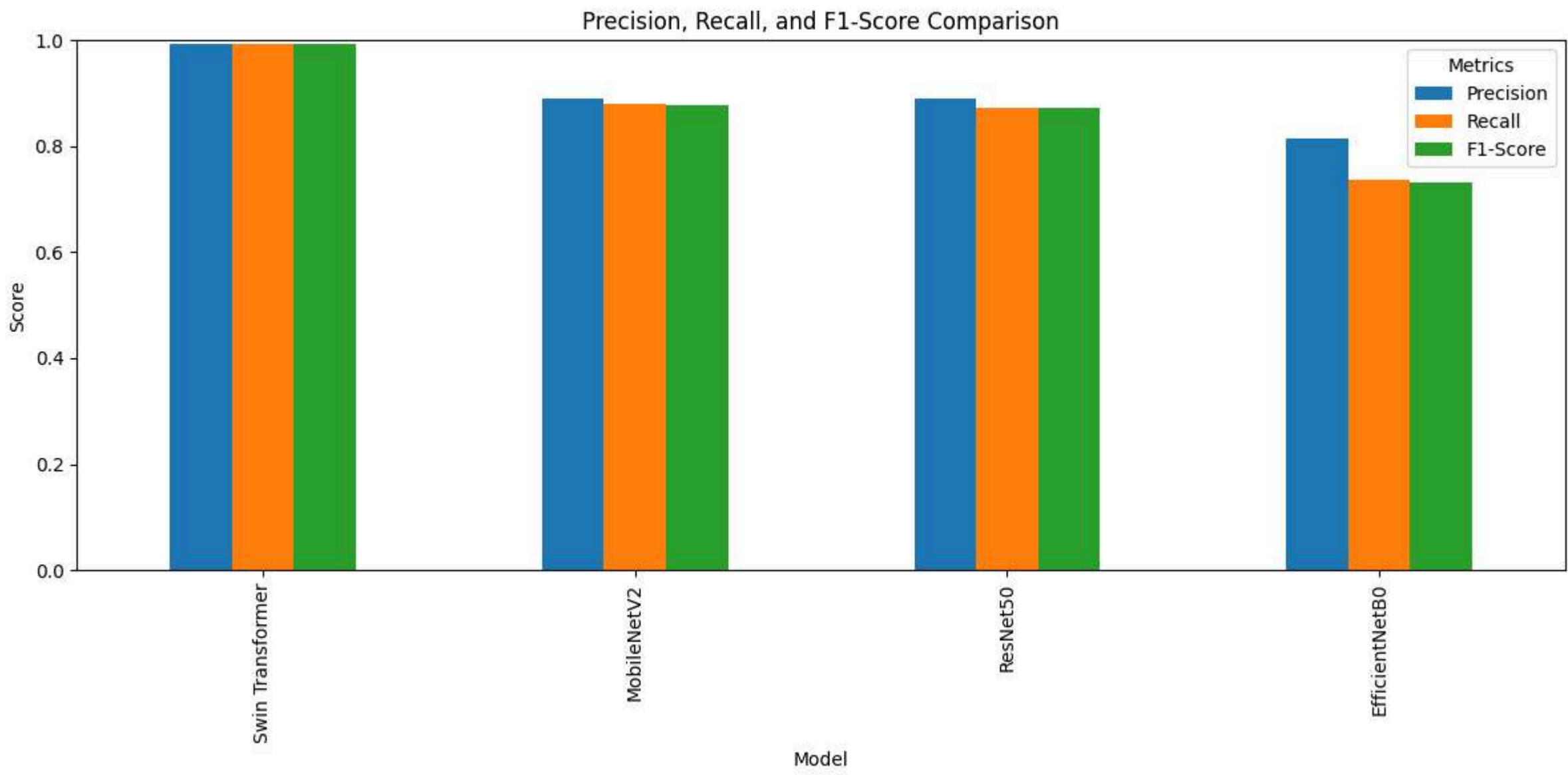
```
In [12]: # Step 12: Plot Precision, Recall, and F1-Score Comparison
# This step compares the main evaluation metrics across all models.

metrics = ["Precision", "Recall", "F1-Score"]

comparison_df.plot(
    x="Model",
    y=metrics,
    kind="bar",
    figsize=(12, 6)
)

plt.title("Precision, Recall, and F1-Score Comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.ylim(0, 1)

plt.legend(title="Metrics")
plt.tight_layout()
plt.show()
```



