

```
In [2]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

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

Mounted at /content/drive

```
In [3]: # =====
# Step 2: Install Required Libraries
# This step installs all required Python libraries used
# throughout the ASL recognition project.
# =====

!pip -q install split-folders
!pip -q install textblob
```

```
In [4]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data
# preprocessing, model training, evaluation, visualization,
# and result analysis.
# =====

import os
import random
import shutil
import time
import json
import warnings

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

from PIL import Image

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import (
    ResNet50,
    preprocess_input
)
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Dense,
    Dropout,
    GlobalAveragePooling2D
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import (
    EarlyStopping,
    ModelCheckpoint,
    ReduceLROnPlateau
)

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

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("TensorFlow Version :", tf.__version__)
print("=" * 60)

=====
TensorFlow Version : 2.20.0
=====
```

```
In [5]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies the TensorFlow version and
# GPU availability before model training.
# =====

SEED = 42

# Python random seed
random.seed(SEED)

# NumPy random seed
np.random.seed(SEED)

# TensorFlow random seed
tf.random.set_seed(SEED)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed      : {SEED}")
print(f"TensorFlow Version: {tf.__version__}")

gpus = tf.config.list_physical_devices('GPU')

if gpus:
    print(f"GPU Available    : Yes")
    print(f"GPU Device        : {gpus[0].name}")
else:
    print("GPU Available    : No")
    print("Please enable T4 GPU before training.")

print("=" * 60)
```

=====

Project Configuration

=====

Random Seed : 42

TensorFlow Version: 2.20.0

GPU Available : Yes

GPU Device : /physical_device:GPU:0

=====

```
In [6]: # =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive,
# verifies that it exists, and displays the available class
# folders before data preparation.
# =====

DATASET_DIR = "/content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset"

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):
    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")

    class_names = sorted(os.listdir(DATASET_DIR))

    print(f"Total Classes  : {len(class_names)}")
    print(f"Class Names    : {class_names}")

else:
    raise FileNotFoundError(
        f"\nDataset not found!\nPlease check the following path:\n{DATASET_DIR}"
    )

print("=" * 60)
```

=====

Dataset Configuration

=====

Dataset Status : Found

Dataset Path : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset

Total Classes : 36

Class Names : ['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 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_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'
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, folder_name))
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes     : {len(EXPECTED_CLASSES)}")
print(f"Available Classes    : {len(available_classes)}")
print(f"Missing Classes      : {missing_classes}")
print(f"Unexpected Classes   : {unexpected_classes}")
print("=" * 60)

if not dataset_ready:
    raise ValueError("Dataset class verification failed.")
```

```
=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes     : 36
Available Classes    : 36
Missing Classes      : []
Unexpected Classes   : []
=====
```

```
In [8]: # =====
# Step 7: Create Final Dataset Split
# This step creates one fixed Train, Validation, and Test split
# that will be used by all models for a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

required_split_folders = ["train", "val", "test"]

split_is_complete = all(
    os.path.isdir(os.path.join(SPLIT_DATASET_DIR, folder_name))
    for folder_name in required_split_folders
)

if not split_is_complete:

    # Remove the empty or incomplete split folder.
    if os.path.exists(SPLIT_DATASET_DIR):
        shutil.rmtree(SPLIT_DATASET_DIR)

    splitfolders.ratio(
        DATASET_DIR,
        output=SPLIT_DATASET_DIR,
        seed=SEED,
        ratio=(0.70, 0.10, 0.20),
        move=False
    )

    print("=" * 60)
    print("Dataset Split Created Successfully")
    print("=" * 60)

else:
    print("=" * 60)
    print("Complete Dataset Split Already Exists")
    print("=" * 60)

print(f"Split Dataset Path : {SPLIT_DATASET_DIR}")
print("=" * 60)
```

```
=====
Complete Dataset Split Already Exists
=====
Split Dataset Path : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final
=====
```

```
In [9]: # =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total
# number of images.
# =====

VALID_EXTENSIONS = (".jpg", ".jpeg", ".png", ".bmp", ".webp")

split_name_options = {
    "Training": ["train", "training"],
    "Validation": ["val", "valid", "validation"],
    "Testing": ["test", "testing"]
}

resolved_splits = {}

for display_name, possible_names in split_name_options.items():
    for folder_name in possible_names:
        folder_path = os.path.join(SPLIT_DATASET_DIR, folder_name)

        if os.path.isdir(folder_path):
            resolved_splits[display_name] = folder_path
            break

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

grand_total = 0

for display_name in ["Training", "Validation", "Testing"]:

    if display_name not in resolved_splits:
        raise FileNotFoundError(
            f"{display_name} folder was not found inside:\n"
            f"{SPLIT_DATASET_DIR}"
        )

    split_path = resolved_splits[display_name]
    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(split_path, class_name)

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(VALID_EXTENSIONS)
        )

    grand_total += total_images
    print(f"{display_name:<12}: {total_images} images")

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)

TRAIN_DIR = resolved_splits["Training"]
VALIDATION_DIR = resolved_splits["Validation"]
TEST_DIR = resolved_splits["Testing"]

print("Dataset split paths configured successfully.")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training   : 1760 images
Validation : 251 images
Testing    : 504 images
-----
Total Images : 2515
=====
Dataset split paths configured successfully.
=====
```

```
In [10]: # =====
# Step 9: Create Image Data Generators
# This step applies ResNet50-specific preprocessing and data
# augmentation to prepare the training, validation, and test data.
# =====

IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.15,
    shear_range=0.10,
    brightness_range=[0.7, 1.3],
    horizontal_flip=False,
    fill_mode="nearest"
)

validation_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
    seed=SEED
)

validation_generator = validation_test_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

test_generator = validation_test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("=" * 60)
print("Image Data Generators Created Successfully")
print("=" * 60)
print(f"Image Size      : {IMAGE_SIZE}")
print(f"Batch Size       : {BATCH_SIZE}")
print(f"Number of Classes : {train_generator.num_classes}")
print("=" * 60)
```

```
Found 1760 images belonging to 36 classes.
Found 251 images belonging to 36 classes.
Found 504 images belonging to 36 classes.
=====
Image Data Generators Created Successfully
=====
Image Size      : (224, 224)
Batch Size     : 16
Number of Classes : 36
=====
```

In [11]:

```
# =====
# Step 10: Display Dataset Information
# This step displays information about the dataset generators
# that will be used for model training and evaluation.
# =====

CLASS_NAMES = list(train_generator.class_indices.keys())
NUM_CLASSES = train_generator.num_classes

print("=" * 60)
print("Dataset Information")
print("=" * 60)
print(f"Training Samples : {train_generator.samples}")
print(f"Validation Samples : {validation_generator.samples}")
print(f"Testing Samples : {test_generator.samples}")
print(f"Number of Classes : {NUM_CLASSES}")
print(f"Batch Size : {BATCH_SIZE}")
print(f"Image Size : {IMAGE_SIZE}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Dataset Information
=====
Training Samples : 1760
Validation Samples : 251
Testing Samples : 504
Number of Classes : 36
Batch Size : 16
Image Size : (224, 224)
=====
Class Names
=====
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 [12]:

```
# =====
# Step 11: Build the ResNet50 Model
# This step loads the pretrained ResNet50 model and replaces
# its original classification head with a new 36-class head.
# =====

base_model = ResNet50(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)

base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.5)(x)

output = Dense(
    NUM_CLASSES,
    activation="softmax"
)(x)

model = Model(
    inputs=base_model.input,
    outputs=output,
    name="ResNet50_AS1_Model"
)

print("=" * 60)
print("ResNet50 Model Created Successfully")
print("=" * 60)
print(f"Base Model Trainable : {base_model.trainable}")
print(f"Number of Classes : {NUM_CLASSES}")
print("=" * 60)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
[1m94765736/94765736][0m [32m-----[0m[37m[0m [1m0s][0m 0us/step
=====
ResNet50 Model Created Successfully
=====
Base Model Trainable : False
Number of Classes : 36
=====
```

In [13]:

```
# =====
# Step 12: Compile the ResNet50 Model
# This step configures the model for training by defining the
# optimizer, loss function, and evaluation metric.
# =====

from tensorflow.keras.optimizers import Adam

LEARNING_RATE = 0.0001

model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

print("=" * 60)
print("Model Compilation Completed Successfully")
print("=" * 60)
print(f"Optimizer : Adam")
print(f"Learning Rate : {LEARNING_RATE}")
print(f"Loss Function : Categorical Crossentropy")
print(f"Evaluation : Accuracy")
print("=" * 60)
```

```
=====
Model Compilation Completed Successfully
=====
Optimizer : Adam
Learning Rate : 0.0001
Loss Function : Categorical Crossentropy
Evaluation : Accuracy
=====
```

```
In [14]: # =====
# Step 13: Configure Training Callbacks
# This step configures Early Stopping, Model Checkpoint,
# and Learning Rate Reduction during model training.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/ResNet50"
)

os.makedirs(MODEL_DIR, exist_ok=True)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_resnet50_asl_model.keras"
)

callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=7,
        restore_best_weights=True,
        verbose=1
    ),

    ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.3,
        patience=4,
        min_lr=1e-6,
        verbose=1
    ),

    ModelCheckpoint(
        filepath=BEST_MODEL_PATH,
        monitor="val_accuracy",
        save_best_only=True,
        verbose=1
    )
]

print("=" * 60)
print("Training Callbacks Configured Successfully")
print("=" * 60)
print("Early Stopping Patience : 7")
print("LR Reduction Patience   : 4")
print(f"Best Model Path           : {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Callbacks Configured Successfully
=====
Early Stopping Patience : 7
LR Reduction Patience   : 4
Best Model Path         : /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras
=====
```

In [15]:

```
# =====
# Step 14: Train the ResNet50 Model
# This step trains the ResNet50 model using the training
# and validation datasets.
# =====

EPOCHS = 30

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1
)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
```

Epoch 1/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 4s/step - accuracy: 0.0326 - loss: 3.8847

Epoch 1: val_accuracy improved from None to 0.21116, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 1: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m514s0[0m 5s/step - accuracy: 0.0648 - loss: 3.5808 - val_accuracy: 0.2112 - val_loss: 3.0573 - learning_rate: 1.0000e-04

Epoch 2/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 248ms/step - accuracy: 0.1640 - loss: 3.0855

Epoch 2: val_accuracy improved from 0.21116 to 0.33865, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 2: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m30s0[0m 266ms/step - accuracy: 0.1875 - loss: 3.0090 - val_accuracy: 0.3386 - val_loss: 2.6464 - learning_rate: 1.0000e-04

Epoch 3/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 251ms/step - accuracy: 0.2456 - loss: 2.7508

Epoch 3: val_accuracy improved from 0.33865 to 0.41036, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 3: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m30s0[0m 269ms/step - accuracy: 0.2597 - loss: 2.6611 - val_accuracy: 0.4104 - val_loss: 2.2692 - learning_rate: 1.0000e-04

Epoch 4/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 248ms/step - accuracy: 0.3332 - loss: 2.4159

Epoch 4: val_accuracy improved from 0.41036 to 0.55378, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 4: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m30s0[0m 267ms/step - accuracy: 0.3438 - loss: 2.3691 - val_accuracy: 0.5538 - val_loss: 1.9753 - learning_rate: 1.0000e-04

Epoch 5/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 247ms/step - accuracy: 0.3501 - loss: 2.2106

Epoch 5: val_accuracy improved from 0.55378 to 0.61355, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 5: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 264ms/step - accuracy: 0.3688 - loss: 2.1534 - val_accuracy: 0.6135 - val_loss: 1.7506 - learning_rate: 1.0000e-04

Epoch 6/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 247ms/step - accuracy: 0.4198 - loss: 2.0419

Epoch 6: val_accuracy improved from 0.61355 to 0.64143, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 6: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 265ms/step - accuracy: 0.4142 - loss: 2.0107 - val_accuracy: 0.6414 - val_loss: 1.5878 - learning_rate: 1.0000e-04

Epoch 7/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 246ms/step - accuracy: 0.4654 - loss: 1.8092

Epoch 7: val_accuracy improved from 0.64143 to 0.66932, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 7: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 264ms/step - accuracy: 0.4619 - loss: 1.8128 - val_accuracy: 0.6693 - val_loss: 1.4333 - learning_rate: 1.0000e-04

Epoch 8/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 246ms/step - accuracy: 0.4926 - loss: 1.7120

Epoch 8: val_accuracy improved from 0.66932 to 0.70916, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 8: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 265ms/step - accuracy: 0.5057 - loss: 1.6890 - val_accuracy: 0.7092 - val_loss: 1.2902 - learning_rate: 1.0000e-04

Epoch 9/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 246ms/step - accuracy: 0.5192 - loss: 1.5962

Epoch 9: val_accuracy improved from 0.70916 to 0.74104, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 9: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 263ms/step - accuracy: 0.5239 - loss: 1.5786 - val_accuracy: 0.7410 - val_loss: 1.1791 - learning_rate: 1.0000e-04

Epoch 10/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 248ms/step - accuracy: 0.5761 - loss: 1.4703

Epoch 10: val_accuracy did not improve from 0.74104

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 258ms/step - accuracy: 0.5739 - loss: 1.4612 - val_accuracy: 0.7251 - val_loss: 1.1070 - learning_rate: 1.0000e-04

Epoch 11/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 239ms/step - accuracy: 0.5657 - loss: 1.4446

Epoch 11: val_accuracy improved from 0.74104 to 0.75697, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 11: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 258ms/step - accuracy: 0.5778 - loss: 1.4097 - val_accuracy: 0.7570 - val_loss: 1.0351 - learning_rate: 1.0000e-04

Epoch 12/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 244ms/step - accuracy: 0.6119 - loss: 1.3511

Epoch 12: val_accuracy improved from 0.75697 to 0.76494, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 12: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 263ms/step - accuracy: 0.6119 - loss: 1.3429 - val_accuracy: 0.7649 - val_loss: 0.9956 - learning_rate: 1.0000e-04

Epoch 13/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 246ms/step - accuracy: 0.6188 - loss: 1.2831

Epoch 13: val_accuracy improved from 0.76494 to 0.77291, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 13: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 264ms/step - accuracy: 0.6119 - loss: 1.2722 - val_accuracy: 0.7729 - val_loss: 0.9038 - learning_rate: 1.0000e-04

Epoch 14/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 248ms/step - accuracy: 0.6745 - loss: 1.1173

Epoch 14: val_accuracy improved from 0.77291 to 0.82869, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 14: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 266ms/step - accuracy: 0.6659 - loss: 1.1380 - val_accuracy: 0.8287 - val_loss: 0.8158 - learning_rate: 1.0000e-04

Epoch 15/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 248ms/step - accuracy: 0.6678 - loss: 1.1355

Epoch 15: val_accuracy did not improve from 0.82869

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m29s0[0m 259ms/step - accuracy: 0.6568 - loss: 1.1416 - val_accuracy: 0.8008 - val_loss: 0.8160 - learning_rate: 1.0000e-04

Epoch 16/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 236ms/step - accuracy: 0.6838 - loss: 1.1169

Epoch 16: val_accuracy did not improve from 0.82869

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 247ms/step - accuracy: 0.6778 - loss: 1.1045 - val_accuracy: 0.8207 - val_loss: 0.7761 - learning_rate: 1.0000e-04

Epoch 17/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 239ms/step - accuracy: 0.6692 - loss: 1.0906

Epoch 17: val_accuracy did not improve from 0.82869

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 249ms/step - accuracy: 0.6926 - loss: 1.0388 - val_accuracy: 0.8167 - val_loss: 0.7131 - learning_rate: 1.0000e-04

Epoch 18/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 236ms/step - accuracy: 0.7033 - loss: 0.9865

Epoch 18: val_accuracy did not improve from 0.82869

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 246ms/step - accuracy: 0.7080 - loss: 0.9665 - val_accuracy: 0.7928 - val_loss: 0.7110 - learning_rate: 1.0000e-04

Epoch 19/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 237ms/step - accuracy: 0.7131 - loss: 0.9364

Epoch 19: val_accuracy improved from 0.82869 to 0.83665, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 19: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 255ms/step - accuracy: 0.7136 - loss: 0.9298 - val_accuracy: 0.8367 - val_loss: 0.6502 - learning_rate: 1.0000e-04

Epoch 20/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 245ms/step - accuracy: 0.7334 - loss: 0.8727

Epoch 20: val_accuracy did not improve from 0.83665

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 254ms/step - accuracy: 0.7335 - loss: 0.8978 - val_accuracy: 0.8048 - val_loss: 0.6557 - learning_rate: 1.0000e-04

Epoch 21/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 238ms/step - accuracy: 0.7245 - loss: 0.9005

Epoch 21: val_accuracy did not improve from 0.83665

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 248ms/step - accuracy: 0.7426 - loss: 0.8782 - val_accuracy: 0.8287 - val_loss: 0.6207 - learning_rate: 1.0000e-04

Epoch 22/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 237ms/step - accuracy: 0.7344 - loss: 0.8607

Epoch 22: val_accuracy did not improve from 0.83665

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 246ms/step - accuracy: 0.7278 - loss: 0.8763 - val_accuracy: 0.8127 - val_loss: 0.6339 - learning_rate: 1.0000e-04

Epoch 23/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 237ms/step - accuracy: 0.7481 - loss: 0.8414

Epoch 23: val_accuracy improved from 0.83665 to 0.87649, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

Epoch 23: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 254ms/step - accuracy: 0.7432 - loss: 0.8413 - val_accuracy: 0.8765 - val_loss: 0.5553 - learning_rate: 1.0000e-04

Epoch 24/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 244ms/step - accuracy: 0.7386 - loss: 0.8384

Epoch 24: val_accuracy did not improve from 0.87649

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 254ms/step - accuracy: 0.7455 - loss: 0.8192 - val_accuracy: 0.8526 - val_loss: 0.5482 - learning_rate: 1.0000e-04

Epoch 25/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 239ms/step - accuracy: 0.7734 - loss: 0.7841

Epoch 25: val_accuracy did not improve from 0.87649

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 249ms/step - accuracy: 0.7767 - loss: 0.7648 - val_accuracy: 0.8685 - val_loss: 0.5270 - learning_rate: 1.0000e-04

Epoch 26/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 240ms/step - accuracy: 0.7782 - loss: 0.7496

Epoch 26: val_accuracy did not improve from 0.87649

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m28s0[0m 250ms/step - accuracy: 0.7699 - loss: 0.7681 - val_accuracy: 0.8566 - val_loss: 0.5251 - learning_rate: 1.0000e-04

Epoch 27/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 235ms/step - accuracy: 0.7828 - loss: 0.7172

Epoch 27: val_accuracy did not improve from 0.87649

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 245ms/step - accuracy: 0.7835 - loss: 0.7135 - val_accuracy: 0.8446 - val_loss: 0.5099 - learning_rate: 1.0000e-04

Epoch 28/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 238ms/step - accuracy: 0.7871 - loss: 0.6822

Epoch 28: val_accuracy did not improve from 0.87649

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 249ms/step - accuracy: 0.7744 - loss: 0.7015 - val_accuracy: 0.8606 - val_loss: 0.4867 - learning_rate: 1.0000e-04

Epoch 29/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 237ms/step - accuracy: 0.7473 - loss: 0.7839

Epoch 29: val_accuracy did not improve from 0.87649

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m27s0[0m 248ms/step - accuracy: 0.7841 - loss: 0.7069 - val_accuracy: 0.8685 - val_loss: 0.4820 - learning_rate: 1.0000e-04

Epoch 30/30

0[1m10/1100[0m @[32m-----0[0m0[37m0[0m @[1m0s0[0m 238ms/step - accuracy: 0.7884 - loss: 0.6595

Epoch 30: val_accuracy improved from 0.87649 to 0.88446, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras

```
Epoch 30: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/best_resnet50_asl_model.keras
[1m10/110][0m  [32m-----[0m[37m[0m [1m28s[0m 258ms/step - accuracy: 0.7847 - loss: 0.6802 - val_accuracy: 0.8845 - val_loss: 0.4523 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.
=====
Model Training Completed Successfully
=====
```

```
In [16]: # =====
# Step 15: Evaluate the ResNet50 Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

test_loss, test_accuracy = model.evaluate(
    test_generator,
    verbose=1
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
[1m32/32][0m  [32m-----[0m[37m[0m [1m16s[0m 4s/step - accuracy: 0.9246 - loss: 0.3774
=====
Model Evaluation
=====
Test Loss      : 0.3774
Test Accuracy : 0.9246
=====
```

```
In [17]: # =====
# Step 16: Generate Predictions and Classification Report
# This step generates predictions on the testing dataset and
# displays the classification report.
# =====

from sklearn.metrics import classification_report
import numpy as np

# Reset the test generator
test_generator.reset()

# Generate predictions
predictions = model.predict(test_generator, verbose=1)

# Predicted and true labels
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

# Classification Report
report = classification_report(
    y_true,
    y_pred,
    target_names=list(test_generator.class_indices.keys()),
    digits=4
)

print("=" * 60)
print("Classification Report")
print("=" * 60)
print(report)
print("=" * 60)
```

```
[1m32/32][0m  [32m-----[0m[37m[0m [1m10s[0m 192ms/step
=====
Classification Report
=====
              precision    recall  f1-score   support

     0   0.9000      0.6429      0.7500        14
     1   1.0000      0.8571      0.9231        14
     2   0.7857      0.7857      0.7857        14
     3   1.0000      1.0000      1.0000        14
     4   0.8000      0.8571      0.8276        14
     5   0.8571      0.8571      0.8571        14
     6   0.6875      0.7857      0.7333        14
     7   1.0000      1.0000      1.0000        14
     8   1.0000      1.0000      1.0000        14
     9   0.9286      0.9286      0.9286        14
    a   1.0000      1.0000      1.0000        14
    b   1.0000      1.0000      1.0000        14
    c   1.0000      1.0000      1.0000        14
    d   1.0000      1.0000      1.0000        14
    e   0.9333      1.0000      0.9655        14
    f   1.0000      1.0000      1.0000        14
    g   1.0000      1.0000      1.0000        14
    h   1.0000      1.0000      1.0000        14
    i   1.0000      1.0000      1.0000        14
    j   1.0000      1.0000      1.0000        14
    k   1.0000      0.9286      0.9630        14
    l   1.0000      1.0000      1.0000        14
    m   0.6364      1.0000      0.7778        14
    n   1.0000      0.5714      0.7273        14
    o   0.7222      0.9286      0.8125        14
    p   1.0000      1.0000      1.0000        14
    q   1.0000      1.0000      1.0000        14
    r   0.8571      0.8571      0.8571        14
    s   1.0000      0.7143      0.8333        14
    t   0.9333      1.0000      0.9655        14
    u   0.8750      1.0000      0.9333        14
    v   0.8462      0.7857      0.8148        14
    w   0.8462      0.7857      0.8148        14
    x   1.0000      1.0000      1.0000        14
    y   1.0000      1.0000      1.0000        14
    z   1.0000      1.0000      1.0000        14

 accuracy          0.9246         504
 macro avg         0.9336      0.9246      0.9242         504
weighted avg         0.9336      0.9246      0.9242         504

=====
```

In [18]:

```
# =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "resnet50_confusion_matrix.png"
)

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(16, 14))

plt.imshow(cm, cmap="Blues")

plt.title(
    "ResNet50 Confusion Matrix",
    fontsize=18
)

plt.xlabel(
    "Predicted Label",
    fontsize=14
)

plt.ylabel(
    "True Label",
    fontsize=14
)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):

        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

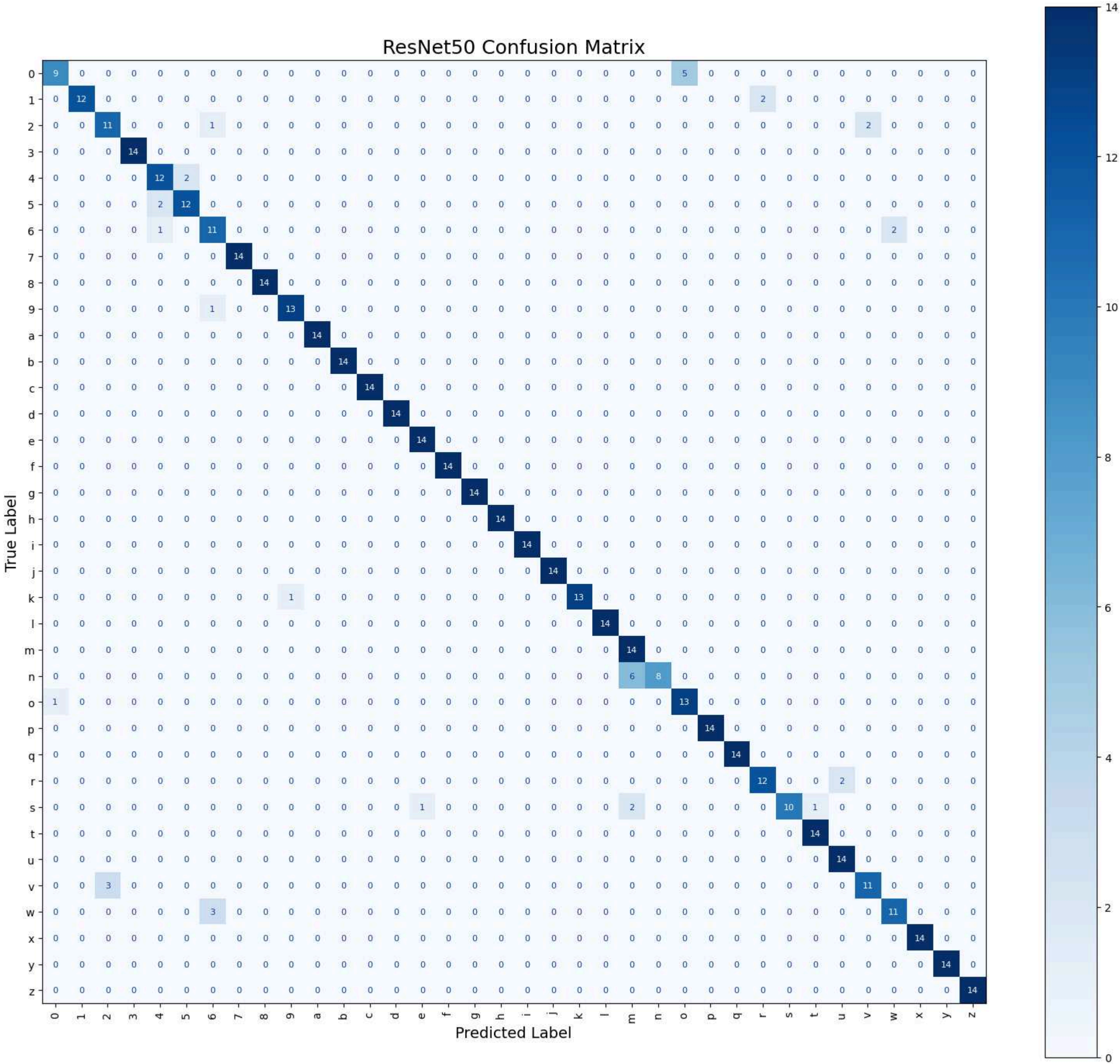
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



```
=====
Confusion Matrix Saved Successfully
=====
Saved Path : /content/drive/MyDrive/ASL_Final_Project/03_Results/Confusion_Matrices/resnet50_confusion_matrix.png
=====
```

In [19]:

```
# =====
# Step 18: Plot Training History
# This step plots the training and validation accuracy and
# loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "resnet50_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "resnet50_loss_curve.png"
)

# Accuracy Curve
plt.figure(figsize=(8, 6))

plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")

plt.title("ResNet50 Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

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

plt.show()

# Loss Curve
plt.figure(figsize=(8, 6))

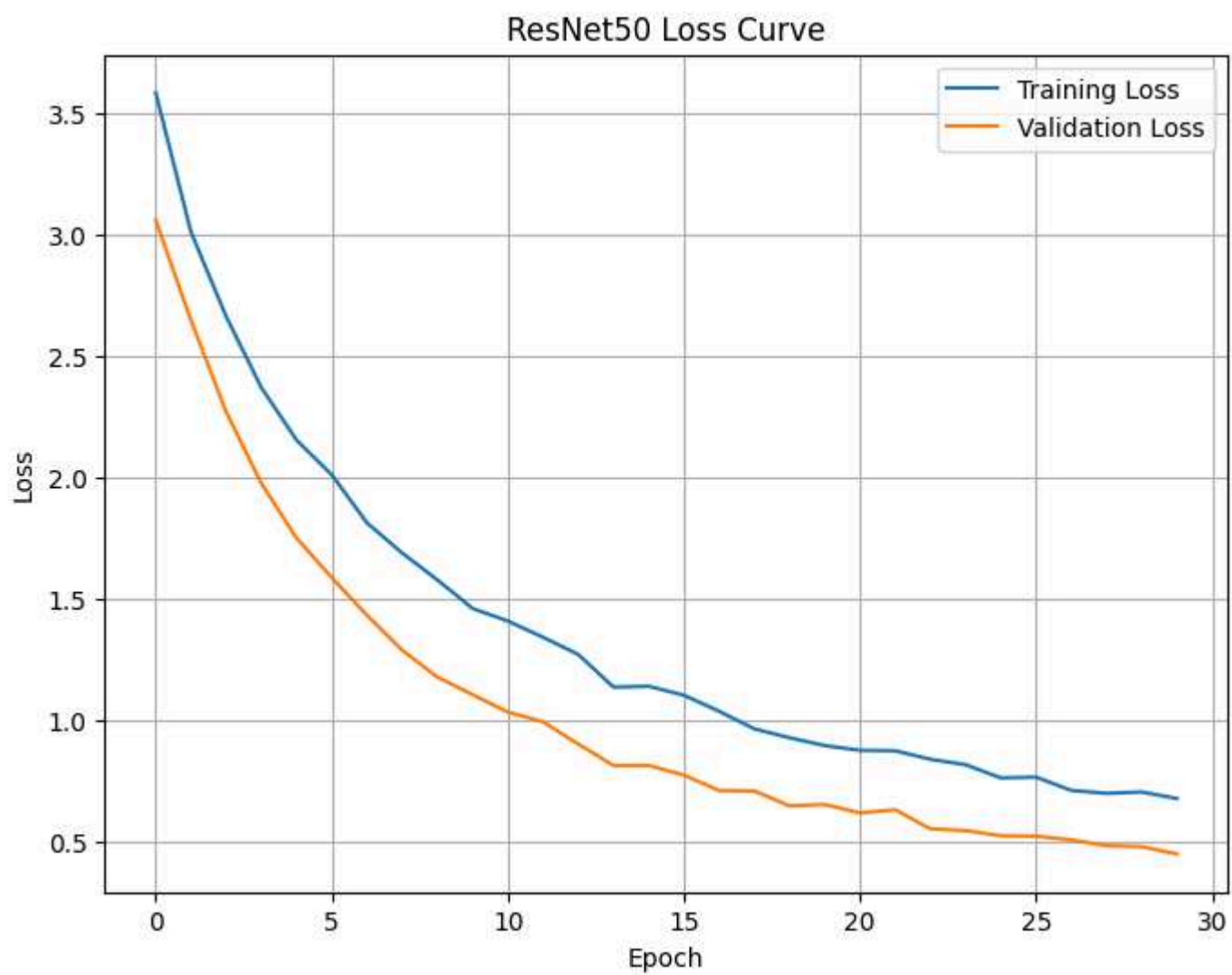
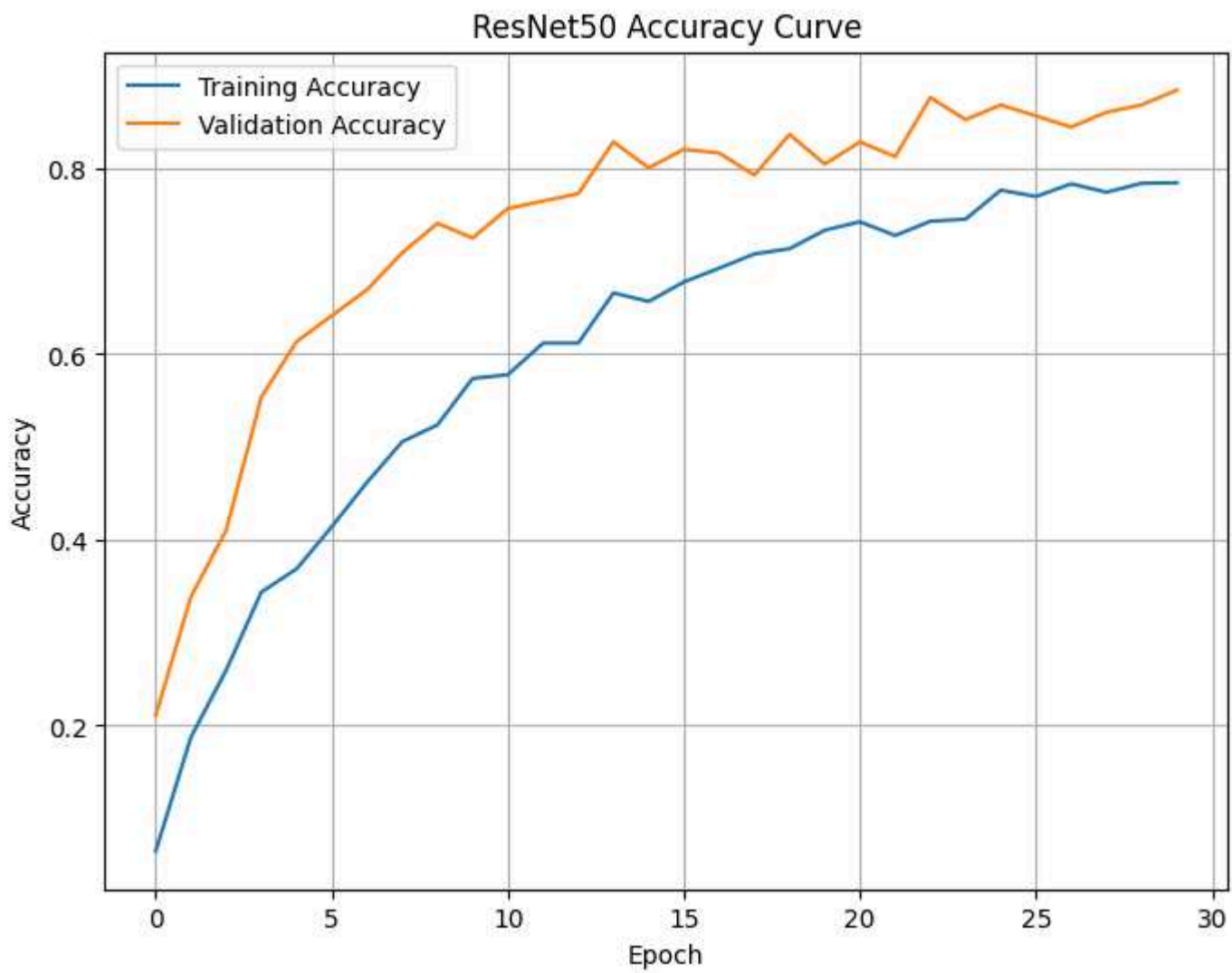
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")

plt.title("ResNet50 Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

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

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/resnet50_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/resnet50_loss_curve.png
=====
```

