

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the saved metrics
# and store the model comparison results.
# =====

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Import Required Libraries
# This step imports all libraries required to load, compare,
# visualize, and save the final model results.
# =====

import os
import json
import warnings

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

warnings.filterwarnings("ignore")

print("=" * 60)
print("Libraries Imported Successfully")
print("=" * 60)
```

=====
Libraries Imported Successfully
=====

```
In [3]: # =====
# Step 3: Define Comparison Paths
# This step defines the metrics input folder and the output
# folder used to save tables, figures, and summaries.
# =====

PROJECT_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project"
)

METRICS_DIR = os.path.join(
    PROJECT_DIR,
    "03_Results",
    "Metrics"
)

COMPARISON_DIR = os.path.join(
    PROJECT_DIR,
    "03_Results",
    "Comparison"
)

os.makedirs(
    COMPARISON_DIR,
    exist_ok=True
)

CSV_PATH = os.path.join(
    COMPARISON_DIR,
    "ASL_Models_Comparison.csv"
)

EXCEL_PATH = os.path.join(
    COMPARISON_DIR,
    "ASL_Models_Comparison.xlsx"
)

SUMMARY_PATH = os.path.join(
    COMPARISON_DIR,
    "ASL_Models_Comparison_Summary.txt"
)

print("=" * 60)
print("Comparison Paths Configured Successfully")
print("=" * 60)
print(f"Metrics Folder    : {METRICS_DIR}")
print(f"Comparison Folder  : {COMPARISON_DIR}")
print("=" * 60)
```

=====
Comparison Paths Configured Successfully
=====

Metrics Folder : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics
Comparison Folder : /content/drive/MyDrive/ASL_Final_Project/03_Results/Comparison
=====

```
In [4]: # =====
# Step 4: Define Metrics Files
# This step defines the six JSON files that contain the final
# evaluation results for all trained ASL models.
# =====

METRICS_FILES = [
    "resnet50_metrics.json",
    "mobilenetv2_metrics.json",
    "efficientnetb0_metrics.json",
    "vit_metrics.json",
    "swin_metrics.json",
    "hybrid_metrics.json"
]

missing_files = []

for file_name in METRICS_FILES:

    file_path = os.path.join(
        METRICS_DIR,
        file_name
    )

    if not os.path.isfile(file_path):
        missing_files.append(file_name)

print("=" * 60)
print("Metrics Files Verification")
print("=" * 60)

if missing_files:

    print("Status : INCOMPLETE")
    print("Missing Files:")

    for file_name in missing_files:
        print(f"- {file_name}")

    raise FileNotFoundError(
        "Some metrics files are missing. "
        "Complete Step 21 for all models first."
    )

else:

    print("Status : PASSED")
    print("All six metrics files were found.")

print("=" * 60)
```

=====
Metrics Files Verification
=====

Status : PASSED
All six metrics files were found.
=====

```
In [5]: # =====
# Step 5: Load All Model Metrics
# This step reads the six JSON files and stores their results
# for automatic model comparison.
# =====

all_metrics = []

for file_name in METRICS_FILES:

    file_path = os.path.join(
        METRICS_DIR,
        file_name
    )

    with open(
        file_path,
        "r"
    ) as metrics_file:

        metrics_data = json.load(
            metrics_file
        )

        all_metrics.append(
            metrics_data
        )

print("=" * 60)
print("All Model Metrics Loaded Successfully")
print("=" * 60)
print(f"Models Loaded : {len(all_metrics)}")

for metrics in all_metrics:
    print(f"- {metrics['model']}")

print("=" * 60)
```

```
=====
All Model Metrics Loaded Successfully
=====
Models Loaded : 6
- ResNet50
- MobileNetV2
- EfficientNetB0
- Vision Transformer
- Swin Transformer
- Hybrid MobileNetV2-Swin
=====
```

```
In [6]: # =====
# Step 6: Create the Comparison DataFrame
# This step organizes all model metrics into one structured
# table for ranking, visualization, and saving.
# =====

comparison_rows = []

for metrics in all_metrics:

    comparison_rows.append({
        "Model": metrics["model"],
        "Framework": metrics["framework"],
        "GPU": metrics["gpu"],
        "Test Images": metrics["test_images"],
        "Test Loss": metrics["test_loss"],
        "Accuracy": metrics["test_accuracy"],
        "Precision": metrics["weighted_precision"],
        "Recall": metrics["weighted_recall"],
        "F1-Score": metrics["weighted_f1_score"],
        "Macro F1-Score": metrics["macro_f1_score"],
        "Inference Time (s)": (
            metrics["average_inference_time_seconds"]
        ),
        "Inference Std (s)": (
            metrics[
                "inference_standard_deviation_seconds"
            ]
        ),
        "Time per Image (s)": (
            metrics["average_time_per_image_seconds"]
        ),
        "Parameters": metrics["parameters"],
        "Model Size (MB)": metrics["model_size_mb"],
        "Model Path": metrics["model_path"]
    })

comparison_df = pd.DataFrame(
    comparison_rows
)

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

comparison_df.insert(
    0,
    "Accuracy Rank",
    np.arange(
        1,
        len(comparison_df) + 1
    )
)

print("=" * 60)
print("ASL Models Comparison Table")
print("=" * 60)

display(
    comparison_df
)

print("=" * 60)
```

```
=====
ASL Models Comparison Table
=====
```

	Accuracy Rank	Model	Framework	GPU	Test Images	Test Loss	Accuracy	Precision	Recall	F1-Score	Macro F1-Score	Inference Time (s)	Inference Std (s)	Time per Image (s)	Parameters	Model Size (MB)	Model Path
0	1	Hybrid MobileNetV2-Swin	PyTorch	NVIDIA A100-SXM4-40GB	504	0.011220	0.998016	0.998148	0.998016	0.998013	0.998013	2.513058	1.121648	0.004986	30810782	117.856645	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
1	2	Swin Transformer	PyTorch	NVIDIA A100-SXM4-40GB	504	0.018935	0.996032	0.996032	0.996032	0.996032	0.996032	1.608323	0.031017	0.003191	27547038	105.150265	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
2	3	Vision Transformer	PyTorch	NVIDIA A100-SXM4-40GB	504	0.042720	0.988095	0.988319	0.988095	0.988080	0.988080	3.851580	3.124089	0.007642	85826340	327.459586	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
3	4	ResNet50	TensorFlow/Keras	NVIDIA A100-SXM4-40GB	504	0.377351	0.924603	0.933573	0.924603	0.924178	0.924178	2.289256	0.502939	0.004542	24121508	96.717209	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
4	5	MobileNetV2	TensorFlow/Keras	NVIDIA A100-SXM4-40GB	504	0.356522	0.918651	0.928646	0.918651	0.919262	0.919262	1.814369	0.081019	0.003600	2595172	13.033375	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
5	6	EfficientNetB0	TensorFlow/Keras	NVIDIA A100-SXM4-40GB	504	0.520006	0.898810	0.914187	0.898810	0.895122	0.895122	1.766245	0.105059	0.003504	4386759	20.113348	/content/drive/MyDrive/ASL_Final_Project/02_Mo...