```
In [13]: # Step 13: Plot Inference Time Comparison
# This step compares the inference time of all models.
```

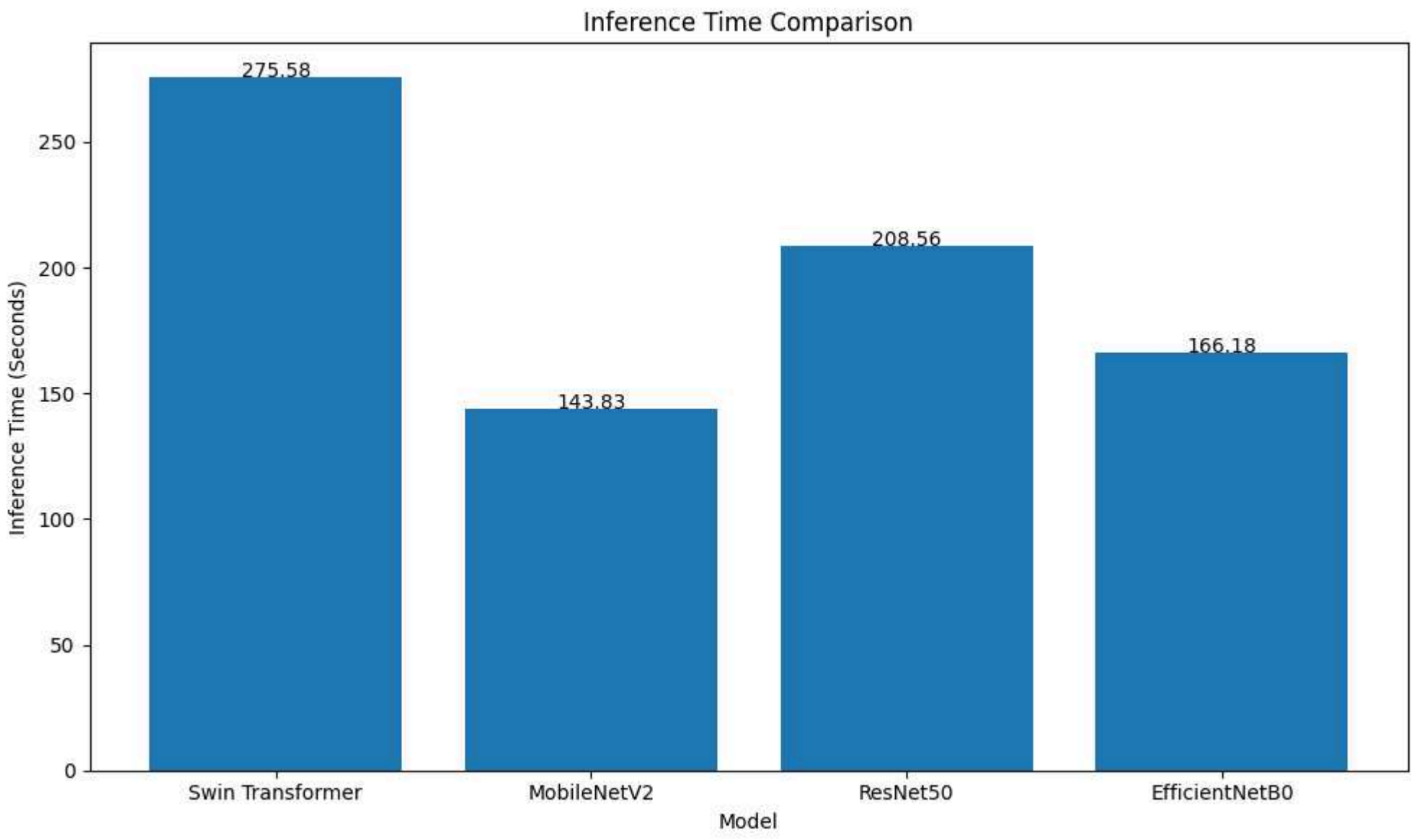
```
plt.figure(figsize=(10, 6))

plt.bar(
    comparison_df["Model"],
    comparison_df["Inference Time (s)"]
)

plt.title("Inference Time Comparison")
plt.xlabel("Model")
plt.ylabel("Inference Time (Seconds)")

for index, value in enumerate(comparison_df["Inference Time (s)"]):
    plt.text(
        index,
        value + 0.05,
        f"{value:.2f}",
        ha="center"
    )

plt.tight_layout()
plt.show()
```