In [20]:

```
# =====
# Step 19: Save the Final ResNet50 Model
# This step saves the trained ResNet50 model and the class
# names for future inference and deployment.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/ResNet50"
)

os.makedirs(MODEL_DIR, exist_ok=True)

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_resnet50_asl_model.keras"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "asl_class_names.txt"
)

# Save the trained model
model.save(FINAL_MODEL_PATH)

# Save class names
with open(CLASS_NAMES_PATH, "w") as file:
    for class_name in CLASS_NAMES:
        file.write(f"{class_name}\n")

print("=" * 60)
print("Model Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print("=" * 60)

=====
Model Saved Successfully
=====

Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/final_resnet50_asl_model.keras
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/ResNet50/asl_class_names.txt
=====
```

In [21]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures the model inference time on the complete
# test dataset using a warm-up run and repeated measurements.
# =====

INFERENCE_REPEATS = 3

# Warm-up run
test_generator.reset()
_ = model.predict(
    test_generator,
    verbose=0
)

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    test_generator.reset()

    start_time = time.perf_counter()

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

    end_time = time.perf_counter()

    inference_times.append(end_time - start_time)

average_inference_time = np.mean(inference_times)
std_inference_time = np.std(inference_times)

average_time_per_image = (
    average_inference_time / test_generator.samples
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print(f"Test Images          : {test_generator.samples}")
print(f"Repeated Measurements  : {INFERENCE_REPEATS}")
print(f"Average Total Time     : {average_inference_time:.4f} seconds")
print(f"Standard Deviation      : {std_inference_time:.4f} seconds")
print(f"Average Time per Image  : {average_time_per_image:.6f} seconds")
print("=" * 60)

=====
Standardized Inference Time
=====
Test Images          : 504
Repeated Measurements : 3
Average Total Time   : 2.2893 seconds
Standard Deviation    : 0.5029 seconds
Average Time per Image : 0.004542 seconds
=====
```

```
In [22]: # =====
# Step 21: Save Final ResNet50 Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "resnet50"
MODEL_DISPLAY_NAME = "ResNet50"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/ResNet50/final_resnet50_asl_model.keras"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "resnet50_metrics.json"
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpus = tf.config.list_physical_devices("GPU")

if gpus:
    gpu_name = (
        tf.config.experimental
        .get_device_details(gpus[0])
        .get("device_name", gpus[0].name)
    )
else:
    gpu_name = "CPU"

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "TensorFlow/Keras",
    "gpu": gpu_name,
    "test_images": int(test_generator.samples),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(model.count_params()),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("ResNet50 Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size     : {model_size_mb:.2f} MB")
print(f"Parameters     : {model.count_params():,}")
print(f"Metrics File   : {METRICS_PATH}")
print("=" * 60)
```

```
=====
ResNet50 Metrics Saved Successfully
=====
Accuracy      : 0.9246
Precision     : 0.9336
Recall        : 0.9246
F1-Score      : 0.9242
Model Size    : 96.72 MB
Parameters    : 24,121,508
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/resnet50_metrics.json
=====
```

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs all required Python libraries used
# throughout the ASL recognition project.
# =====

!pip -q install split-folders
!pip -q install textblob
```

```
In [3]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data
# preprocessing, model training, evaluation, visualization,
# and result analysis.
# =====

import os
import random
import shutil
import time
import json
import warnings

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

from PIL import Image

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

from tensorflow.keras.models import Model

from tensorflow.keras.layers import (
    Dense,
    Dropout,
    GlobalAveragePooling2D
)

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import (
    EarlyStopping,
    ModelCheckpoint,
    ReduceLROnPlateau
)

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

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("TensorFlow Version :", tf.__version__)
print("=" * 60)

=====
TensorFlow Version : 2.20.0
=====
```

```
In [4]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies the TensorFlow version and
# GPU availability before model training.
# =====

SEED = 42

# Python random seed
random.seed(SEED)

# NumPy random seed
np.random.seed(SEED)

# TensorFlow random seed
tf.random.set_seed(SEED)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed      : {SEED}")
print(f"TensorFlow Version: {tf.__version__}")

gpus = tf.config.list_physical_devices('GPU')

if gpus:
    print(f"GPU Available    : Yes")
    print(f"GPU Device       : {gpus[0].name}")
else:
    print("GPU Available    : No")
    print("Please enable T4 GPU before training.")

print("=" * 60)

=====
Project Configuration
=====
Random Seed      : 42
TensorFlow Version: 2.20.0
GPU Available     : Yes
GPU Device       : /physical_device:GPU:0
=====
```

In [5]:

```
# =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive,
# verifies that it exists, and displays the available class
# folders before data preparation.
# =====

DATASET_DIR = "/content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset"

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):
    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")

    class_names = sorted(os.listdir(DATASET_DIR))

    print(f"Total Classes   : {len(class_names)}")
    print(f"Class Names      : {class_names}")

else:
    raise FileNotFoundError(
        f"\nDataset not found!\nPlease check the following path:\n{DATASET_DIR}"
    )

print("=" * 60)

=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
Class Names    : ['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 [6]:

```
# =====
# Step 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_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'
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, folder_name))
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes    : {len(EXPECTED_CLASSES)}")
print(f"Available Classes    : {len(available_classes)}")
print(f"Missing Classes      : {len(missing_classes)}")
print(f"Unexpected Classes    : {len(unexpected_classes)}")
print("=" * 60)

if not dataset_ready:
    raise ValueError("Dataset class verification failed.")

=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes    : 36
Available Classes    : 36
Missing Classes      : []
Unexpected Classes    : []
=====
```

In [7]:

```
# =====
# Step 7: Create Final Dataset Split
# This step creates one fixed Train, Validation, and Test split
# that will be used by all models for a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

required_split_folders = ["train", "val", "test"]

split_is_complete = all(
    os.path.isdir(os.path.join(SPLIT_DATASET_DIR, folder_name))
    for folder_name in required_split_folders
)

if not split_is_complete:

    # Remove the empty or incomplete split folder.
    if os.path.exists(SPLIT_DATASET_DIR):
        shutil.rmtree(SPLIT_DATASET_DIR)

    splitfolders.ratio(
        DATASET_DIR,
        output=SPLIT_DATASET_DIR,
        seed=SEED,
        ratio=(0.70, 0.10, 0.20),
        move=False
    )

    print("=" * 60)
    print("Dataset Split Created Successfully")
    print("=" * 60)

else:
    print("=" * 60)
    print("Complete Dataset Split Already Exists")
    print("=" * 60)

print(f"Split Dataset Path : {SPLIT_DATASET_DIR}")
print("=" * 60)

=====
Complete Dataset Split Already Exists
=====
Split Dataset Path : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final
=====
```

```
In [8]: # =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total
# number of images.
# =====

VALID_EXTENSIONS = (".jpg", ".jpeg", ".png", ".bmp", ".webp")

split_name_options = {
    "Training": ["train", "training"],
    "Validation": ["val", "valid", "validation"],
    "Testing": ["test", "testing"]
}

resolved_splits = {}

for display_name, possible_names in split_name_options.items():
    for folder_name in possible_names:
        folder_path = os.path.join(SPLIT_DATASET_DIR, folder_name)

        if os.path.isdir(folder_path):
            resolved_splits[display_name] = folder_path
            break

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

grand_total = 0

for display_name in ["Training", "Validation", "Testing"]:

    if display_name not in resolved_splits:
        raise FileNotFoundError(
            f"{display_name} folder was not found inside:\n"
            f"{SPLIT_DATASET_DIR}"
        )

    split_path = resolved_splits[display_name]
    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(split_path, class_name)

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(VALID_EXTENSIONS)
        )

    grand_total += total_images
    print(f"{display_name:<12}: {total_images} images")

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)

TRAIN_DIR = resolved_splits["Training"]
VALIDATION_DIR = resolved_splits["Validation"]
TEST_DIR = resolved_splits["Testing"]

print("Dataset split paths configured successfully.")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training   : 1760 images
Validation : 251 images
Testing    : 504 images
-----
Total Images : 2515
=====
Dataset split paths configured successfully.
=====
```

```
In [9]: # =====
# Step 9: Create Image Data Generators
# This step applies MobileNetV2-specific preprocessing and
# data augmentation to prepare the training, validation,
# and testing datasets.
# =====

IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.15,
    shear_range=0.10,
    brightness_range=[0.7, 1.3],
    horizontal_flip=False,
    fill_mode="nearest"
)

validation_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
    seed=SEED
)

validation_generator = validation_test_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

test_generator = validation_test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("=" * 60)
print("Image Data Generators Created Successfully")
print("=" * 60)
print(f"Image Size      : {IMAGE_SIZE}")
print(f"Batch Size       : {BATCH_SIZE}")
print(f"Number of Classes : {train_generator.num_classes}")
print("=" * 60)
```

```
Found 1760 images belonging to 36 classes.
Found 251 images belonging to 36 classes.
Found 504 images belonging to 36 classes.
=====
Image Data Generators Created Successfully
=====
Image Size      : (224, 224)
Batch Size     : 16
Number of Classes : 36
=====
```

```
In [10]: # =====
# Step 10: Display Dataset Information
# This step displays information about the dataset generators
# that will be used for model training and evaluation.
# =====

CLASS_NAMES = list(train_generator.class_indices.keys())
NUM_CLASSES = train_generator.num_classes

print("=" * 60)
print("Dataset Information")
print("=" * 60)
print(f"Training Samples : {train_generator.samples}")
print(f"Validation Samples : {validation_generator.samples}")
print(f"Testing Samples : {test_generator.samples}")
print(f"Number of Classes : {NUM_CLASSES}")
print(f"Batch Size : {BATCH_SIZE}")
print(f"Image Size : {IMAGE_SIZE}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Dataset Information
=====
Training Samples : 1760
Validation Samples : 251
Testing Samples : 504
Number of Classes : 36
Batch Size : 16
Image Size : (224, 224)
=====
Class Names
=====
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 [11]: # =====
# Step 11: Build the MobileNetV2 Model
# This step loads the pretrained MobileNetV2 model and replaces
# its original classification head with a new 36-class head.
# =====

base_model = MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)

base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.5)(x)

output = Dense(
    NUM_CLASSES,
    activation="softmax"
)(x)

model = Model(
    inputs=base_model.input,
    outputs=output,
    name="MobileNetV2_ASL_Model"
)

print("=" * 60)
print("MobileNetV2 Model Created Successfully")
print("=" * 60)
print(f"Base Model Trainable : {base_model.trainable}")
print(f"Number of Classes : {NUM_CLASSES}")
print("=" * 60)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
[1m9486464/9406464][0m [32m-----[0m [37m[0m [1m0s[0m 0us/step
=====
MobileNetV2 Model Created Successfully
=====
Base Model Trainable : False
Number of Classes : 36
=====
```

```
In [12]: # =====
# Step 12: Compile the MobileNetV2 Model
# This step configures the model for training by defining the
# optimizer, loss function, and evaluation metric.
# =====

from tensorflow.keras.optimizers import Adam

LEARNING_RATE = 0.0001

model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

print("=" * 60)
print("Model Compilation Completed Successfully")
print("=" * 60)
print(f"Optimizer : Adam")
print(f"Learning Rate : {LEARNING_RATE}")
print(f"Loss Function : Categorical Crossentropy")
print(f"Evaluation : Accuracy")
print("=" * 60)
```

```
=====
Model Compilation Completed Successfully
=====
Optimizer : Adam
Learning Rate : 0.0001
Loss Function : Categorical Crossentropy
Evaluation : Accuracy
=====
```

In [13]:

```
# =====
# Step 13: Configure Training Callbacks
# This step configures Early Stopping, Model Checkpoint,
# and Learning Rate Reduction during model training.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/MobileNetV2"
)

os.makedirs(MODEL_DIR, exist_ok=True)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_mobilenetv2_asl_model.keras"
)

callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=7,
        restore_best_weights=True,
        verbose=1
    ),

    ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.3,
        patience=4,
        min_lr=1e-6,
        verbose=1
    ),

    ModelCheckpoint(
        filepath=BEST_MODEL_PATH,
        monitor="val_accuracy",
        save_best_only=True,
        verbose=1
    )
]

print("=" * 60)
print("Training Callbacks Configured Successfully")
print("=" * 60)
print("Early Stopping Patience : 7")
print("LR Reduction Patience   : 4")
print(f"Best Model Path           : {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Callbacks Configured Successfully
=====
Early Stopping Patience : 7
LR Reduction Patience   : 4
Best Model Path         : /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
=====
```

In [14]:

```
# =====
# Step 14: Train the MobileNetV2 Model
# This step trains the ResNet50 model using the training
# and validation datasets.
# =====

EPOCHS = 30

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1
)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
```