```
=====
```

In [7]:

```
# =====
# Step 7: Verify Comparison Fairness
# This step verifies that all models were evaluated using the
# same number of test images and records the GPU information.
# =====

unique_test_counts = sorted(
    comparison_df["Test Images"].unique()
)

unique_gpus = sorted(
    comparison_df["GPU"].astype(str).unique()
)

print("=" * 60)
print("Comparison Fairness Verification")
print("=" * 60)
print(f"Test Image Counts : {unique_test_counts}")
print(f"GPU Devices      : {unique_gpus}")

if len(unique_test_counts) == 1:
    print("Test Set Status : CONSISTENT")
else:
    print("Test Set Status : WARNING - COUNTS DIFFER")

if len(unique_gpus) == 1:
    print("GPU Status      : CONSISTENT")
else:
    print("GPU Status      : WARNING - DEVICES DIFFER")

print("=" * 60)
```

```
=====
Comparison Fairness Verification
=====
Test Image Counts : [np.int64(504)]
GPU Devices      : ['NVIDIA A100-SXM4-40GB']
Test Set Status  : CONSISTENT
GPU Status       : CONSISTENT
=====
```

In [8]:

```
# =====
# Step 8: Save the Comparison Table
# This step saves the complete comparison results as CSV
# and Excel files.
# =====

comparison_df.to_csv(
    CSV_PATH,
    index=False
)

comparison_df.to_excel(
    EXCEL_PATH,
    index=False
)

print("=" * 60)
print("Comparison Table Saved Successfully")
print("=" * 60)
print(f"CSV File   : {CSV_PATH}")
print(f"Excel File : {EXCEL_PATH}")
print("=" * 60)
```

```
=====
Comparison Table Saved Successfully
=====
CSV File   : /content/drive/MyDrive/ASL_Final_Project/03_Results/Comparison/ASL_Models_Comparison.csv
Excel File : /content/drive/MyDrive/ASL_Final_Project/03_Results/Comparison/ASL_Models_Comparison.xlsx
=====
```

```
In [9]: # =====
# Step 9: Plot Accuracy Comparison
# This step compares the Test Accuracy of all evaluated models
# and saves the figure at 600 DPI.
# =====

ACCURACY_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Accuracy_Comparison.png"
)

accuracy_percent = (
    comparison_df["Accuracy"] * 100
)

plt.figure(
    figsize=(12, 7)
)

bars = plt.bar(
    comparison_df["Model"],
    accuracy_percent
)

plt.title(
    "ASL Model Accuracy Comparison",
    fontsize=16
)

plt.xlabel(
    "Model",
    fontsize=12
)

plt.ylabel(
    "Test Accuracy (%)",
    fontsize=12
)

plt.xticks(
    rotation=25,
    ha="right"
)

plt.ylim(
    0,
    105
)

plt.grid(
    axis="y",
    alpha=0.3
)

for bar, value in zip(
    bars,
    accuracy_percent
):

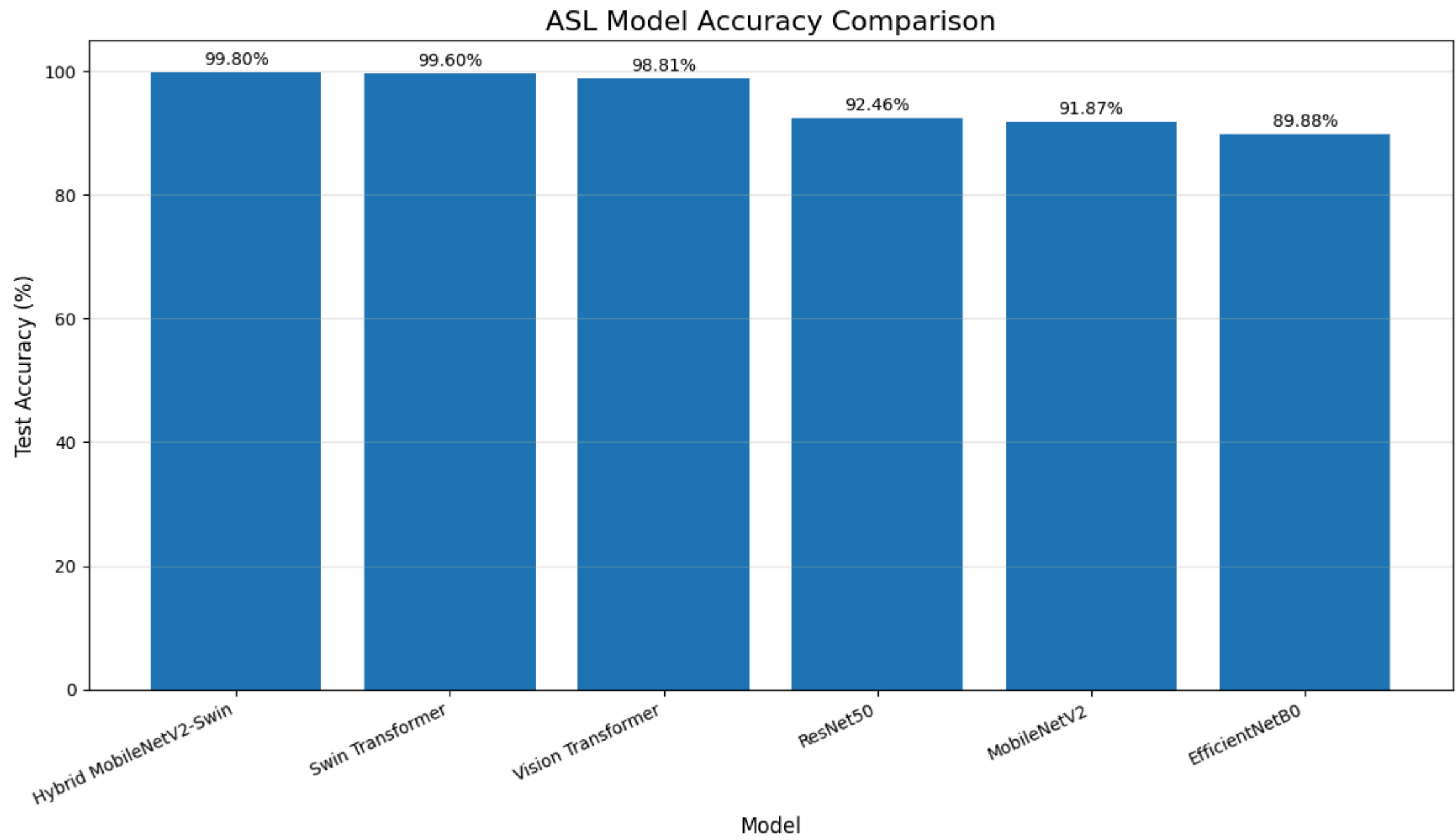
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.6,
        f"{value:.2f}%",
        ha="center",
        va="bottom",
        fontsize=10
    )