```
In [14]: # Step 14: Select the Best Performing Model
# This step identifies the best model based on test accuracy.
```

```
best_model = comparison_df.iloc[0]

print("=" * 60)
print("FINAL MODEL COMPARISON SUMMARY")
print("=" * 60)

print(f"Best Model      : {best_model['Model']}")
print(f"Test Accuracy    : {best_model['Accuracy']*100:.2f}%")
print(f"Precision        : {best_model['Precision']:.4f}")
print(f"Recall           : {best_model['Recall']:.4f}")
print(f"F1-Score         : {best_model['F1-Score']:.4f}")
print(f"Inference Time   : {best_model['Inference Time (s)']:.2f} seconds")

print("=" * 60)
print("Ranking of Models")
print("=" * 60)

display(comparison_df)
```

```
=====
FINAL MODEL COMPARISON SUMMARY
=====
Best Model      : Swin Transformer
Test Accuracy   : 99.21%
Precision       : 0.9928
Recall          : 0.9921
F1-Score        : 0.9921
Inference Time  : 275.58 seconds
=====
Ranking of Models
=====
```

	Model	Test Images	Accuracy	Precision	Recall	F1-Score	Inference Time (s)
0	Swin Transformer	505	0.992079	0.992838	0.992079	0.992054	275.576531
1	MobileNetV2	505	0.879208	0.890759	0.879208	0.877554	143.833358
2	ResNet50	505	0.873267	0.888986	0.873267	0.873152	208.556154
3	EfficientNetB0	539	0.736549	0.815643	0.736549	0.731494	166.182902