```
Epoch 1/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 6s/step - accuracy: 0.0357 - loss: 3.8439
Epoch 1: val_accuracy improved from None to 0.21912, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 1: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m769s[0m 7s/step - accuracy: 0.0619 - loss: 3.5866 - val_accuracy: 0.2191 - val_loss: 3.0541 - learning_rate: 1.0000e-04
Epoch 2/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.1531 - loss: 3.1045
Epoch 2: val_accuracy improved from 0.21912 to 0.39841, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 2: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.1773 - loss: 3.0212 - val_accuracy: 0.3984 - val_loss: 2.5413 - learning_rate: 1.0000e-04
Epoch 3/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 230ms/step - accuracy: 0.2606 - loss: 2.6775
Epoch 3: val_accuracy improved from 0.39841 to 0.52191, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 3: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.2773 - loss: 2.5820 - val_accuracy: 0.5219 - val_loss: 2.1127 - learning_rate: 1.0000e-04
Epoch 4/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.3500 - loss: 2.3386
Epoch 4: val_accuracy improved from 0.52191 to 0.61753, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 4: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 247ms/step - accuracy: 0.3483 - loss: 2.2855 - val_accuracy: 0.6175 - val_loss: 1.8102 - learning_rate: 1.0000e-04
Epoch 5/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 230ms/step - accuracy: 0.3943 - loss: 2.0957
Epoch 5: val_accuracy improved from 0.61753 to 0.63745, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 5: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.4091 - loss: 2.0312 - val_accuracy: 0.6375 - val_loss: 1.5975 - learning_rate: 1.0000e-04
Epoch 6/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 228ms/step - accuracy: 0.4397 - loss: 1.9141
Epoch 6: val_accuracy improved from 0.63745 to 0.69721, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 6: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m26s[0m 240ms/step - accuracy: 0.4597 - loss: 1.8509 - val_accuracy: 0.6972 - val_loss: 1.4135 - learning_rate: 1.0000e-04
Epoch 7/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 229ms/step - accuracy: 0.4903 - loss: 1.7122
Epoch 7: val_accuracy improved from 0.69721 to 0.70916, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 7: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 241ms/step - accuracy: 0.4903 - loss: 1.7136 - val_accuracy: 0.7092 - val_loss: 1.3146 - learning_rate: 1.0000e-04
Epoch 8/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 229ms/step - accuracy: 0.5340 - loss: 1.5747
Epoch 8: val_accuracy improved from 0.70916 to 0.72908, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 8: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.5409 - loss: 1.5523 - val_accuracy: 0.7291 - val_loss: 1.1670 - learning_rate: 1.0000e-04
Epoch 9/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 234ms/step - accuracy: 0.5818 - loss: 1.4533
Epoch 9: val_accuracy did not improve from 0.72908
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 244ms/step - accuracy: 0.5699 - loss: 1.4568 - val_accuracy: 0.7291 - val_loss: 1.0751 - learning_rate: 1.0000e-04
Epoch 10/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.6049 - loss: 1.3708
Epoch 10: val_accuracy improved from 0.72908 to 0.73307, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 10: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 248ms/step - accuracy: 0.6006 - loss: 1.3511 - val_accuracy: 0.7331 - val_loss: 1.0070 - learning_rate: 1.0000e-04
Epoch 11/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 242ms/step - accuracy: 0.6012 - loss: 1.3355
Epoch 11: val_accuracy improved from 0.73307 to 0.75697, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 11: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m28s[0m 255ms/step - accuracy: 0.6011 - loss: 1.3013 - val_accuracy: 0.7570 - val_loss: 0.9374 - learning_rate: 1.0000e-04
Epoch 12/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 237ms/step - accuracy: 0.6349 - loss: 1.2057
Epoch 12: val_accuracy did not improve from 0.75697
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 246ms/step - accuracy: 0.6210 - loss: 1.2271 - val_accuracy: 0.7410 - val_loss: 0.9058 - learning_rate: 1.0000e-04
Epoch 13/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.6697 - loss: 1.1262
Epoch 13: val_accuracy improved from 0.75697 to 0.78486, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 13: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 249ms/step - accuracy: 0.6517 - loss: 1.1417 - val_accuracy: 0.7849 - val_loss: 0.8455 - learning_rate: 1.0000e-04
Epoch 14/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 234ms/step - accuracy: 0.6944 - loss: 1.0575
Epoch 14: val_accuracy did not improve from 0.78486
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 243ms/step - accuracy: 0.6790 - loss: 1.0803 - val_accuracy: 0.7729 - val_loss: 0.7858 - learning_rate: 1.0000e-04
Epoch 15/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.6706 - loss: 1.0688
Epoch 15: val_accuracy did not improve from 0.78486
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.6858 - loss: 1.0493 - val_accuracy: 0.7809 - val_loss: 0.7723 - learning_rate: 1.0000e-04
Epoch 16/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 237ms/step - accuracy: 0.7221 - loss: 0.9580
Epoch 16: val_accuracy improved from 0.78486 to 0.81275, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 16: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 250ms/step - accuracy: 0.6989 - loss: 0.9874 - val_accuracy: 0.8127 - val_loss: 0.7275 - learning_rate: 1.0000e-04
Epoch 17/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 238ms/step - accuracy: 0.7069 - loss: 0.9527
Epoch 17: val_accuracy did not improve from 0.81275
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 246ms/step - accuracy: 0.6960 - loss: 0.9626 - val_accuracy: 0.8048 - val_loss: 0.6925 - learning_rate: 1.0000e-04
Epoch 18/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 231ms/step - accuracy: 0.7022 - loss: 0.9788
Epoch 18: val_accuracy did not improve from 0.81275
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 241ms/step - accuracy: 0.7188 - loss: 0.9180 - val_accuracy: 0.8048 - val_loss: 0.6782 - learning_rate: 1.0000e-04
Epoch 19/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 236ms/step - accuracy: 0.7242 - loss: 0.9180
Epoch 19: val_accuracy did not improve from 0.81275
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.7188 - loss: 0.9077 - val_accuracy: 0.8088 - val_loss: 0.6730 - learning_rate: 1.0000e-04
Epoch 20/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.7267 - loss: 0.8995
Epoch 20: val_accuracy did not improve from 0.81275
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 244ms/step - accuracy: 0.7239 - loss: 0.8758 - val_accuracy: 0.7849 - val_loss: 0.6663 - learning_rate: 1.0000e-04
Epoch 21/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.7540 - loss: 0.8390
Epoch 21: val_accuracy did not improve from 0.81275
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 244ms/step - accuracy: 0.7648 - loss: 0.8139 - val_accuracy: 0.7888 - val_loss: 0.6220 - learning_rate: 1.0000e-04
Epoch 22/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.7754 - loss: 0.7755
Epoch 22: val_accuracy did not improve from 0.81275
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 241ms/step - accuracy: 0.7636 - loss: 0.8072 - val_accuracy: 0.8127 - val_loss: 0.6221 - learning_rate: 1.0000e-04
Epoch 23/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.7485 - loss: 0.8060
Epoch 23: val_accuracy improved from 0.81275 to 0.82072, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 23: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 246ms/step - accuracy: 0.7528 - loss: 0.7850 - val_accuracy: 0.8207 - val_loss: 0.6088 - learning_rate: 1.0000e-04
Epoch 24/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.7772 - loss: 0.7410
Epoch 24: val_accuracy improved from 0.82072 to 0.82470, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 24: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 248ms/step - accuracy: 0.7778 - loss: 0.7585 - val_accuracy: 0.8247 - val_loss: 0.5682 - learning_rate: 1.0000e-04
Epoch 25/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 232ms/step - accuracy: 0.7762 - loss: 0.7307
Epoch 25: val_accuracy improved from 0.82470 to 0.82869, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 25: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.7750 - loss: 0.7313 - val_accuracy: 0.8287 - val_loss: 0.5539 - learning_rate: 1.0000e-04
Epoch 26/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 232ms/step - accuracy: 0.7954 - loss: 0.6879
Epoch 26: val_accuracy did not improve from 0.82869
[1m10/110[0m @32m-----[0m[37m[0m @1m26s[0m 241ms/step - accuracy: 0.7909 - loss: 0.6929 - val_accuracy: 0.8247 - val_loss: 0.5463 - learning_rate: 1.0000e-04
Epoch 27/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 231ms/step - accuracy: 0.7744 - loss: 0.7057
Epoch 27: val_accuracy improved from 0.82869 to 0.85657, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 27: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.7744 - loss: 0.7128 - val_accuracy: 0.8566 - val_loss: 0.5239 - learning_rate: 1.0000e-04
Epoch 28/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 234ms/step - accuracy: 0.7922 - loss: 0.6692
Epoch 28: val_accuracy did not improve from 0.85657
[1m10/110[0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.7875 - loss: 0.6857 - val_accuracy: 0.8406 - val_loss: 0.5456 - learning_rate: 1.0000e-04
Epoch 29/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 230ms/step - accuracy: 0.7844 - loss: 0.7204
Epoch 29: val_accuracy did not improve from 0.85657
[1m10/110[0m @32m-----[0m[37m[0m @1m26s[0m 239ms/step - accuracy: 0.7989 - loss: 0.6645 - val_accuracy: 0.8526 - val_loss: 0.5107 - learning_rate: 1.0000e-04
Epoch 30/30
[1m10/110[0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.8139 - loss: 0.6237
```

```
Epoch 30: val_accuracy did not improve from 0.85657
[1m110/110][0m  [32m-----[0m[37m[0m  [1m27s[0m  242ms/step - accuracy: 0.8080 - loss: 0.6377 - val_accuracy: 0.8446 - val_loss: 0.4992 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.
=====
Model Training Completed Successfully
=====
```

```
In [15]: # =====
# Step 15: Evaluate the MobileNetV2 Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

test_loss, test_accuracy = model.evaluate(
    test_generator,
    verbose=1
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
[1m32/32][0m  [32m-----[0m[37m[0m  [1m180s[0m  6s/step - accuracy: 0.9187 - loss: 0.3565
=====
Model Evaluation
=====
Test Loss      : 0.3565
Test Accuracy : 0.9187
=====
```

```
In [16]: # =====
# Step 16: Generate Predictions and Classification Report
# This step generates predictions on the testing dataset and
# displays the classification report.
# =====

from sklearn.metrics import classification_report
import numpy as np

# Reset the test generator
test_generator.reset()

# Generate predictions
predictions = model.predict(test_generator, verbose=1)

# Predicted and true labels
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

# Classification Report
report = classification_report(
    y_true,
    y_pred,
    target_names=list(test_generator.class_indices.keys()),
    digits=4
)

print("=" * 60)
print("Classification Report")
print("=" * 60)
print(report)
print("=" * 60)
```

```
[1m32/32][0m  [32m-----[0m[37m[0m  [1m11s[0m  189ms/step
=====
Classification Report
=====
              precision    recall  f1-score   support

    0   0.6667    0.8571    0.7500        14
    1   0.8750    1.0000    0.9333        14
    2   1.0000    0.8571    0.9231        14
    3   1.0000    1.0000    1.0000        14
    4   0.8571    0.8571    0.8571        14
    5   0.8000    0.8571    0.8276        14
    6   0.7857    0.7857    0.7857        14
    7   0.8235    1.0000    0.9032        14
    8   1.0000    0.8571    0.9231        14
    9   1.0000    0.9286    0.9630        14
    a   0.8667    0.9286    0.8966        14
    b   1.0000    1.0000    1.0000        14
    c   1.0000    1.0000    1.0000        14
    d   1.0000    1.0000    1.0000        14
    e   1.0000    1.0000    1.0000        14
    f   1.0000    1.0000    1.0000        14
    g   1.0000    1.0000    1.0000        14
    h   1.0000    1.0000    1.0000        14
    i   1.0000    1.0000    1.0000        14
    j   1.0000    1.0000    1.0000        14
    k   0.9333    1.0000    0.9655        14
    l   1.0000    1.0000    1.0000        14
    m   0.8000    0.8571    0.8276        14
    n   1.0000    0.7857    0.8800        14
    o   0.8000    0.5714    0.6667        14
    p   1.0000    1.0000    1.0000        14
    q   1.0000    1.0000    1.0000        14
    r   1.0000    0.8571    0.9231        14
    s   1.0000    0.6429    0.7826        14
    t   0.6190    0.9286    0.7429        14
    u   1.0000    1.0000    1.0000        14
    v   0.9167    0.7857    0.8462        14
    w   0.6875    0.7857    0.7333        14
    x   1.0000    1.0000    1.0000        14
    y   1.0000    0.9286    0.9630        14
    z   1.0000    1.0000    1.0000        14

 accuracy          0.9187         504
 macro avg         0.9286    0.9187    0.9193         504
weighted avg         0.9286    0.9187    0.9193         504

=====
```

In [17]:

```
# =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "mobilenetv2_confusion_matrix.png"
)

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(16, 14))
plt.imshow(cm, cmap="Blues")

plt.title(
    "MobileNetV2 Confusion Matrix",
    fontsize=18
)

plt.xlabel("Predicted Label", fontsize=14)
plt.ylabel("True Label", fontsize=14)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):

        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

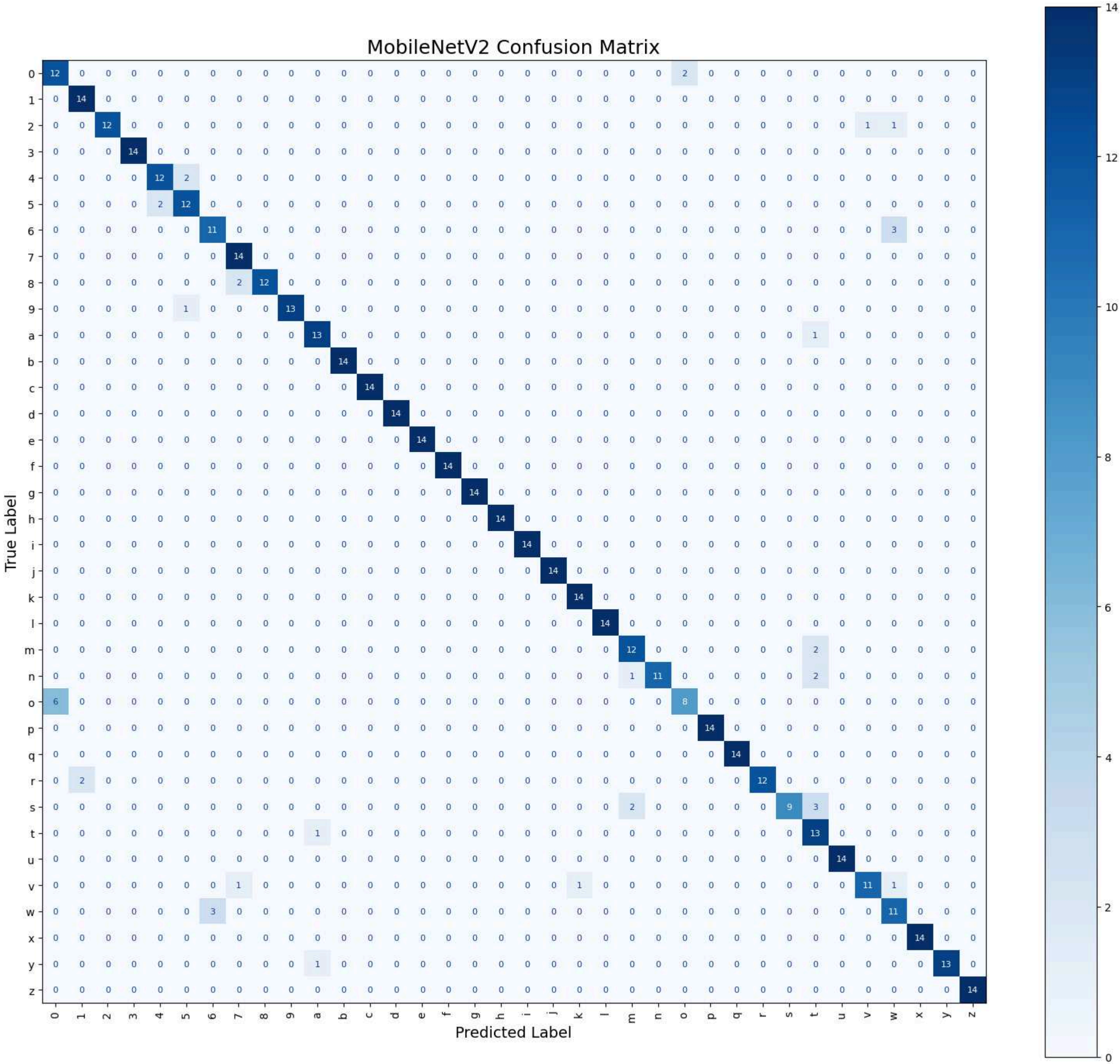
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



```
=====
Confusion Matrix Saved Successfully
=====
Saved Path : /content/drive/MyDrive/ASL_Final_Project/03_Results/Confusion_Matrices/mobilenetv2_confusion_matrix.png
=====
```