plt.tight_layout()

plt.savefig(
    ACCURACY_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Accuracy comparison figure saved successfully."
)
```



Accuracy comparison figure saved successfully.

```
In [10]: # =====
# Step 10: Plot Classification Metrics Comparison
# This step compares Precision, Recall, and F1-Score across
# all evaluated ASL models.
# =====

METRICS_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Evaluation_Metrics_Comparison.png"
)

positions = np.arange(
    len(comparison_df)
)

bar_width = 0.25

plt.figure(
    figsize=(14, 7)
)

plt.bar(
    positions - bar_width,
    comparison_df["Precision"],
    width=bar_width,
    label="Precision"
)

plt.bar(
    positions,
    comparison_df["Recall"],
    width=bar_width,
    label="Recall"
)

plt.bar(
    positions + bar_width,
    comparison_df["F1-Score"],
    width=bar_width,
    label="F1-Score"
)

plt.title(
    "Precision, Recall, and F1-Score Comparison",
    fontsize=16
)

plt.xlabel(
    "Model",
    fontsize=12
)

plt.ylabel(
    "Score",
    fontsize=12
)

plt.xticks(
    positions,
    comparison_df["Model"],
    rotation=25,
    ha="right"
)

plt.ylim(
    0,
    1.05
)

plt.legend()

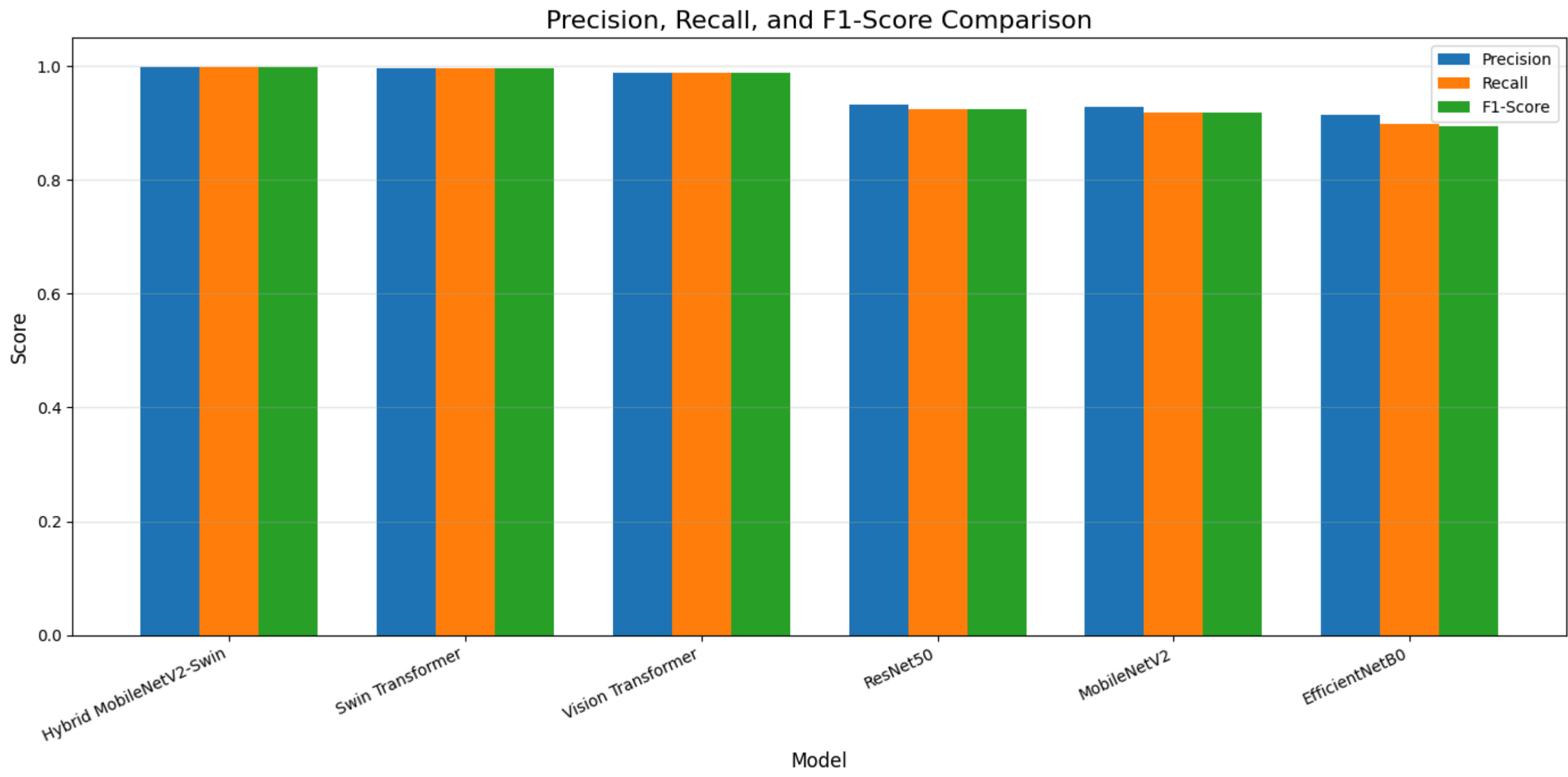
plt.grid(
    axis="y",
    alpha=0.3
)

plt.tight_layout()

plt.savefig(
    METRICS_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Evaluation metrics figure saved successfully."
)
```



Evaluation metrics figure saved successfully.