```
In [15]: # Step 15: Save Figures
# This step saves all generated figures as high-quality PNG images.

# Accuracy Comparison
plt.figure(figsize=(10,6))
plt.bar(comparison_df["Model"], comparison_df["Accuracy"] * 100)
plt.title("Model Accuracy Comparison")
plt.xlabel("Model")
plt.ylabel("Accuracy (%)")
plt.ylim(0,100)

for i, value in enumerate(comparison_df["Accuracy"] * 100):
    plt.text(i, value + 1, f"{value:.2f}%", ha="center")

plt.tight_layout()
plt.savefig(
    "/content/drive/MyDrive/Accuracy_Comparison.png",
    dpi=300,
    bbox_inches="tight"
)
plt.show()

# Precision / Recall / F1
comparison_df.plot(
    x="Model",
    y=["Precision", "Recall", "F1-Score"],
    kind="bar",
    figsize=(12,6)
)

plt.title("Evaluation Metrics Comparison")
plt.ylabel("Score")
plt.ylim(0,1)

plt.tight_layout()
plt.savefig(
    "/content/drive/MyDrive/Metrics_Comparison.png",
    dpi=300,
    bbox_inches="tight"
)
plt.show()

# Inference Time
plt.figure(figsize=(10,6))

plt.bar(
    comparison_df["Model"],
    comparison_df["Inference Time (s)"]
)

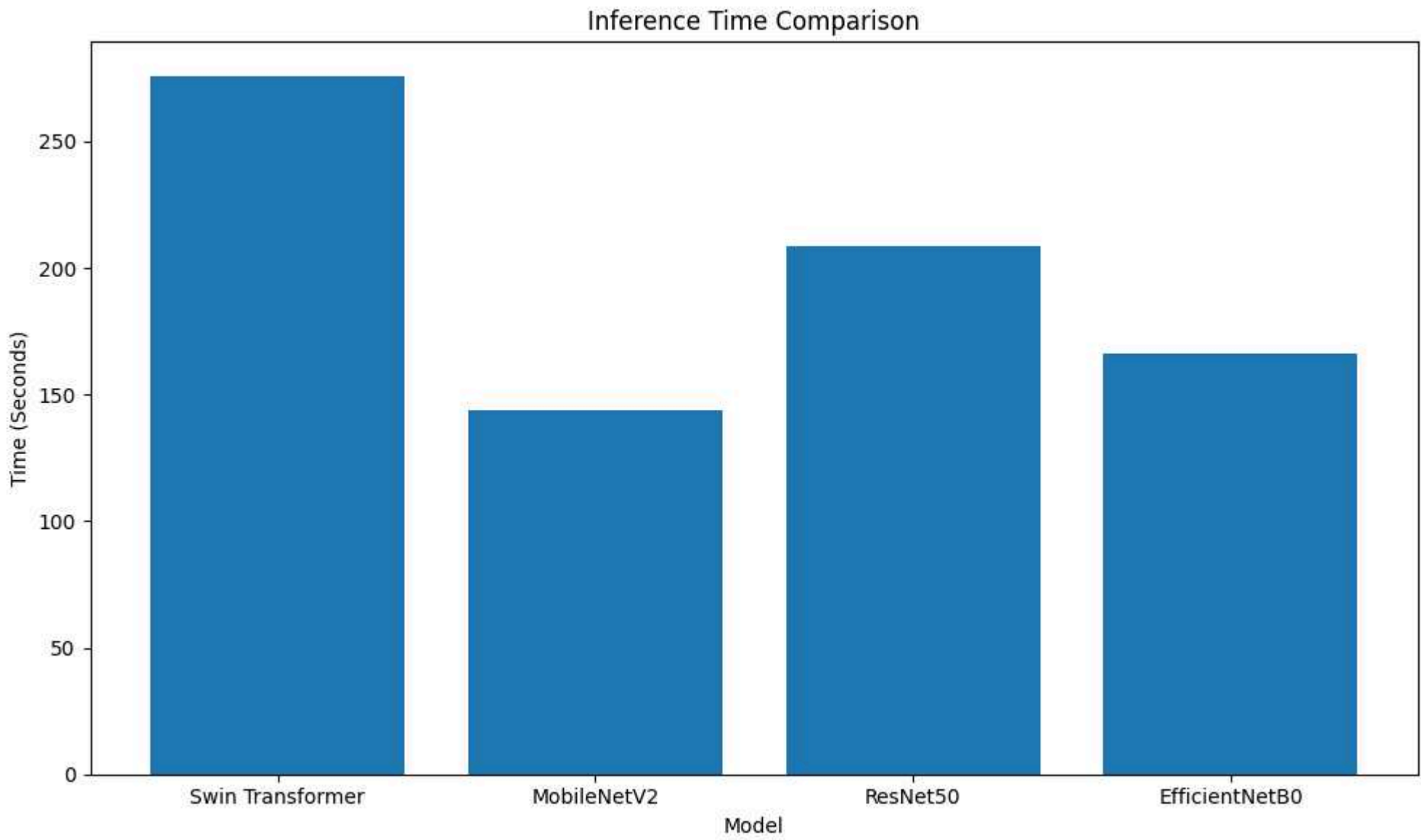
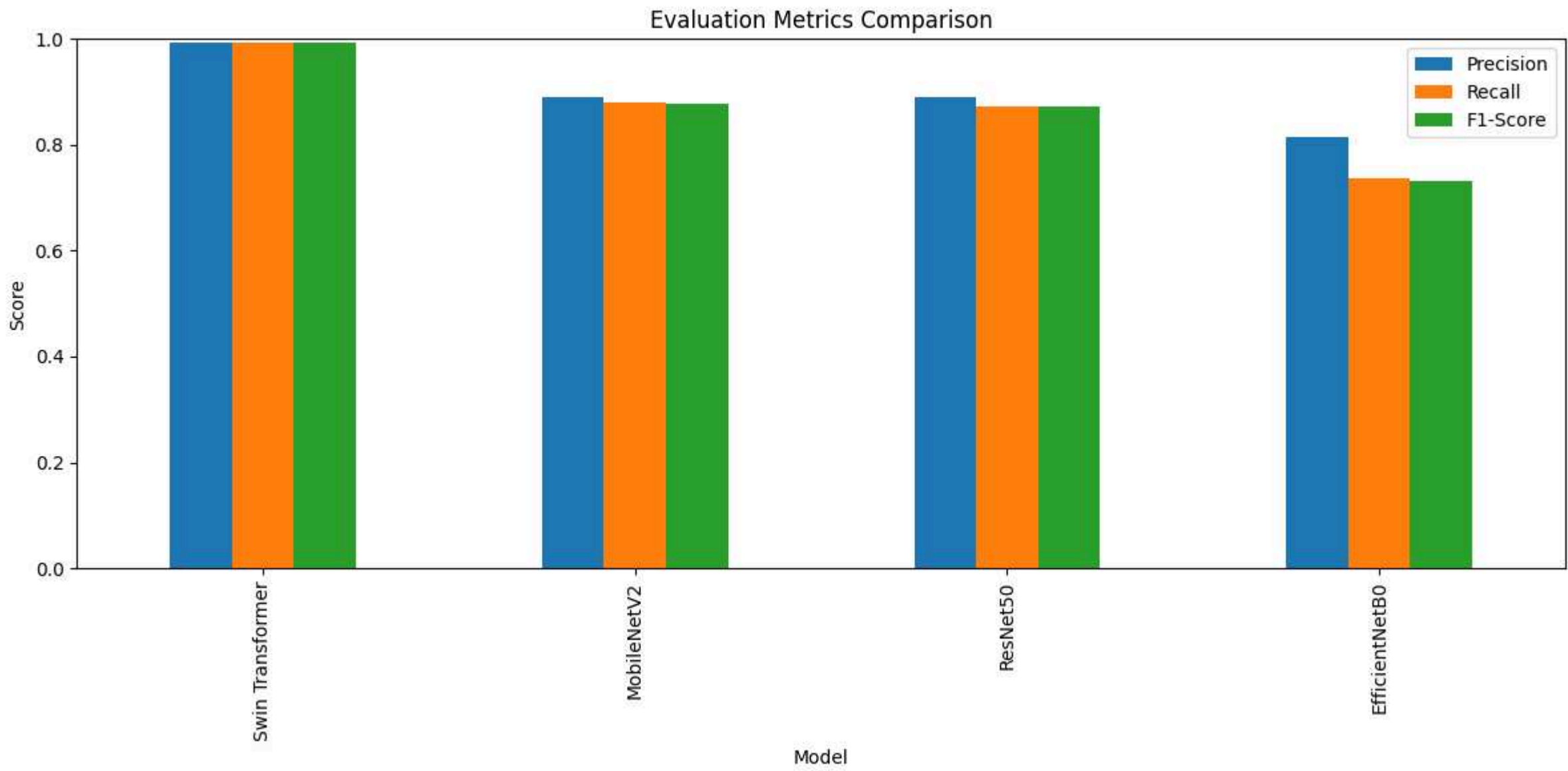
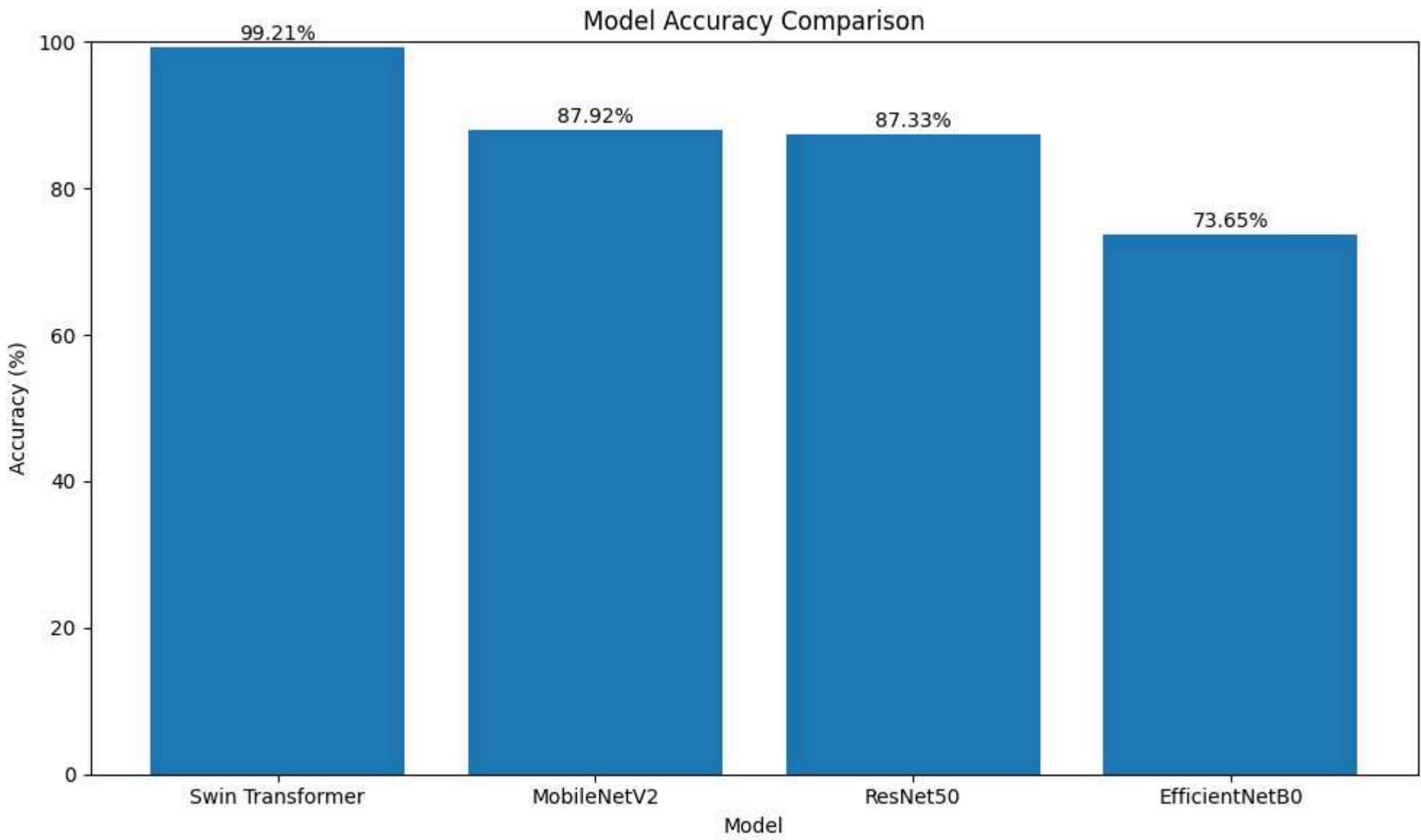
plt.title("Inference Time Comparison")