In [18]:

```
# =====
# Step 18: Plot Training History
# This step plots the training and validation accuracy and
# loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "mobilenetv2_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "mobilenetv2_loss_curve.png"
)

# Accuracy Curve
plt.figure(figsize=(8, 6))

plt.plot(
    history.history["accuracy"],
    label="Training Accuracy"
)

plt.plot(
    history.history["val_accuracy"],
    label="Validation Accuracy"
)

plt.title("MobileNetV2 Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

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

plt.show()

# Loss Curve
plt.figure(figsize=(8, 6))

plt.plot(
    history.history["loss"],
    label="Training Loss"
)

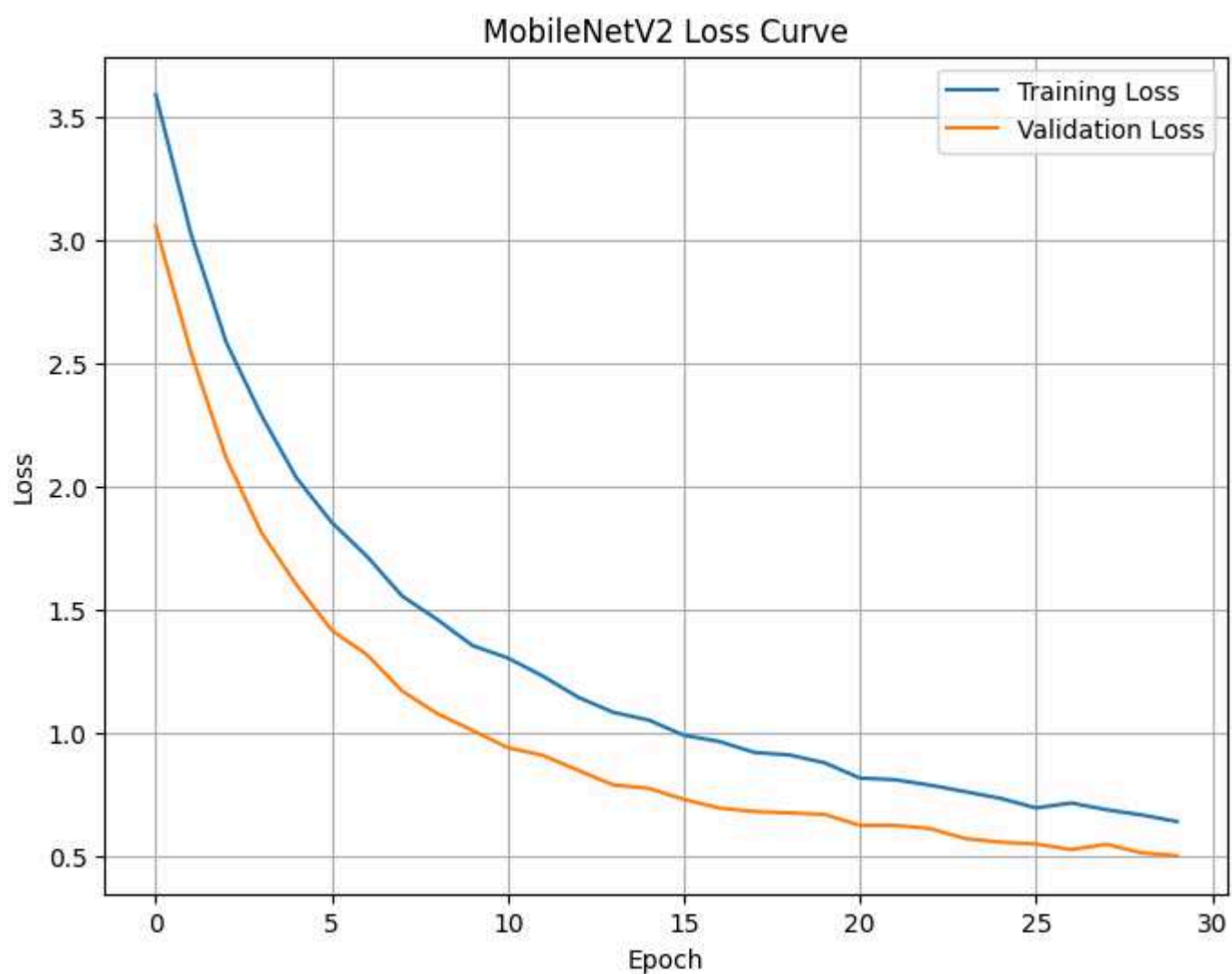
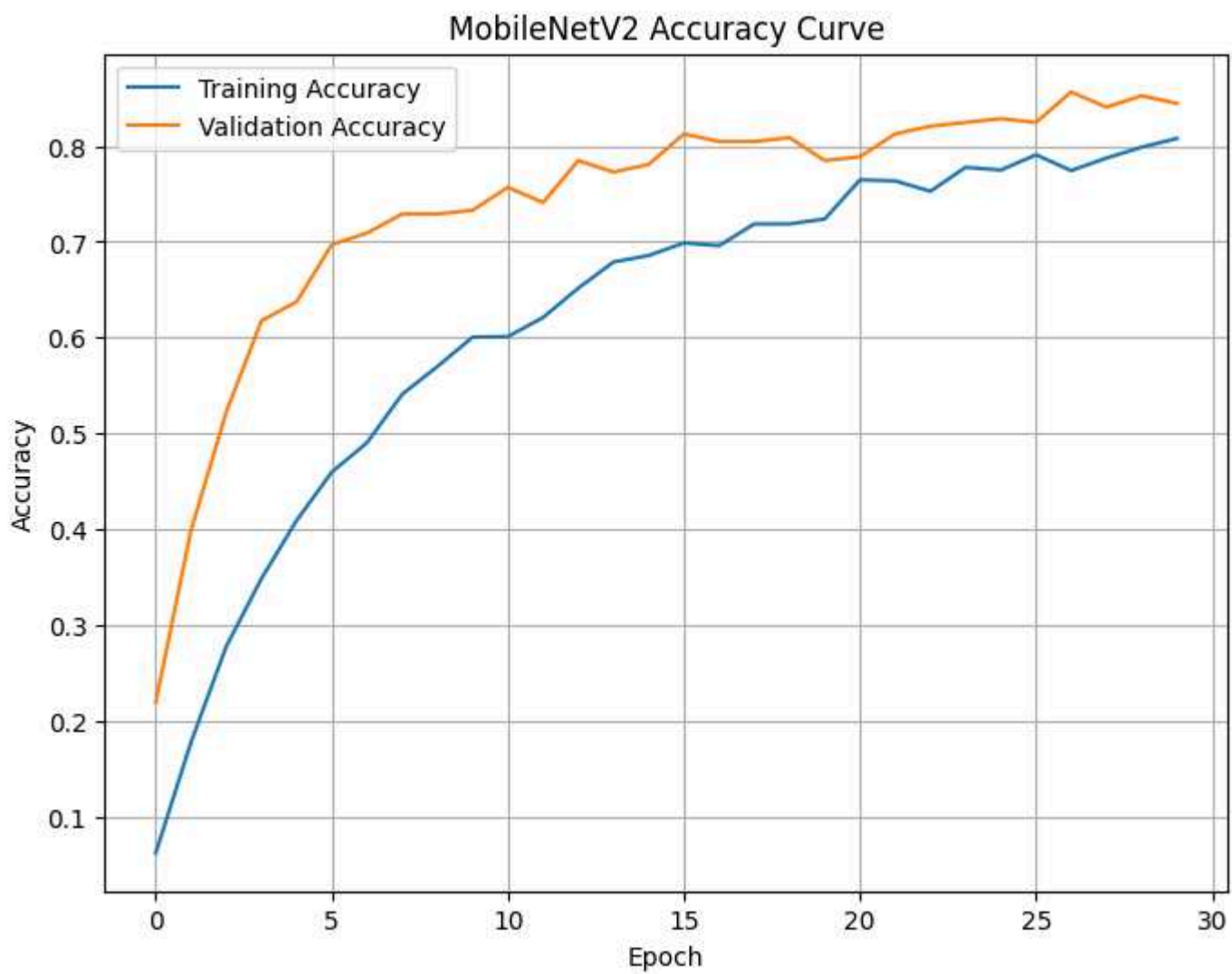
plt.plot(
    history.history["val_loss"],
    label="Validation Loss"
)

plt.title("MobileNetV2 Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

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

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/mobilenetv2_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/mobilenetv2_loss_curve.png
=====
```

In [19]:

```
# =====
# Step 19: Save the Final MobileNetV2 Model
# This step saves the trained MobileNetV2 model and the class
# names for future inference and deployment.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/MobileNetV2"
)

os.makedirs(MODEL_DIR, exist_ok=True)

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_mobilenetv2_asl_model.keras"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "mobilenetv2_class_names.txt"
)

model.save(FINAL_MODEL_PATH)

with open(CLASS_NAMES_PATH, "w") as file:
    for class_name in CLASS_NAMES:
        file.write(f"{class_name}\n")

print("=" * 60)
print("Model Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print("=" * 60)

=====
Model Saved Successfully
=====
Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/final_mobilenetv2_asl_model.keras
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/mobilenetv2_class_names.txt
=====
```

In [20]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures the model inference time on the complete
# test dataset using a warm-up run and repeated measurements.
# =====

INFERENCE_REPEATS = 3

# Warm-up run
test_generator.reset()
_ = model.predict(
    test_generator,
    verbose=0
)

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    test_generator.reset()

    start_time = time.perf_counter()

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

    end_time = time.perf_counter()

    inference_times.append(end_time - start_time)

average_inference_time = np.mean(inference_times)
std_inference_time = np.std(inference_times)

average_time_per_image = (
    average_inference_time / test_generator.samples
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print(f"Test Images          : {test_generator.samples}")
print(f"Repeated Measurements  : {INFERENCE_REPEATS}")
print(f"Average Total Time     : {average_inference_time:.4f} seconds")
print(f"Standard Deviation     : {std_inference_time:.4f} seconds")
print(f"Average Time per Image : {average_time_per_image:.6f} seconds")
print("=" * 60)

=====
Standardized Inference Time
=====
Test Images          : 504
Repeated Measurements : 3
Average Total Time   : 1.8144 seconds
Standard Deviation   : 0.0810 seconds
Average Time per Image : 0.003600 seconds
=====
```

```
In [21]: # =====
# Step 21: Save Final MobileNetV2 Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "mobilenetv2"
MODEL_DISPLAY_NAME = "MobileNetV2"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/MobileNetV2/final_mobilenetv2_asl_model.keras"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "mobilenetv2_metrics.json"
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpus = tf.config.list_physical_devices("GPU")

if gpus:
    gpu_name = (
        tf.config.experimental
        .get_device_details(gpus[0])
        .get("device_name", gpus[0].name)
    )
else:
    gpu_name = "CPU"

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "TensorFlow/Keras",
    "gpu": gpu_name,
    "test_images": int(test_generator.samples),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(model.count_params()),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("MobileNetV2 Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size     : {model_size_mb:.2f} MB")
print(f"Parameters     : {model.count_params():,}")
print(f"Metrics File   : {METRICS_PATH}")
print("=" * 60)
```

```
=====
MobileNetV2 Metrics Saved Successfully
=====
Accuracy      : 0.9187
Precision     : 0.9286
Recall        : 0.9187
F1-Score      : 0.9193
Model Size    : 13.03 MB
Parameters    : 2,595,172
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/mobilenetv2_metrics.json
=====
```

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

drive.mount("/content/drive")

Mounted at /content/drive
```

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs the additional libraries required for
# dataset splitting and spelling correction.
# =====

!pip -q install split-folders
!pip -q install textblob
```

```
In [3]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data
# preprocessing, model training, evaluation, visualization,
# and result analysis.
# =====

import os
import random
import shutil
import time
import json
import warnings

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

from PIL import Image

import tensorflow as tf
import splitfolders

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.efficientnet import preprocess_input

from tensorflow.keras.models import Model

from tensorflow.keras.layers import (
    Dense,
    Dropout,
    GlobalAveragePooling2D
)

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import (
    EarlyStopping,
    ModelCheckpoint,
    ReduceLROnPlateau
)

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

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("TensorFlow Version :", tf.__version__)
print("=" * 60)

=====
TensorFlow Version : 2.20.0
=====
```

```
In [4]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies the TensorFlow version and
# GPU availability before model training.
# =====

SEED = 42

random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed      : {SEED}")
print(f"TensorFlow Version : {tf.__version__}")

gpus = tf.config.list_physical_devices("GPU")

if gpus:
    print("GPU Available      : Yes")
    print(f"GPU Device          : {gpus[0].name}")
else:
    raise RuntimeError(
        "GPU was not detected. Enable A100 GPU before training."
    )

print("=" * 60)

=====
Project Configuration
=====
Random Seed      : 42
TensorFlow Version : 2.20.0
GPU Available     : Yes
GPU Device       : /physical_device:GPU:0
=====
```

In [5]:

```
# =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive,
# verifies that it exists, and displays the available class
# folders before data preparation.
# =====

DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Original/asl_dataset"
)

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):

    class_names = sorted([
        folder_name
        for folder_name in os.listdir(DATASET_DIR)
        if os.path.isdir(os.path.join(DATASET_DIR, folder_name))
    ])

    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")
    print(f"Total Classes   : {len(class_names)}")
    print(f"Class Names      : {class_names}")

else:
    raise FileNotFoundError(
        f"\nDataset was not found.\n"
        f"Please check this path:\n{DATASET_DIR}"
    )

print("=" * 60)

=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
Class Names    : ['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 [6]:

```
# =====
# Step 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_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"
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, folder_name))
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes    : {len(EXPECTED_CLASSES)}")
print(f"Available Classes   : {len(available_classes)}")
print(f"Missing Classes     : {missing_classes}")
print(f"Unexpected Classes  : {unexpected_classes}")
print("=" * 60)

if not dataset_ready:
    raise ValueError("Dataset class verification failed.")

=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes    : 36
Available Classes   : 36
Missing Classes     : []
Unexpected Classes  : []
=====
```