```
In [11]: # =====
# Step 11: Plot Inference Time Comparison
# This step compares the average total inference time required
# to process the complete test dataset.
# =====

INERENCE_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Inference_Time_Comparison.png"
)

inference_df = comparison_df.sort_values(
    by="Inference Time (s)",
    ascending=True
)

plt.figure(
    figsize=(12, 7)
)

bars = plt.bar(
    inference_df["Model"],
    inference_df["Inference Time (s)"]
)

plt.title(
    "Inference Time Comparison",
    fontsize=16
)

plt.xlabel(
    "Model",
    fontsize=12
)

plt.ylabel(
    "Average Total Inference Time (Seconds)",
    fontsize=12
)

plt.xticks(
    rotation=25,
    ha="right"
)

plt.grid(
    axis="y",
    alpha=0.3
)

for bar, value in zip(
    bars,
    inference_df["Inference Time (s)"]
):

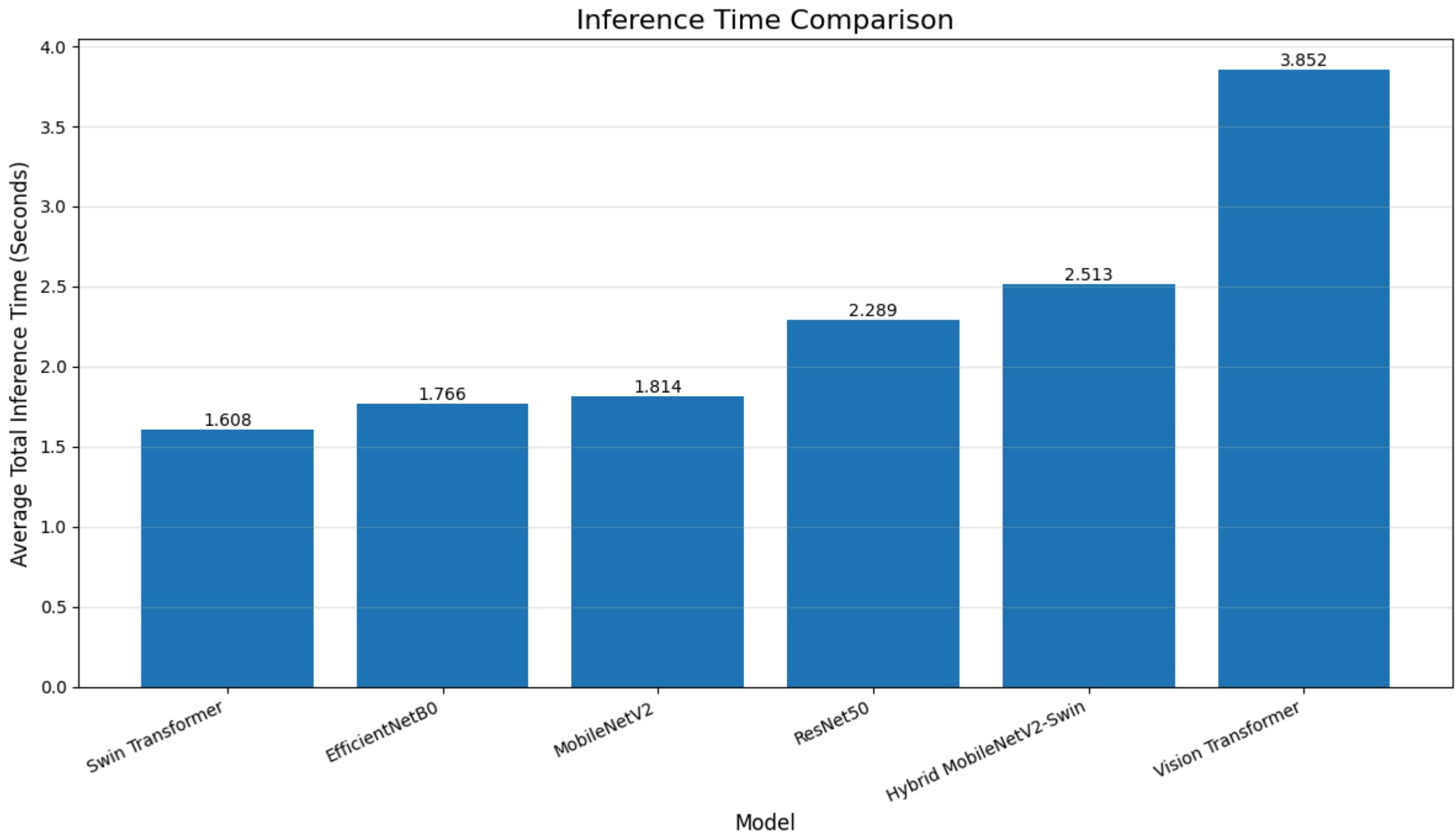
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.3f}",
        ha="center",
        va="bottom",
        fontsize=10
    )

plt.tight_layout()

plt.savefig(
    INERENCE_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Inference time figure saved successfully."
)
```



Inference time figure saved successfully.

```
In [12]: # =====
# Step 12: Plot Average Time per Image
# This step compares the average inference latency per image
# in milliseconds for all evaluated models.
# =====

LATENCY_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Time_Per_Image_Comparison.png"
)

latency_df = comparison_df.copy()

latency_df["Time per Image (ms)"] = (
    latency_df["Time per Image (s)"]
    * 1000
)

latency_df = latency_df.sort_values(
    by="Time per Image (ms)",
    ascending=True
)

plt.figure(
    figsize=(12, 7)
)

bars = plt.bar(
    latency_df["Model"],
    latency_df["Time per Image (ms)"]
)

plt.title(
    "Average Inference Time per Image",
    fontsize=16
)

plt.xlabel(
    "Model",
    fontsize=12
)

plt.ylabel(
    "Time per Image (Milliseconds)",
    fontsize=12
)

plt.xticks(
    rotation=25,
    ha="right"
)

plt.grid(
    axis="y",
    alpha=0.3
)

for bar, value in zip(
    bars,
    latency_df["Time per Image (ms)"]
):