plt.xlabel("Model")
plt.ylabel("Time (Seconds)")

plt.tight_layout()

plt.savefig(
    "/content/drive/MyDrive/Inference_Time_Comparison.png",
    dpi=300,
    bbox_inches="tight"
)

plt.show()

print("All figures saved successfully.")
```



All figures saved successfully.

```
In [16]: # Step 16: Final Comparison Summary
# This step prints a final academic summary of the model comparison.

print("=" * 80)
print("ASL SIGN LANGUAGE RECOGNITION - FINAL MODEL COMPARISON")
print("=" * 80)

print("\nComparison Results:")
display(comparison_df)

best_model = comparison_df.iloc[0]

print("\n" + "=" * 80)
print("BEST PERFORMING MODEL")
print("=" * 80)

print(f"Model Name      : {best_model['Model']}")
print(f"Accuracy         : {best_model['Accuracy']*100:.2f}%")
print(f"Precision        : {best_model['Precision']:.4f}")
print(f"Recall           : {best_model['Recall']:.4f}")
print(f"F1-Score         : {best_model['F1-Score']:.4f}")
print(f"Inference Time   : {best_model['Inference Time (s)']:.2f} seconds")

print("\nConclusion:")
print(
    f"The experimental comparison demonstrates that the "
    f"{best_model['Model']} achieved the highest classification performance "
    f"among all evaluated deep learning models. "