In [7]:

```
# =====
# Step 7: Load or Create Final Dataset Split
# This step uses the same fixed Train, Validation, and Test
# split for all models to ensure a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

required_split_folders = ["train", "val", "test"]

split_is_complete = all(
    os.path.isdir(
        os.path.join(SPLIT_DATASET_DIR, folder_name)
    )
    for folder_name in required_split_folders
)

if not split_is_complete:

    if os.path.exists(SPLIT_DATASET_DIR):
        shutil.rmtree(SPLIT_DATASET_DIR)

    splitfolders.ratio(
        DATASET_DIR,
        output=SPLIT_DATASET_DIR,
        seed=SEED,
        ratio=(0.70, 0.10, 0.20),
        move=False
    )

    print("=" * 60)
    print("Dataset Split Created Successfully")
    print("=" * 60)

else:
    print("=" * 60)
    print("Complete Dataset Split Already Exists")
    print("=" * 60)

print(f"Split Dataset Path : {SPLIT_DATASET_DIR}")
print("=" * 60)

=====
Complete Dataset Split Already Exists
=====
Split Dataset Path : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final
=====
```

```
In [8]: # =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total.
# =====

VALID_EXTENSIONS = (
    ".jpg",
    ".jpeg",
    ".png",
    ".bmp",
    ".webp"
)

split_name_options = {
    "Training": ["train", "training"],
    "Validation": ["val", "valid", "validation"],
    "Testing": ["test", "testing"]
}

resolved_splits = {}

for display_name, possible_names in split_name_options.items():

    for folder_name in possible_names:

        folder_path = os.path.join(
            SPLIT_DATASET_DIR,
            folder_name
        )

        if os.path.isdir(folder_path):
            resolved_splits[display_name] = folder_path
            break

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

grand_total = 0

for display_name in ["Training", "Validation", "Testing"]:

    if display_name not in resolved_splits:
        raise FileNotFoundError(
            f"{display_name} folder was not found inside:\n"
            f"{SPLIT_DATASET_DIR}"
        )

    split_path = resolved_splits[display_name]
    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(
            split_path,
            class_name
        )

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(VALID_EXTENSIONS)
        )

    grand_total += total_images

    print(
        f"{display_name:<12}: "
        f"{total_images} images"
    )

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)

TRAIN_DIR = resolved_splits["Training"]
VALIDATION_DIR = resolved_splits["Validation"]
TEST_DIR = resolved_splits["Testing"]

print("Dataset split paths configured successfully.")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training      : 1760 images
Validation    : 251 images
Testing       : 504 images
-----
Total Images  : 2515
=====
Dataset split paths configured successfully.
=====
```

```
In [9]: # =====
# Step 9: Create Image Data Generators
# This step applies EfficientNetB0-specific preprocessing and
# data augmentation to prepare the training, validation,
# and testing datasets.
# =====

IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.15,
    shear_range=0.10,
    brightness_range=[0.7, 1.3],
    horizontal_flip=False,
    fill_mode="nearest"
)

validation_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
    seed=SEED
)

validation_generator = validation_test_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

test_generator = validation_test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("=" * 60)
print("Image Data Generators Created Successfully")
print("=" * 60)
print(f"Image Size      : {IMAGE_SIZE}")
print(f"Batch Size       : {BATCH_SIZE}")
print(f"Number of Classes : {train_generator.num_classes}")
print("=" * 60)
```

```
Found 1760 images belonging to 36 classes.
Found 251 images belonging to 36 classes.
Found 504 images belonging to 36 classes.
=====
Image Data Generators Created Successfully
=====
Image Size      : (224, 224)
Batch Size      : 16
Number of Classes : 36
=====
```

```
In [10]: # =====
# Step 10: Display Dataset Information
# This step displays information about the dataset generators
# that will be used for model training and evaluation.
# =====

CLASS_NAMES = list(
    train_generator.class_indices.keys()
)

NUM_CLASSES = train_generator.num_classes

print("=" * 60)
print("Dataset Information")
print("=" * 60)
print(f"Training Samples   : {train_generator.samples}")
print(f"Validation Samples : {validation_generator.samples}")
print(f"Testing Samples    : {test_generator.samples}")
print(f"Number of Classes   : {NUM_CLASSES}")
print(f"Batch Size         : {BATCH_SIZE}")
print(f"Image Size         : {IMAGE_SIZE}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Dataset Information
=====
Training Samples   : 1760
Validation Samples : 251
Testing Samples    : 504
Number of Classes   : 36
Batch Size         : 16
Image Size         : (224, 224)
=====
Class Names
=====
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 [11]: # =====
# Step 11: Build the EfficientNetB0 Model
# This step loads the pretrained EfficientNetB0 model and
# replaces its original head with a new 36-class classifier.
# =====

base_model = EfficientNetB0(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)

base_model.trainable = False

x = base_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(
    256,
    activation="relu"
)(x)

x = Dropout(0.5)(x)

output = Dense(
    NUM_CLASSES,
    activation="softmax"
)(x)

model = Model(
    inputs=base_model.input,
    outputs=output,
    name="EfficientNetB0_AS_L_Model"
)

print("=" * 60)
print("EfficientNetB0 Model Created Successfully")
print("=" * 60)
print(f"Base Model Trainable : {base_model.trainable}")
print(f"Number of Classes    : {NUM_CLASSES}")
print(f"Total Parameters      : {model.count_params():,}")
print("=" * 60)
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
[1m16705208/16705208][0m [32m-----[0m [37m[0m [1m2s[0m 0us/step
EfficientNetB0 Model Created Successfully
=====
Base Model Trainable : False
Number of Classes    : 36
Total Parameters      : 4,386,759
=====
```

```
In [12]: # =====
# Step 12: Compile the EfficientNetB0 Model
# This step configures the optimizer, loss function,
# and evaluation metric used during model training.
# =====

LEARNING_RATE = 0.0001

model.compile(
    optimizer=Adam(
        learning_rate=LEARNING_RATE
    ),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

print("=" * 60)
print("Model Compilation Completed Successfully")
print("=" * 60)
print(f"Optimizer           : Adam")
print(f"Learning Rate      : {LEARNING_RATE}")
print(f"Loss Function      : Categorical Crossentropy")
print(f"Evaluation         : Accuracy")
print("=" * 60)
```

```
=====
Model Compilation Completed Successfully
=====
Optimizer           : Adam
Learning Rate      : 0.0001
Loss Function      : Categorical Crossentropy
Evaluation         : Accuracy
=====
```

In [13]:

```
# =====
# Step 13: Configure Training Callbacks
# This step configures Early Stopping, Model Checkpoint,
# and Learning Rate Reduction during model training.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/EfficientNetB0"
)

os.makedirs(
    MODEL_DIR,
    exist_ok=True
)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_efficientnetb0_asl_model.keras"
)

callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=7,
        restore_best_weights=True,
        verbose=1
    ),

    ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.3,
        patience=4,
        min_lr=1e-6,
        verbose=1
    ),

    ModelCheckpoint(
        filepath=BEST_MODEL_PATH,
        monitor="val_accuracy",
        save_best_only=True,
        verbose=1
    )
]

print("=" * 60)
print("Training Callbacks Configured Successfully")
print("=" * 60)
print("Early Stopping Patience : 7")
print("LR Reduction Patience   : 4")
print(f"Best Model Path           : {BEST_MODEL_PATH}")
print("=" * 60)

=====
Training Callbacks Configured Successfully
=====
Early Stopping Patience : 7
LR Reduction Patience   : 4
Best Model Path         : /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
=====
```

```
In [14]: # =====
# Step 14: Train the EfficientNetB0 Model
# This step trains the EfficientNetB0 model using the
# Training and Validation datasets.
# =====

EPOCHS = 30

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1
)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
```

Epoch 1/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 12s/step - accuracy: 0.0337 - loss: 3.6286
Epoch 1: val_accuracy improved from None to 0.21514, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 1: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m1569s@0m 14s/step - accuracy: 0.0472 - loss: 3.5471 - val_accuracy: 0.2151 - val_loss: 3.3066 - learning_rate: 1.0000e-04
Epoch 2/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 222ms/step - accuracy: 0.1216 - loss: 3.2946
Epoch 2: val_accuracy improved from 0.21514 to 0.34263, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 2: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 236ms/step - accuracy: 0.1398 - loss: 3.2478 - val_accuracy: 0.3426 - val_loss: 2.9981 - learning_rate: 1.0000e-04
Epoch 3/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 231ms/step - accuracy: 0.2005 - loss: 3.0521
Epoch 3: val_accuracy improved from 0.34263 to 0.51793, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 3: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 245ms/step - accuracy: 0.2176 - loss: 2.9865 - val_accuracy: 0.5179 - val_loss: 2.6769 - learning_rate: 1.0000e-04
Epoch 4/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 222ms/step - accuracy: 0.2750 - loss: 2.7714
Epoch 4: val_accuracy improved from 0.51793 to 0.57371, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 4: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 237ms/step - accuracy: 0.2795 - loss: 2.7320 - val_accuracy: 0.5737 - val_loss: 2.3964 - learning_rate: 1.0000e-04
Epoch 5/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 227ms/step - accuracy: 0.3070 - loss: 2.5768
Epoch 5: val_accuracy improved from 0.57371 to 0.61753, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 5: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 240ms/step - accuracy: 0.3244 - loss: 2.5085 - val_accuracy: 0.6175 - val_loss: 2.1392 - learning_rate: 1.0000e-04
Epoch 6/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 220ms/step - accuracy: 0.3840 - loss: 2.3116
Epoch 6: val_accuracy improved from 0.61753 to 0.64940, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 6: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 234ms/step - accuracy: 0.3858 - loss: 2.2886 - val_accuracy: 0.6494 - val_loss: 1.9339 - learning_rate: 1.0000e-04
Epoch 7/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 223ms/step - accuracy: 0.4319 - loss: 2.1433
Epoch 7: val_accuracy improved from 0.64940 to 0.65737, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 7: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 237ms/step - accuracy: 0.4187 - loss: 2.1241 - val_accuracy: 0.6574 - val_loss: 1.7733 - learning_rate: 1.0000e-04
Epoch 8/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 239ms/step - accuracy: 0.4467 - loss: 2.0053
Epoch 8: val_accuracy improved from 0.65737 to 0.70916, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 8: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m28s@0m 257ms/step - accuracy: 0.4608 - loss: 1.9832 - val_accuracy: 0.7092 - val_loss: 1.6218 - learning_rate: 1.0000e-04
Epoch 9/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 232ms/step - accuracy: 0.4944 - loss: 1.8911
Epoch 9: val_accuracy improved from 0.70916 to 0.72112, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 9: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 247ms/step - accuracy: 0.4761 - loss: 1.8783 - val_accuracy: 0.7211 - val_loss: 1.5234 - learning_rate: 1.0000e-04
Epoch 10/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 238ms/step - accuracy: 0.4939 - loss: 1.7822
Epoch 10: val_accuracy did not improve from 0.72112
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 247ms/step - accuracy: 0.4989 - loss: 1.7582 - val_accuracy: 0.7092 - val_loss: 1.4410 - learning_rate: 1.0000e-04
Epoch 11/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 227ms/step - accuracy: 0.5158 - loss: 1.7159
Epoch 11: val_accuracy improved from 0.72112 to 0.74104, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 11: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 241ms/step - accuracy: 0.5432 - loss: 1.6498 - val_accuracy: 0.7410 - val_loss: 1.3533 - learning_rate: 1.0000e-04
Epoch 12/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 232ms/step - accuracy: 0.5594 - loss: 1.5918
Epoch 12: val_accuracy did not improve from 0.74104
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 241ms/step - accuracy: 0.5653 - loss: 1.5908 - val_accuracy: 0.7291 - val_loss: 1.2762 - learning_rate: 1.0000e-04
Epoch 13/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 231ms/step - accuracy: 0.5654 - loss: 1.5549
Epoch 13: val_accuracy did not improve from 0.74104
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 241ms/step - accuracy: 0.5710 - loss: 1.5183 - val_accuracy: 0.7211 - val_loss: 1.2014 - learning_rate: 1.0000e-04
Epoch 14/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 229ms/step - accuracy: 0.6014 - loss: 1.4177
Epoch 14: val_accuracy improved from 0.74104 to 0.75697, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 14: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 243ms/step - accuracy: 0.5864 - loss: 1.4556 - val_accuracy: 0.7570 - val_loss: 1.1476 - learning_rate: 1.0000e-04
Epoch 15/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 234ms/step - accuracy: 0.5997 - loss: 1.3974
Epoch 15: val_accuracy improved from 0.75697 to 0.76096, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 15: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 248ms/step - accuracy: 0.6102 - loss: 1.3815 - val_accuracy: 0.7610 - val_loss: 1.1053 - learning_rate: 1.0000e-04
Epoch 16/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 229ms/step - accuracy: 0.6096 - loss: 1.3363
Epoch 16: val_accuracy improved from 0.76096 to 0.76892, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 16: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 243ms/step - accuracy: 0.6165 - loss: 1.3400 - val_accuracy: 0.7689 - val_loss: 1.0371 - learning_rate: 1.0000e-04
Epoch 17/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 231ms/step - accuracy: 0.6137 - loss: 1.3292
Epoch 17: val_accuracy did not improve from 0.76892
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 239ms/step - accuracy: 0.6284 - loss: 1.2907 - val_accuracy: 0.7530 - val_loss: 1.0184 - learning_rate: 1.0000e-04
Epoch 18/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 231ms/step - accuracy: 0.6602 - loss: 1.2370
Epoch 18: val_accuracy did not improve from 0.76892
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 240ms/step - accuracy: 0.6619 - loss: 1.2259 - val_accuracy: 0.7570 - val_loss: 0.9663 - learning_rate: 1.0000e-04
Epoch 19/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 227ms/step - accuracy: 0.6409 - loss: 1.2429
Epoch 19: val_accuracy improved from 0.76892 to 0.78486, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 19: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 241ms/step - accuracy: 0.6545 - loss: 1.1955 - val_accuracy: 0.7849 - val_loss: 0.9485 - learning_rate: 1.0000e-04
Epoch 20/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 237ms/step - accuracy: 0.6597 - loss: 1.1781
Epoch 20: val_accuracy did not improve from 0.78486
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 246ms/step - accuracy: 0.6585 - loss: 1.1706 - val_accuracy: 0.7809 - val_loss: 0.9018 - learning_rate: 1.0000e-04
Epoch 21/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 229ms/step - accuracy: 0.6575 - loss: 1.1571
Epoch 21: val_accuracy improved from 0.78486 to 0.79283, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 21: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m27s@0m 243ms/step - accuracy: 0.6790 - loss: 1.1245 - val_accuracy: 0.7928 - val_loss: 0.8600 - learning_rate: 1.0000e-04
Epoch 22/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 226ms/step - accuracy: 0.6951 - loss: 1.1098
Epoch 22: val_accuracy improved from 0.79283 to 0.80080, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 22: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 240ms/step - accuracy: 0.6847 - loss: 1.1019 - val_accuracy: 0.8008 - val_loss: 0.8348 - learning_rate: 1.0000e-04
Epoch 23/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 238ms/step - accuracy: 0.6918 - loss: 1.0854
Epoch 23: val_accuracy improved from 0.80080 to 0.80876, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 23: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m28s@0m 252ms/step - accuracy: 0.6994 - loss: 1.0546 - val_accuracy: 0.8088 - val_loss: 0.7969 - learning_rate: 1.0000e-04
Epoch 24/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 226ms/step - accuracy: 0.7042 - loss: 1.0363
Epoch 24: val_accuracy did not improve from 0.80876
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 234ms/step - accuracy: 0.7125 - loss: 1.0168 - val_accuracy: 0.8088 - val_loss: 0.7698 - learning_rate: 1.0000e-04
Epoch 25/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 225ms/step - accuracy: 0.7034 - loss: 0.9977
Epoch 25: val_accuracy improved from 0.80876 to 0.82072, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 25: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 239ms/step - accuracy: 0.7165 - loss: 0.9958 - val_accuracy: 0.8207 - val_loss: 0.7615 - learning_rate: 1.0000e-04
Epoch 26/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 227ms/step - accuracy: 0.7313 - loss: 0.9429
Epoch 26: val_accuracy did not improve from 0.82072
[1m10/110][0m @32m-----[0m@37m@0m @1m26s@0m 236ms/step - accuracy: 0.7210 - loss: 0.9649 - val_accuracy: 0.8127 - val_loss: 0.7408 - learning_rate: 1.0000e-04
Epoch 27/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 220ms/step - accuracy: 0.7394 - loss: 0.9087
Epoch 27: val_accuracy did not improve from 0.82072
[1m10/110][0m @32m-----[0m@37m@0m @1m25s@0m 229ms/step - accuracy: 0.7267 - loss: 0.9322 - val_accuracy: 0.8207 - val_loss: 0.7150 - learning_rate: 1.0000e-04
Epoch 28/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 222ms/step - accuracy: 0.7479 - loss: 0.8818
Epoch 28: val_accuracy did not improve from 0.82072
[1m10/110][0m @32m-----[0m@37m@0m @1m25s@0m 230ms/step - accuracy: 0.7449 - loss: 0.9064 - val_accuracy: 0.8207 - val_loss: 0.7146 - learning_rate: 1.0000e-04
Epoch 29/30
[1m10/110][0m @32m-----[0m@37m@0m @1m0s@0m 222ms/step - accuracy: 0.7160 - loss: 0.9582

```
Epoch 29: val_accuracy improved from 0.82072 to 0.84064, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras

Epoch 29: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/best_efficientnetb0_asl_model.keras
1110/11000 32m 0m 37m 0m 1m26s 237ms/step - accuracy: 0.7392 - loss: 0.9047 - val_accuracy: 0.8406 - val_loss: 0.6640 - learning_rate: 1.0000e-04
Epoch 30/30
1110/11000 32m 0m 37m 0m 1m0s 229ms/step - accuracy: 0.7550 - loss: 0.8654
Epoch 30: val_accuracy did not improve from 0.84064
1110/11000 32m 0m 37m 0m 1m26s 238ms/step - accuracy: 0.7455 - loss: 0.8846 - val_accuracy: 0.8247 - val_loss: 0.6634 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.
=====
Model Training Completed Successfully
=====
```

In [15]:

```
# =====
# Step 15: Evaluate the EfficientNetB0 Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

test_loss, test_accuracy = model.evaluate(
    test_generator,
    verbose=1
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
1132/32000 32m 0m 37m 0m 1m36s 12s/step - accuracy: 0.8988 - loss: 0.5200
=====
Model Evaluation
=====
Test Loss      : 0.5200
Test Accuracy : 0.8988
=====
```