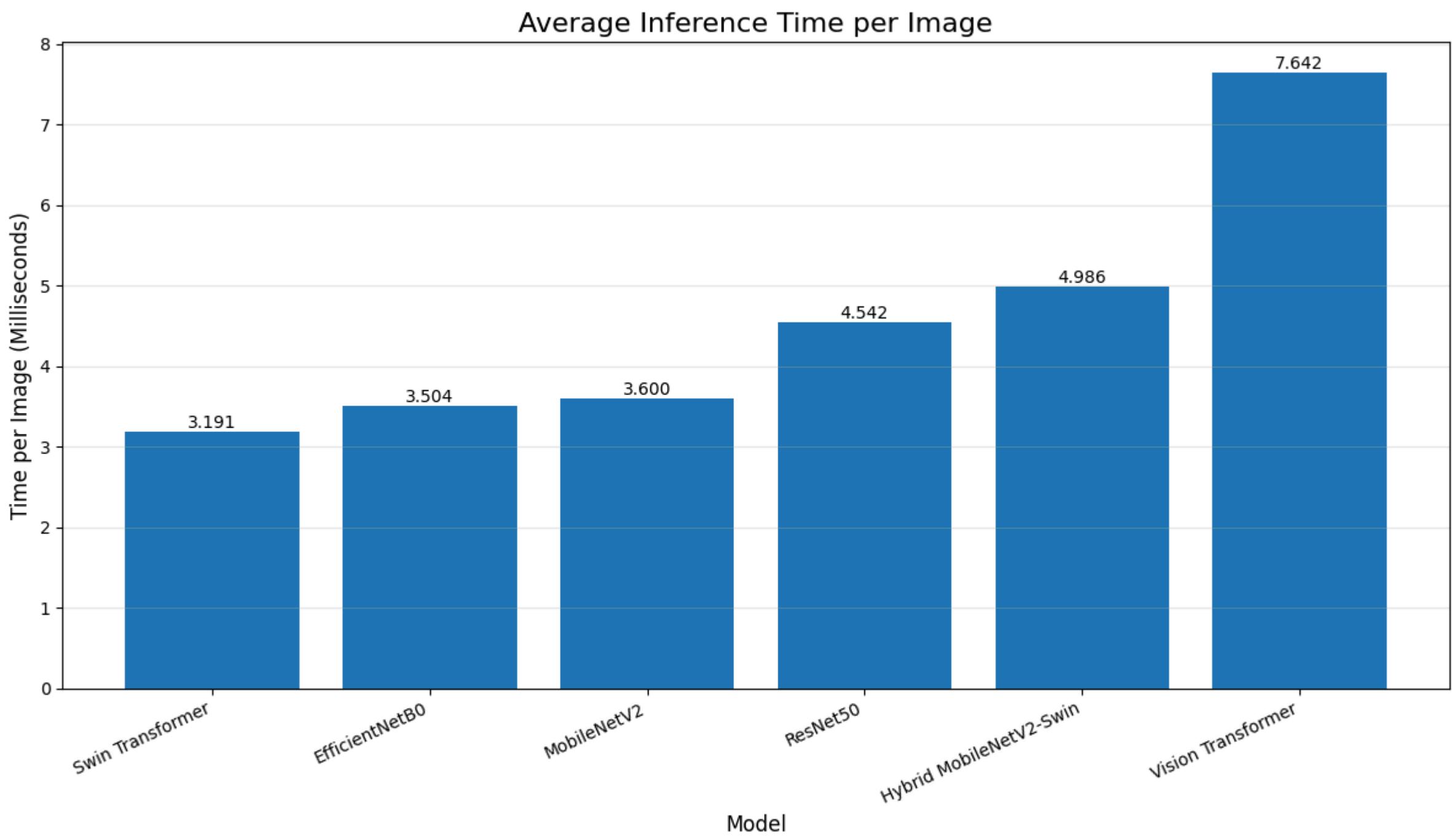
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.3f}",
        ha="center",
        va="bottom",
        fontsize=10
    )

plt.tight_layout()

plt.savefig(
    LATENCY_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Time-per-image figure saved successfully."
)
```



Time-per-image figure saved successfully.

```
In [13]: # =====
# Step 13: Plot Model Size Comparison
# This step compares the saved model file sizes to identify
# lightweight and storage-efficient models.
# =====

MODEL_SIZE_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Model_Size_Comparison.png"
)

size_df = comparison_df.sort_values(
    by="Model Size (MB)",
    ascending=True
)

plt.figure(
    figsize=(12, 7)
)

bars = plt.bar(
    size_df["Model"],
    size_df["Model Size (MB)"]
)

plt.title(
    "Model Size Comparison",
    fontsize=16
)

plt.xlabel(
    "Model",
    fontsize=12
)

plt.ylabel(
    "Model Size (MB)",
    fontsize=12
)

plt.xticks(
    rotation=25,
    ha="right"
)

plt.grid(
    axis="y",
    alpha=0.3
)

for bar, value in zip(
    bars,
    size_df["Model Size (MB)"]
):