    f"With a test accuracy of {best_model['Accuracy']*100:.2f}%, "
    f"it outperformed ResNet50, MobileNetV2, and EfficientNetB0, "
    f"making it the most suitable model for the proposed ASL recognition system."
)

print("=" * 80)
print("Comparison Completed Successfully.")
print("=" * 80)
```

=====
ASL SIGN LANGUAGE RECOGNITION - FINAL MODEL COMPARISON
=====

Comparison Results:

	Model	Test Images	Accuracy	Precision	Recall	F1-Score	Inference Time (s)
0	Swin Transformer	505	0.992079	0.992838	0.992079	0.992054	275.576531
1	MobileNetV2	505	0.879208	0.890759	0.879208	0.877554	143.833358
2	ResNet50	505	0.873267	0.888986	0.873267	0.873152	208.556154
3	EfficientNetB0	539	0.736549	0.815643	0.736549	0.731494	166.182902

=====
BEST PERFORMING MODEL
=====

Model Name : Swin Transformer
Accuracy : 99.21%
Precision : 0.9928
Recall : 0.9921
F1-Score : 0.9921
Inference Time : 275.58 seconds

Conclusion:
The experimental comparison demonstrates that the Swin Transformer achieved the highest classification performance among all evaluated deep learning models. With a test accuracy of 99.21%, it outperformed ResNet50, MobileNetV2, and EfficientNetB0, making it the most suitable model for the proposed ASL recognition system.

=====
Comparison Completed Successfully.
=====