In [16]:

```
# =====
# Step 16: Generate Predictions and Classification Report
# This step generates test predictions, calculates the main
# evaluation metrics, and saves the classification report.
# =====

CLASSIFICATION_REPORT_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Classification_Reports/"
    "efficientnetb0_classification_report.txt"
)

test_generator.reset()

predictions = model.predict(
    test_generator,
    verbose=1
)

y_pred = np.argmax(
    predictions,
    axis=1
)

y_true = test_generator.classes

report = classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    digits=4,
    zero_division=0
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

print("=" * 60)
print("EfficientNetB0 Classification Report")
print("=" * 60)
print(report)

print("=" * 60)
print("Main Evaluation Metrics")
print("=" * 60)
print(f"Accuracy          : {test_accuracy:.4f}")
print(f"Weighted Precision : {weighted_precision:.4f}")
print(f"Weighted Recall    : {weighted_recall:.4f}")
print(f"Weighted F1-Score  : {weighted_f1:.4f}")
print(f"Macro F1-Score     : {macro_f1:.4f}")
print("=" * 60)

with open(
    CLASSIFICATION_REPORT_PATH,
    "w"
) as report_file:

    report_file.write(
        "EfficientNetB0 Classification Report\n"
    )

    report_file.write("=" * 60 + "\n")
    report_file.write(report)

    report_file.write("\n" + "=" * 60 + "\n")

    report_file.write(
        f"Accuracy: {test_accuracy:.4f}\n"
    )

    report_file.write(
        f"Weighted Precision: "
        f"{weighted_precision:.4f}\n"
    )

    report_file.write(
        f"Weighted Recall: "
        f"{weighted_recall:.4f}\n"
    )

    report_file.write(
        f"Weighted F1-Score: "
        f"{weighted_f1:.4f}\n"
    )

    report_file.write(
        f"Macro F1-Score: "
        f"{macro_f1:.4f}\n"
    )

print(
    "Classification Report Saved Successfully."
)
```

1m32/320[0m 32m0m0[37m0m 1m15s0m 255ms/step

EfficientNetB0 Classification Report

	precision	recall	f1-score	support
0	0.6842	0.9286	0.7879	14
1	0.8667	0.9286	0.8966	14
2	0.8667	0.9286	0.8966	14
3	1.0000	1.0000	1.0000	14
4	0.8000	0.8571	0.8276	14
5	0.8571	0.8571	0.8571	14
6	0.6667	0.7143	0.6897	14
7	0.6190	0.9286	0.7429	14
8	0.9333	1.0000	0.9655	14
9	1.0000	0.9286	0.9630	14
a	0.9286	0.9286	0.9286	14
b	1.0000	1.0000	1.0000	14
c	1.0000	1.0000	1.0000	14
d	1.0000	1.0000	1.0000	14
e	1.0000	1.0000	1.0000	14
f	1.0000	1.0000	1.0000	14
g	1.0000	1.0000	1.0000	14
h	1.0000	1.0000	1.0000	14
i	0.8235	1.0000	0.9032	14
j	0.9286	0.9286	0.9286	14
k	1.0000	1.0000	1.0000	14
l	1.0000	0.9286	0.9630	14
m	0.6500	0.9286	0.7647	14
n	1.0000	0.4286	0.6000	14
o	0.8889	0.5714	0.6957	14
p	1.0000	1.0000	1.0000	14
q	1.0000	1.0000	1.0000	14
r	1.0000	1.0000	1.0000	14
s	1.0000	0.7857	0.8800	14
t	0.7368	1.0000	0.8485	14
u	1.0000	1.0000	1.0000	14
v	1.0000	0.4286	0.6000	14
w	0.7273	0.5714	0.6400	14
x	1.0000	1.0000	1.0000	14
y	0.9333	1.0000	0.9655	14
z	1.0000	0.7857	0.8800	14
accuracy			0.8988	504
macro avg	0.9142	0.8988	0.8951	504
weighted avg	0.9142	0.8988	0.8951	504

=====
Main Evaluation Metrics
=====
Accuracy : 0.8988

```
Weighted Precision : 0.9142
Weighted Recall   : 0.8988
Weighted F1-Score  : 0.8951
Macro F1-Score     : 0.8951
=====
Classification Report Saved Successfully.
```

```
In [17]: # =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "efficientnetb0_confusion_matrix.png"
)

cm = confusion_matrix(
    y_true,
    y_pred
)

plt.figure(
    figsize=(16, 14)
)

plt.imshow(
    cm,
    cmap="Blues"
)

plt.title(
    "EfficientNetB0 Confusion Matrix",
    fontsize=18
)

plt.xlabel(
    "Predicted Label",
    fontsize=14
)

plt.ylabel(
    "True Label",
    fontsize=14
)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

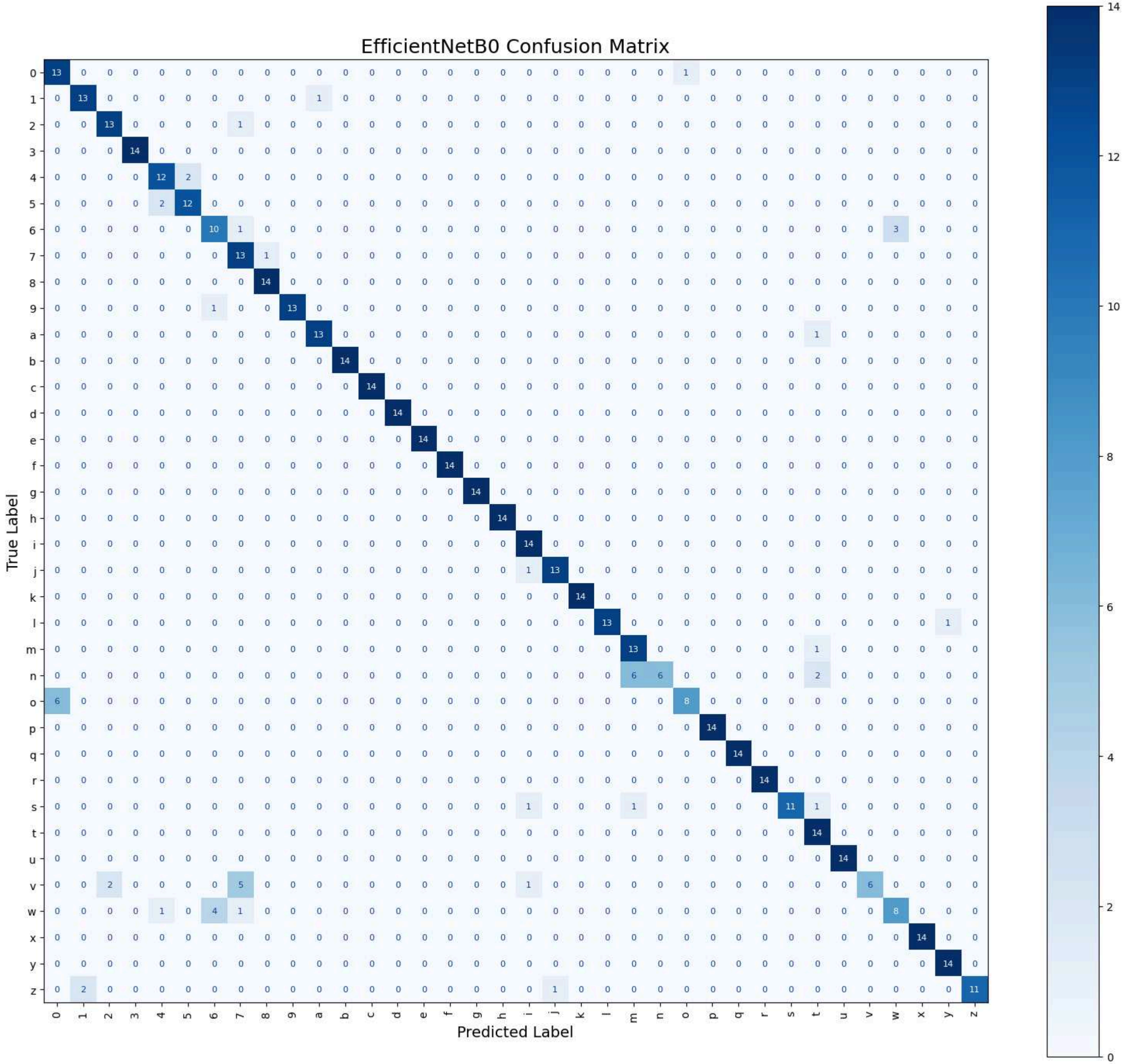
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



=====
Confusion Matrix Saved Successfully
=====
Saved Path : ./content/drive/MyDrive/ASU_Final_Project/03_Results/Confusion_Matrices/efficientnetb0_confusion_matrix.png
=====

```
In [18]: # =====
# Step 18: Plot Training History
# This step plots the Training and Validation Accuracy and
# Loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "efficientnetb0_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "efficientnetb0_loss_curve.png"
)

# Accuracy Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    history.history["accuracy"],
    label="Training Accuracy"
)

plt.plot(
    history.history["val_accuracy"],
    label="Validation Accuracy"
)

plt.title(
    "EfficientNetB0 Accuracy Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

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

plt.show()

# Loss Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    history.history["loss"],
    label="Training Loss"
)

plt.plot(
    history.history["val_loss"],
    label="Validation Loss"
)

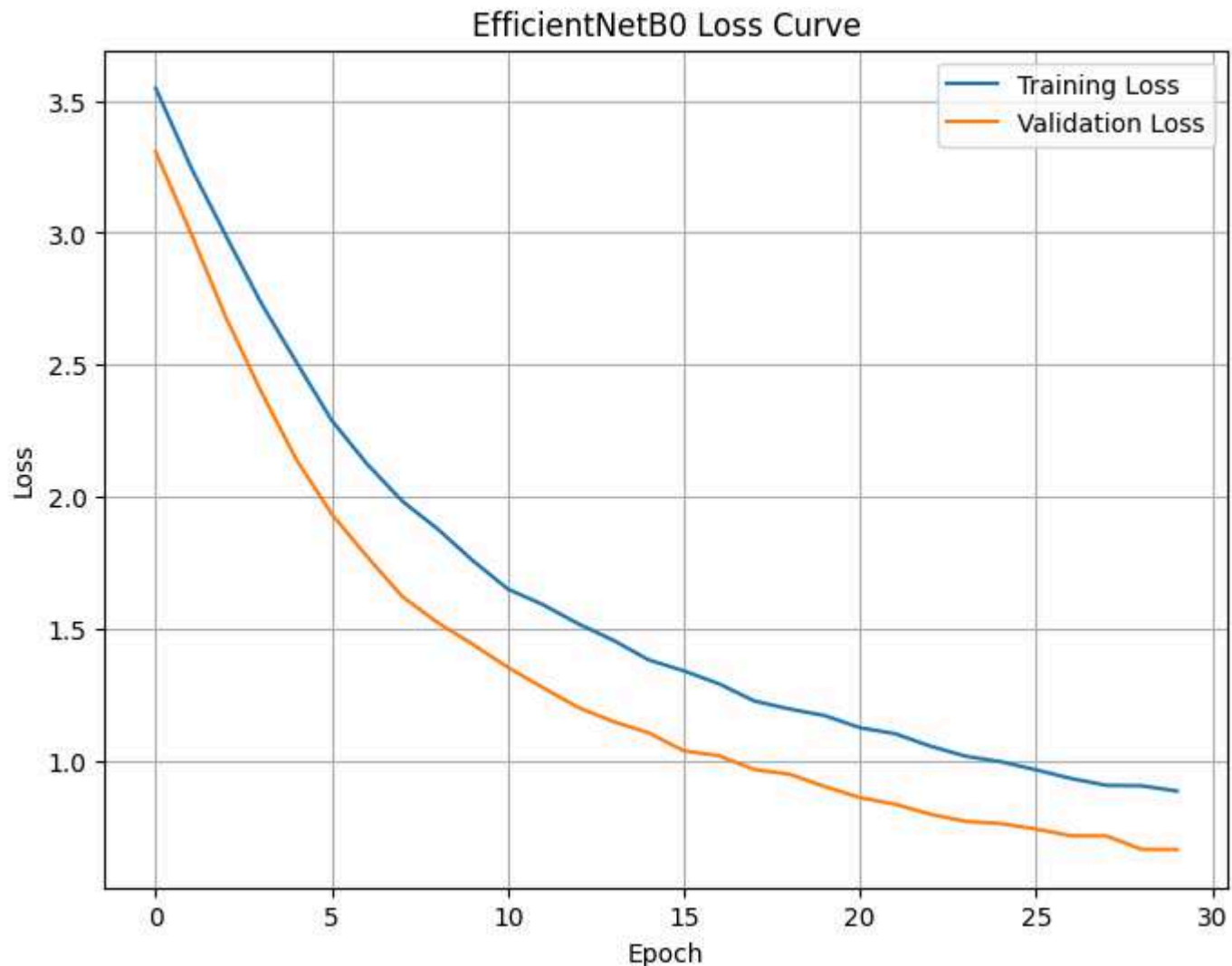
plt.title(
    "EfficientNetB0 Loss Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

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

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/efficientnetb0_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/efficientnetb0_loss_curve.png
=====
```

In [19]:

```
# =====
# Step 19: Save the Final EfficientNetB0 Model
# This step saves the trained model, class names, and final
# evaluation metrics for future comparison and deployment.
# =====

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_efficientnetb0_asl_model.keras"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "efficientnetb0_class_names.txt"
)

METRICS_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics/"
    "efficientnetb0_metrics.json"
)

model.save(
    FINAL_MODEL_PATH
)

with open(
    CLASS_NAMES_PATH,
    "w"
) as class_file:

    for class_name in CLASS_NAMES:
        class_file.write(
            f"{class_name}\n"
        )

metrics_data = {
    "model": "EfficientNetB0",
    "test_images": int(test_generator.samples),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "total_parameters": int(model.count_params())
}

with open(
    METRICS_PATH,
    "w"
) as metrics_file:

    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("Model and Results Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print(f"Metrics      : {METRICS_PATH}")
print("=" * 60)
```

```
=====
Model and Results Saved Successfully
=====
Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/final_efficientnetb0_asl_model.keras
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/EfficientNetB0/efficientnetb0_class_names.txt
Metrics      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/efficientnetb0_metrics.json
=====
```

In [20]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures EfficientNetB0 inference time on the
# complete test dataset using an A100 warm-up and repeated runs.
# =====

INERENCE_REPEATS = 3

# Warm-up run
test_generator.reset()

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

inference_times = []

for repeat in range(INERENCE_REPEATS):

    test_generator.reset()

    start_time = time.perf_counter()

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

    end_time = time.perf_counter()

    inference_times.append(
        end_time - start_time
    )

average_inference_time = float(
    np.mean(inference_times)
)

std_inference_time = float(
    np.std(inference_times)
)

average_time_per_image = (
    average_inference_time
    / test_generator.samples
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print("Model           : EfficientNetB0")
print(f"Test Images       : {test_generator.samples}")
print(f"Repeated Measurements : {INERENCE_REPEATS}")
print(f"Average Total Time   : {average_inference_time:.4f} seconds")
print(f"Standard Deviation   : {std_inference_time:.4f} seconds")
print(f"Average Time per Image : {average_time_per_image:.6f} seconds")
print("=" * 60)

# Update the saved metrics file with inference results
metrics_data.update({
    "inference_repeats": INERENCE_REPEATS,
    "average_inference_time_seconds": average_inference_time,
    "inference_standard_deviation_seconds": std_inference_time,
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "gpu": "A100"
})

with open(
    METRICS_PATH,
    "w"
) as metrics_file:

    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("Inference results added to the metrics file.")
```

```
=====
Standardized Inference Time
=====
Model           : EfficientNetB0
Test Images     : 504
Repeated Measurements : 3
Average Total Time   : 1.7662 seconds
Standard Deviation   : 0.1051 seconds
Average Time per Image : 0.003504 seconds
=====
Inference results added to the metrics file.
```

```
In [21]: # =====
# Step 21: Save Final EfficientNetB0 Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "efficientnetb0"
MODEL_DISPLAY_NAME = "EfficientNetB0"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/EfficientNetB0/"
    "final_efficientnetb0_asl_model.keras"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "efficientnetb0_metrics.json"
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpus = tf.config.list_physical_devices("GPU")

if gpus:
    gpu_name = (
        tf.config.experimental
        .get_device_details(gpus[0])
        .get("device_name", gpus[0].name)
    )
else:
    gpu_name = "CPU"

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "TensorFlow/Keras",
    "gpu": gpu_name,
    "test_images": int(test_generator.samples),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(model.count_params()),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("EfficientNetB0 Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size     : {model_size_mb:.2f} MB")
print(f"Parameters     : {model.count_params():,}")
print(f"Metrics File   : {METRICS_PATH}")
print("=" * 60)
```

```
=====
EfficientNetB0 Metrics Saved Successfully
=====
Accuracy      : 0.8988
Precision     : 0.9142
Recall        : 0.8988
F1-Score      : 0.8951
Model Size    : 20.11 MB
Parameters    : 4,386,759
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/efficientnetb0_metrics.json
=====
```