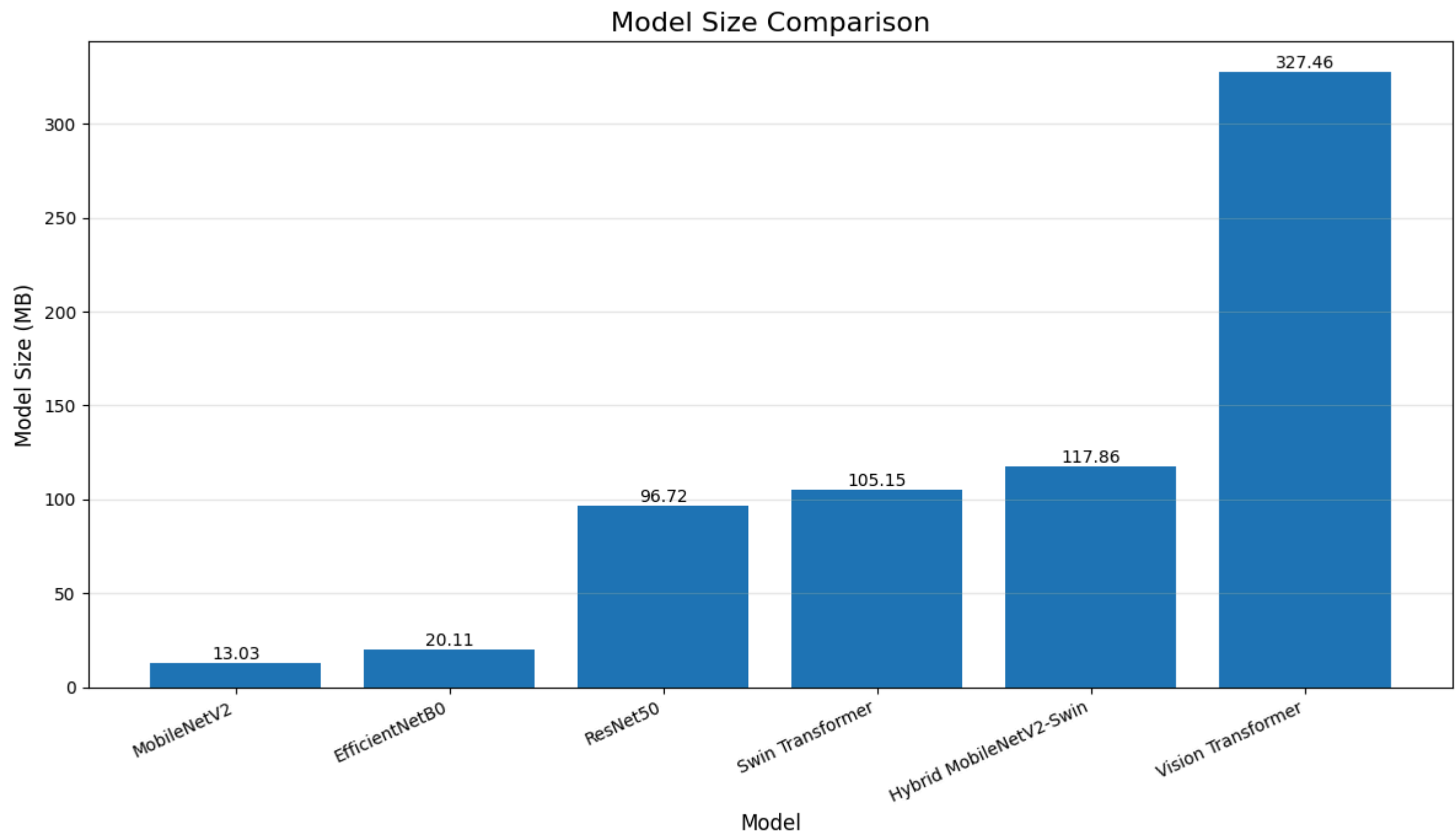
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.2f}",
        ha="center",
        va="bottom",
        fontsize=10
    )

plt.tight_layout()

plt.savefig(
    MODEL_SIZE_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Model size figure saved successfully."
)
```



Model size figure saved successfully.

```
In [14]: # =====
# Step 14: Plot Parameter Count Comparison
# This step compares the total number of trainable and
# non-trainable parameters in each saved model.
# =====

PARAMETERS_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Parameters_Comparison.png"
)

parameters_df = comparison_df.copy()

parameters_df["Parameters (Millions)"] = (
    parameters_df["Parameters"]
    / 1_000_000
)

parameters_df = parameters_df.sort_values(
    by="Parameters (Millions)",
    ascending=True
)

plt.figure(
    figsize=(12, 7)
)

bars = plt.bar(
    parameters_df["Model"],
    parameters_df["Parameters (Millions)"]
)

plt.title(
    "Model Parameter Count Comparison",
    fontsize=16
)

plt.xlabel(
    "Model",
    fontsize=12
)

plt.ylabel(
    "Parameters (Millions)",
    fontsize=12
)

plt.xticks(
    rotation=25,
    ha="right"
)

plt.grid(
    axis="y",
    alpha=0.3
)

for bar, value in zip(
    bars,
    parameters_df["Parameters (Millions)"]
):

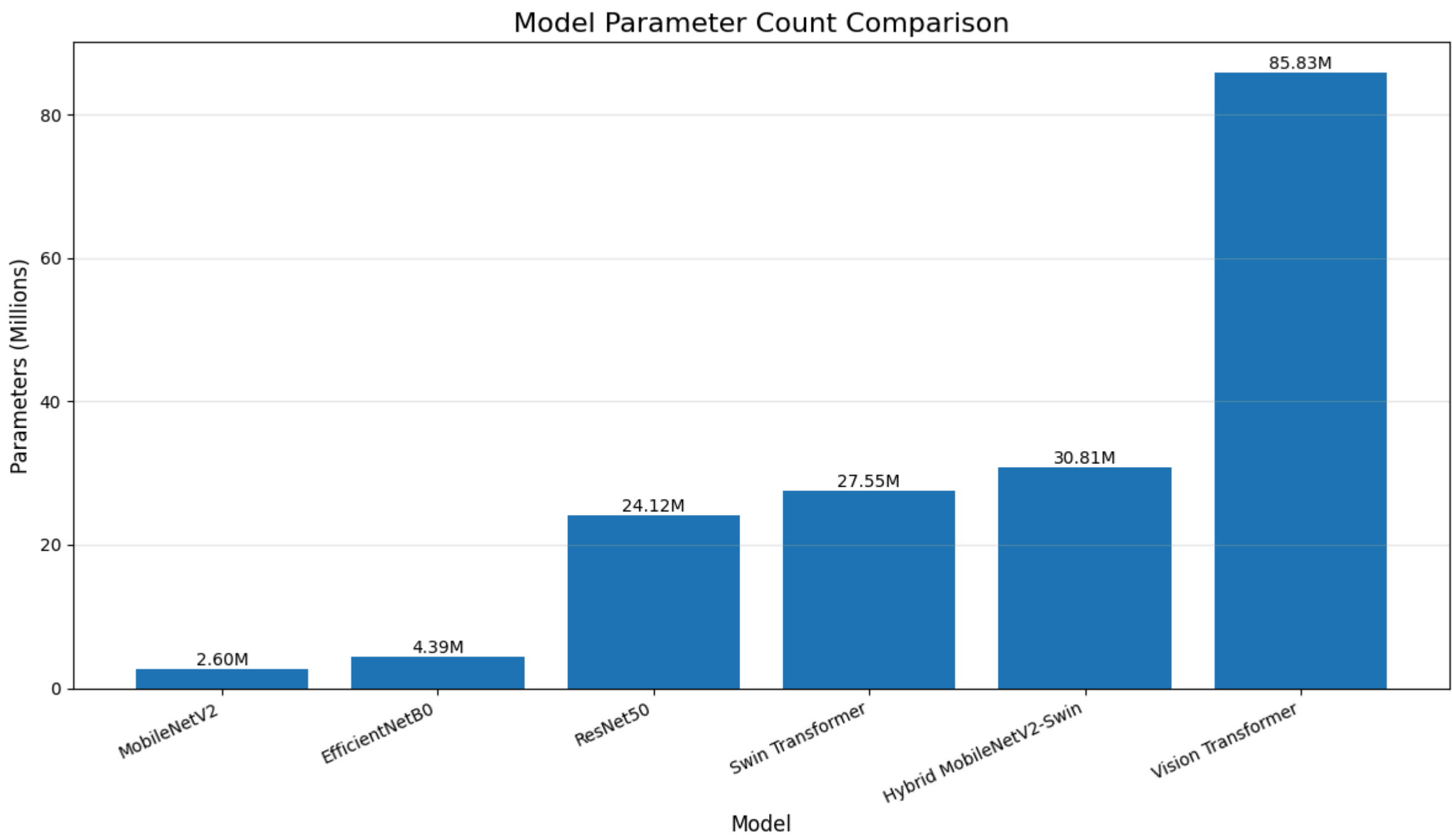
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.2f}M",
        ha="center",
        va="bottom",
        fontsize=10
    )

plt.tight_layout()

plt.savefig(
    PARAMETERS_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Parameter comparison figure saved successfully."
)
```



Parameter comparison figure saved successfully.

```
In [15]: # =====
# Step 15: Plot Accuracy-Speed Trade-off
# This step visualizes the relationship between model accuracy
# and average inference latency per image.
# =====

TRADEOFF_FIGURE_PATH = os.path.join(
    COMPARISON_DIR,
    "Accuracy_Speed_Tradeoff.png"
)

accuracy_values = (
    comparison_df["Accuracy"] * 100
)

latency_values = (
    comparison_df["Time per Image (s)"]
    * 1000
)

plt.figure(
    figsize=(11, 8)
)

plt.scatter(
    latency_values,
    accuracy_values,
    s=140
)

for index, model_name in enumerate(
    comparison_df["Model"]
):
    plt.annotate(
        model_name,
        (
            latency_values.iloc[index],
            accuracy_values.iloc[index]
        ),
        xytext=(7, 7),
        textcoords="offset points",
        fontsize=9
    )

plt.title(
    "Accuracy-Speed Trade-off",
    fontsize=16
)

plt.xlabel(
    "Average Inference Time per Image (ms)",
    fontsize=12
)

plt.ylabel(
    "Test Accuracy (%)",
    fontsize=12
)

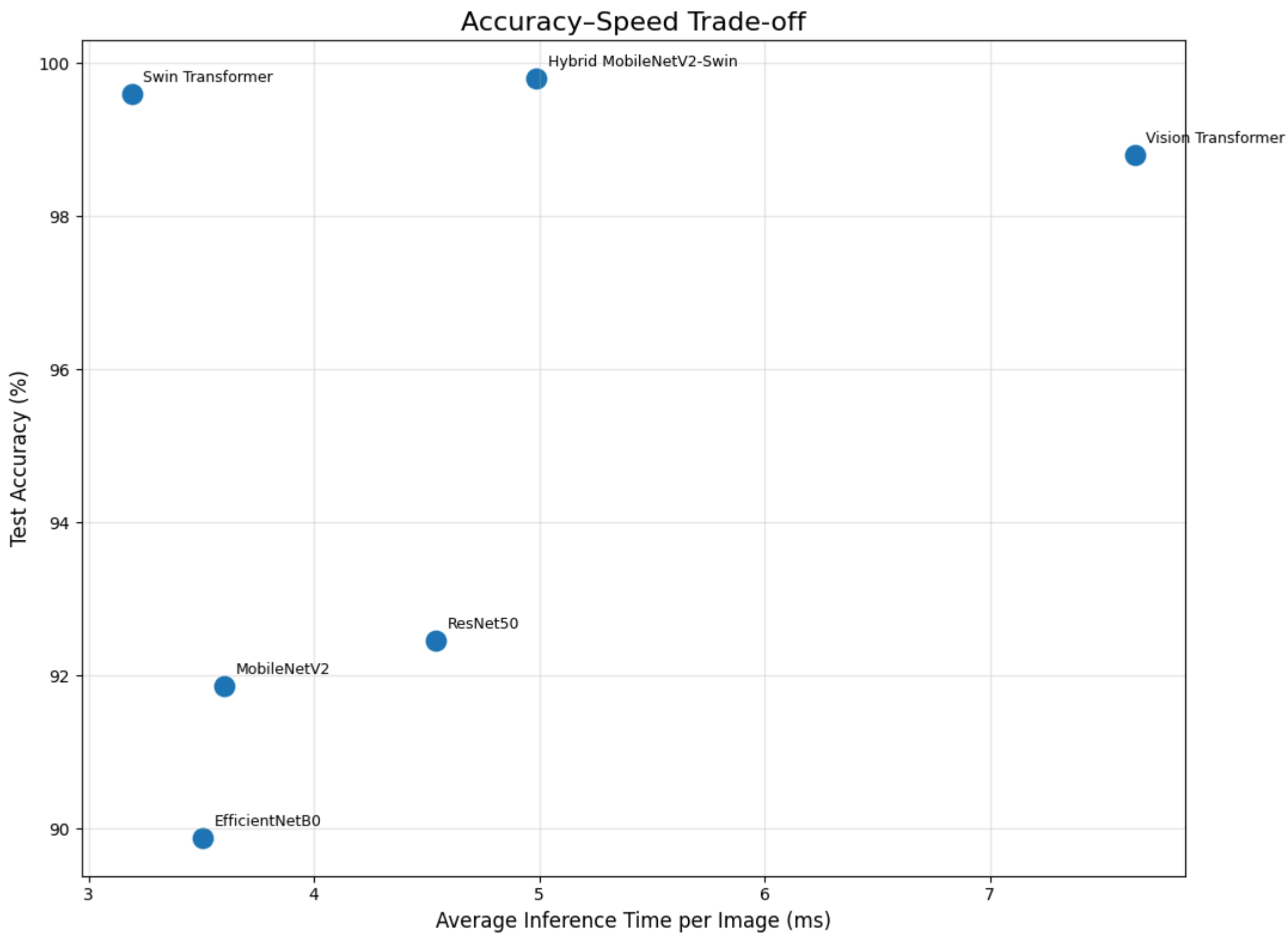
plt.grid(
    alpha=0.3
)

plt.tight_layout()

plt.savefig(
    TRADEOFF_FIGURE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print(
    "Accuracy-speed trade-off figure saved successfully."
)
```



Accuracy-speed trade-off figure saved successfully.

In [16]:

```
# =====
# Step 16: Display Final Comparison Summary
# This step identifies the highest-accuracy model, fastest
# model, smallest model, and highest F1-score model.
# =====

best_accuracy_model = comparison_df.loc[
    comparison_df["Accuracy"].idxmax()
]

best_f1_model = comparison_df.loc[
    comparison_df["F1-Score"].idxmax()
]

fastest_model = comparison_df.loc[
    comparison_df["Time per Image (s)"].idxmin()
]

smallest_model = comparison_df.loc[
    comparison_df["Model Size (MB)"].idxmin()
]

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

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

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

print(
    f"Highest Accuracy : "
    f"{best_accuracy_model['Model']} "
    f"({best_accuracy_model['Accuracy'] * 100:.2f}%"
)

print(
    f"Highest F1-Score : "
    f"{best_f1_model['Model']} "
    f"({best_f1_model['F1-Score']:.4f})"
)

print(
    f"Fastest Model : "
    f"{fastest_model['Model']} "
    f"({fastest_model['Time per Image (s)'] * 1000:.3f} ms/image)"
)

print(
    f"Smallest Model : "
    f"{smallest_model['Model']} "
    f"({smallest_model['Model Size (MB)']:.2f} MB)"
)

print("=" * 80)
```

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

Comparison Results:

	Accuracy Rank	Model	Framework	GPU	Test Images	Test Loss	Accuracy	Precision	Recall	F1-Score	Macro F1-Score	Inference Time (s)	Inference Std (s)	Time per Image (s)	Parameters	Model Size (MB)	Model Path
0	1	Hybrid MobileNetV2-Swin	PyTorch	NVIDIA A100-SXM4-40GB	504	0.011220	0.998016	0.998148	0.998016	0.998013	0.998013	2.513058	1.121648	0.004986	30810782	117.856645	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
1	2	Swin Transformer	PyTorch	NVIDIA A100-SXM4-40GB	504	0.018935	0.996032	0.996032	0.996032	0.996032	0.996032	1.608323	0.031017	0.003191	27547038	105.150265	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
2	3	Vision Transformer	PyTorch	NVIDIA A100-SXM4-40GB	504	0.042720	0.988095	0.988319	0.988095	0.988080	0.988080	3.851580	3.124089	0.007642	85826340	327.459586	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
3	4	ResNet50	TensorFlow/Keras	NVIDIA A100-SXM4-40GB	504	0.377351	0.924603	0.933573	0.924603	0.924178	0.924178	2.289256	0.502939	0.004542	24121508	96.717209	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
4	5	MobileNetV2	TensorFlow/Keras	NVIDIA A100-SXM4-40GB	504	0.356522	0.918651	0.928646	0.918651	0.919262	0.919262	1.814369	0.081019	0.003600	2595172	13.033375	/content/drive/MyDrive/ASL_Final_Project/02_Mo...
5	6	EfficientNetB0	TensorFlow/Keras	NVIDIA A100-SXM4-40GB	504	0.520006	0.898810	0.914187	0.898810	0.895122	0.895122	1.766245	0.105059	0.003504	4386759	20.113348	/content/drive/MyDrive/ASL_Final_Project/02_Mo...

=====
BEST MODEL SUMMARY
=====
Highest Accuracy : Hybrid MobileNetV2-Swin (99.80%)
Highest F1-Score : Hybrid MobileNetV2-Swin (0.9980)
Fastest Model : Swin Transformer (3.191 ms/image)
Smallest Model : MobileNetV2 (13.03 MB)
=====

In [17]:

```
# =====
# Step 17: Save the Academic Summary
# This step creates and saves an automatic academic conclusion
# based on the final model comparison results.
# =====

best_model_name = best_accuracy_model["Model"]
best_accuracy = best_accuracy_model["Accuracy"] * 100
best_precision = best_accuracy_model["Precision"]
best_recall = best_accuracy_model["Recall"]
best_f1 = best_accuracy_model["F1-Score"]
best_inference = best_accuracy_model[
    "Inference Time (s)"
]

summary_text = f"""
ASL SIGN LANGUAGE RECOGNITION - MODEL COMPARISON
=====

The experimental comparison evaluated six deep learning models:
ResNet50, MobileNetV2, EfficientNetB0, Vision Transformer,
Swin Transformer, and Hybrid MobileNetV2-Swin.

The highest classification accuracy was achieved by
{best_model_name}, with a test accuracy of {best_accuracy:.2f}%,
a weighted precision of {best_precision:.4f}, a weighted recall
of {best_recall:.4f}, and a weighted F1-score of {best_f1:.4f}.

The model required an average total inference time of
{best_inference:.4f} seconds for the complete test dataset.

The comparison also considered inference latency, model size,
and parameter count to provide a balanced assessment of
classification performance and computational efficiency.

Comparison Completed Successfully.
=====
""".strip()

print(summary_text)

with open(
    SUMMARY_PATH,
    "w"
) as summary_file:

    summary_file.write(
        summary_text
    )

print("\n" + "=" * 60)
print("Academic Summary Saved Successfully")
print("=" * 60)
print(f"Summary File : {SUMMARY_PATH}")
print("=" * 60)
```

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

The experimental comparison evaluated six deep learning models:
ResNet50, MobileNetV2, EfficientNetB0, Vision Transformer,
Swin Transformer, and Hybrid MobileNetV2-Swin.

The highest classification accuracy was achieved by
Hybrid MobileNetV2-Swin, with a test accuracy of 99.80%,
a weighted precision of 0.9981, a weighted recall
of 0.9980, and a weighted F1-score of 0.9980.

The model required an average total inference time of
2.5131 seconds for the complete test dataset.

The comparison also considered inference latency, model size,
and parameter count to provide a balanced assessment of
classification performance and computational efficiency.

Comparison Completed Successfully.
=====

Academic Summary Saved Successfully
=====
Summary File : /content/drive/MyDrive/ASL_Final_Project/03_Results/Comparison/ASL_Models_Comparison_Summary.txt
=====
```