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs all additional libraries required for
# Vision Transformer training and spelling correction.
# =====

!pip -q install timm
!pip -q install textblob
```

```
In [3]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data loading,
# preprocessing, model training, evaluation, visualization,
# and result analysis.
# =====

import os
import random
import shutil
import time
import warnings

import numpy as np
import matplotlib.pyplot as plt

from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision import datasets
from torchvision import transforms

import timm

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

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("PyTorch Version :", torch.__version__)
print("TIMM Version   :", timm.__version__)
print("=" * 60)
```

```
=====
PyTorch Version : 2.11.0+cu128
TIMM Version    : 1.0.27
=====
```

```
In [4]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies GPU availability.
# =====

SEED = 42

random.seed(SEED)
np.random.seed(SEED)

torch.manual_seed(SEED)

if torch.cuda.is_available():
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)

DEVICE = torch.device(
    "cuda"
    if torch.cuda.is_available()
    else "cpu"
)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed : {SEED}")
print(f"Device      : {DEVICE}")

if torch.cuda.is_available():
    print(
        "GPU Name   :",
        torch.cuda.get_device_name(0)
    )

print("=" * 60)
```

```
=====
Project Configuration
=====
Random Seed : 42
Device      : cuda
GPU Name    : NVIDIA A100-SXM4-40GB
=====
```

```
In [5]: # =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive
# and verifies that all class folders are available.
# =====

DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Original/asl_dataset"
)

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):

    class_names = sorted([
        folder
        for folder in os.listdir(DATASET_DIR)
        if os.path.isdir(
            os.path.join(DATASET_DIR, folder)
        )
    ])

    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")
    print(f"Total Classes  : {len(class_names)}")

else:

    raise FileNotFoundError(
        f"Dataset not found:\n{DATASET_DIR}"
    )

print("=" * 60)
```

```
=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
=====
```

In [6]:

```
# =====
# Step 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_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"
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(
        os.path.join(DATASET_DIR, folder_name)
    )
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes     : {len(EXPECTED_CLASSES)}")
print(f"Available Classes    : {len(available_classes)}")
print(f"Missing Classes      : {missing_classes}")
print(f"Unexpected Classes    : {unexpected_classes}")
print("=" * 60)

if not dataset_ready:
    raise ValueError(
        "Dataset class verification failed."
    )
```

```
=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes     : 36
Available Classes    : 36
Missing Classes      : []
Unexpected Classes    : []
=====
```

In [7]:

```
# =====
# Step 7: Load Final Dataset Split
# This step loads the same fixed Train, Validation, and Test
# split used by all models for a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

TRAIN_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "train"
)

VALIDATION_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "val"
)

TEST_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "test"
)

required_paths = [
    TRAIN_DIR,
    VALIDATION_DIR,
    TEST_DIR
]

if not all(
    os.path.isdir(path)
    for path in required_paths
):
    raise FileNotFoundError(
        "The final dataset split was not found."
    )

print("=" * 60)
print("Final Dataset Split Loaded Successfully")
print("=" * 60)
print(f"Training Path       : {TRAIN_DIR}")
print(f"Validation Path     : {VALIDATION_DIR}")
print(f"Testing Path        : {TEST_DIR}")
print("=" * 60)
```

```
=====
Final Dataset Split Loaded Successfully
=====
Training Path       : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/train
Validation Path     : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/val
Testing Path        : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/test
=====
```

```
In [8]: # =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total.
# =====

VALID_EXTENSIONS = (
    ".jpg",
    ".jpeg",
    ".png",
    ".bmp",
    ".webp"
)

split_paths = {
    "Training": TRAIN_DIR,
    "Validation": VALIDATION_DIR,
    "Testing": TEST_DIR
}

grand_total = 0

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

for split_name, split_path in split_paths.items():

    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(
            split_path,
            class_name
        )

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(
                VALID_EXTENSIONS
            )
        )

    grand_total += total_images

    print(
        f"{split_name:<12}: "
        f"{total_images} images"
    )

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training    : 1760 images
Validation  : 251 images
Testing     : 504 images
-----
Total Images : 2515
=====
```

```
In [9]: # =====
# Step 9: Create Image Transformations
# This step applies Vision Transformer preprocessing and data
# augmentation to the Training, Validation, and Testing images.
# =====

IMAGE_SIZE = 224
BATCH_SIZE = 16

train_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.RandomRotation(15),

    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

validation_test_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

print("=" * 60)
print("Image Transformations Created Successfully")
print("=" * 60)
print(f"Image Size : {IMAGE_SIZE} x {IMAGE_SIZE}")
print(f"Batch Size : {BATCH_SIZE}")
print("=" * 60)
```

```
=====
Image Transformations Created Successfully
=====
Image Size : 224 x 224
Batch Size : 16
=====
```

```
In [10]: # =====
# Step 10: Load Datasets and Create DataLoaders
# This step loads the Training, Validation, and Testing
# datasets and creates DataLoaders for model training.
# =====

train_dataset = datasets.ImageFolder(
    root=TRAIN_DIR,
    transform=train_transform
)

validation_dataset = datasets.ImageFolder(
    root=VALIDATION_DIR,
    transform=validation_test_transform
)

test_dataset = datasets.ImageFolder(
    root=TEST_DIR,
    transform=validation_test_transform
)

CLASS_NAMES = train_dataset.classes
NUM_CLASSES = len(CLASS_NAMES)

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2,
    pin_memory=True
)

validation_loader = DataLoader(
    validation_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

print("=" * 60)
print("Datasets and DataLoaders Created Successfully")
print("=" * 60)
print(f"Training Samples   : {len(train_dataset)}")
print(f"Validation Samples : {len(validation_dataset)}")
print(f"Testing Samples    : {len(test_dataset)}")
print(f"Number of Classes   : {NUM_CLASSES}")
print(f"Training Batches    : {len(train_loader)}")
print(f"Validation Batches  : {len(validation_loader)}")
print(f"Testing Batches     : {len(test_loader)}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Datasets and DataLoaders Created Successfully
=====
Training Samples   : 1760
Validation Samples : 251
Testing Samples    : 504
Number of Classes   : 36
Training Batches    : 110
Validation Batches  : 16
Testing Batches     : 32
=====
Class Names
=====
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 [11]: # =====
# Step 11: Build the Vision Transformer Model
# This step loads the pretrained Vision Transformer model
# and replaces its classification head for 36 ASL classes.
# =====

MODEL_NAME = "vit_base_patch16_224"

model = timm.create_model(
    MODEL_NAME,
    pretrained=True,
    num_classes=NUM_CLASSES
)

model = model.to(DEVICE)

print("=" * 60)
print("Vision Transformer Model Created Successfully")
print("=" * 60)
print(f"Model Name       : {MODEL_NAME}")
print(f"Number of Classes : {NUM_CLASSES}")
print(f"Total Parameters  : {sum(p.numel() for p in model.parameters());}")
print(f"Device           : {DEVICE}")
print("=" * 60)
```

Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.

```
model.safetensors:   0% |          | 0.00/346M [00:00<?, ?B/s]

=====
Vision Transformer Model Created Successfully
=====
Model Name       : vit_base_patch16_224
Number of Classes : 36
Total Parameters  : 85,826,340
Device           : cuda
=====
```

In [12]:

```
# =====
# Step 12: Configure the Vision Transformer Model
# This step defines the loss function, optimizer, learning
# rate scheduler, and model saving path.
# =====

LEARNING_RATE = 0.0001

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/VisionTransformer"
)

os.makedirs(
    MODEL_DIR,
    exist_ok=True
)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_vit_asl_model.pth"
)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(
    model.parameters(),
    lr=LEARNING_RATE,
    weight_decay=0.01
)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode="min",
    factor=0.3,
    patience=4,
    min_lr=1e-6
)

print("=" * 60)
print("Training Setup Configured Successfully")
print("=" * 60)
print("Optimizer      : AdamW")
print(f"Learning Rate   : {LEARNING_RATE}")
print("Loss Function   : Cross Entropy Loss")
print("Weight Decay    : 0.01")
print(f"Best Model Path: {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Setup Configured Successfully
=====
Optimizer      : AdamW
Learning Rate   : 0.0001
Loss Function   : Cross Entropy Loss
Weight Decay    : 0.01
Best Model Path: /content/drive/MyDrive/ASL_Final_Project/02_Models/VisionTransformer/best_vit_asl_model.pth
=====
```

```
In [13]: # =====
# Step 13: Create Training and Evaluation Functions
# This step creates the functions used to train and evaluate
# the Vision Transformer model during each epoch.
# =====

def train_one_epoch(
    model,
    data_loader,
    criterion,
    optimizer,
    device
):
    model.train()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for images, labels in data_loader:

        images = images.to(
            device,
            non_blocking=True
        )

        labels = labels.to(
            device,
            non_blocking=True
        )

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(
            outputs,
            labels
        )

        loss.backward()

        optimizer.step()

        running_loss += (
            loss.item() * images.size(0)
        )

        predicted_labels = outputs.argmax(
            dim=1
        )

        correct_predictions += (
            predicted_labels == labels
        ).sum().item()

        total_samples += labels.size(0)

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return epoch_loss, epoch_accuracy

def evaluate_model(
    model,
    data_loader,
    criterion,
    device
):
    model.eval()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    true_labels = []
    predicted_labels = []

    with torch.no_grad():

        for images, labels in data_loader:

            images = images.to(
                device,
                non_blocking=True
            )

            labels = labels.to(
                device,
                non_blocking=True
            )

            outputs = model(images)

            loss = criterion(
                outputs,
                labels
            )

            running_loss += (
                loss.item() * images.size(0)
            )

            predictions = outputs.argmax(
                dim=1
            )

            correct_predictions += (
                predictions == labels
            ).sum().item()

            total_samples += labels.size(0)

            true_labels.extend(
                labels.cpu().numpy()
            )

            predicted_labels.extend(
                predictions.cpu().numpy()
            )

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return (
        epoch_loss,
        epoch_accuracy,
        true_labels,
        predicted_labels
    )

print("=" * 60)
print("Training and Evaluation Functions Created Successfully")
print("=" * 60)
```

```
=====
Training and Evaluation Functions Created Successfully
=====
```

In [14]:

```
# =====
# Step 14: Train the Vision Transformer Model
# This step trains the model, monitors validation performance,
# applies Early Stopping, and saves the best model.
# =====

EPOCHS = 30
EARLY_STOPPING_PATIENCE = 7

train_losses = []
validation_losses = []

train_accuracies = []
validation_accuracies = []

best_validation_accuracy = 0.0
best_validation_loss = float("inf")

early_stopping_counter = 0

for epoch in range(EPOCHS):

    train_loss, train_accuracy = train_one_epoch(
        model,
        train_loader,
        criterion,
        optimizer,
        DEVICE
    )

    (
        validation_loss,
        validation_accuracy,
        _
    ) = evaluate_model(
        model,
        validation_loader,
        criterion,
        DEVICE
    )

    scheduler.step(
        validation_loss
    )

    train_losses.append(
        train_loss
    )

    validation_losses.append(
        validation_loss
    )

    train_accuracies.append(
        train_accuracy
    )

    validation_accuracies.append(
        validation_accuracy
    )

    current_learning_rate = (
        optimizer.param_groups[0]["lr"]
    )

    print("=" * 60)
    print(f"Epoch [{epoch + 1}/{EPOCHS}]")
    print("=" * 60)
    print(f"Training Loss      : {train_loss:.4f}")
    print(f"Training Accuracy    : {train_accuracy:.4f}")
    print(f"Validation Loss       : {validation_loss:.4f}")
    print(f"Validation Accuracy   : {validation_accuracy:.4f}")
    print(f"Learning Rate        : {current_learning_rate:.8f}")

    if validation_accuracy > best_validation_accuracy:

        best_validation_accuracy = (
            validation_accuracy
        )

        torch.save(
            model.state_dict(),
            BEST_MODEL_PATH
        )

        print("Best model saved successfully.")

    if validation_loss < best_validation_loss:

        best_validation_loss = (
            validation_loss
        )

        early_stopping_counter = 0

    else:

        early_stopping_counter += 1

        print(
            "Early Stopping Counter : "
            f"{early_stopping_counter}/"
            f"{EARLY_STOPPING_PATIENCE}"
        )

    if (
        early_stopping_counter
        >= EARLY_STOPPING_PATIENCE
    ):

        print("=" * 60)
        print("Early Stopping Activated")
        print("=" * 60)

        break

model.load_state_dict(
    torch.load(
        BEST_MODEL_PATH,
        map_location=DEVICE
    )
)

model = model.to(DEVICE)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
print(
    f"Best Validation Accuracy : "
    f"{best_validation_accuracy:.4f}"
)
print("Best model weights restored successfully.")
print("=" * 60)
```

```
=====
Epoch [1/30]
=====
Training Loss      : 3.4832
Training Accuracy  : 0.0716
Validation Loss    : 2.6513
Validation Accuracy : 0.2829
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [2/30]
=====
Training Loss      : 1.2265
Training Accuracy  : 0.5881
Validation Loss    : 0.5569
Validation Accuracy : 0.7968
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [3/30]
=====
Training Loss      : 0.4696
Training Accuracy  : 0.8426
Validation Loss    : 0.3135
Validation Accuracy : 0.9044
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [4/30]
=====
Training Loss      : 0.2991
Training Accuracy  : 0.8989
Validation Loss    : 0.2440
Validation Accuracy : 0.9163
```

```
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [5/30]
=====
Training Loss      : 0.1811
Training Accuracy   : 0.9364
Validation Loss     : 0.1838
Validation Accuracy : 0.9203
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [6/30]
=====
Training Loss      : 0.1534
Training Accuracy   : 0.9517
Validation Loss     : 0.1842
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
Best model saved successfully.
Early Stopping Counter : 1/7
=====
Epoch [7/30]
=====
Training Loss      : 0.1719
Training Accuracy   : 0.9443
Validation Loss     : 0.3328
Validation Accuracy : 0.9203
Learning Rate      : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [8/30]
=====
Training Loss      : 0.1679
Training Accuracy   : 0.9443
Validation Loss     : 0.1623
Validation Accuracy : 0.9522
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [9/30]
=====
Training Loss      : 0.1065
Training Accuracy   : 0.9693
Validation Loss     : 0.1709
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [10/30]
=====
Training Loss      : 0.1067
Training Accuracy   : 0.9608
Validation Loss     : 0.2084
Validation Accuracy : 0.9402
Learning Rate      : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [11/30]
=====
Training Loss      : 0.0861
Training Accuracy   : 0.9699
Validation Loss     : 0.1340
Validation Accuracy : 0.9522
Learning Rate      : 0.00010000
=====
Epoch [12/30]
=====
Training Loss      : 0.0894
Training Accuracy   : 0.9744
Validation Loss     : 0.0986
Validation Accuracy : 0.9602
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [13/30]
=====
Training Loss      : 0.0357
Training Accuracy   : 0.9864
Validation Loss     : 0.1351
Validation Accuracy : 0.9522
Learning Rate      : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [14/30]
=====
Training Loss      : 0.1299
Training Accuracy   : 0.9563
Validation Loss     : 0.1437
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [15/30]
=====
Training Loss      : 0.1426
Training Accuracy   : 0.9545
Validation Loss     : 0.1770
Validation Accuracy : 0.9562
Learning Rate      : 0.00010000
Early Stopping Counter : 3/7
=====
Epoch [16/30]
=====
Training Loss      : 0.0861
Training Accuracy   : 0.9790
Validation Loss     : 0.1600
Validation Accuracy : 0.9562
Learning Rate      : 0.00010000
Early Stopping Counter : 4/7
=====
Epoch [17/30]
=====
Training Loss      : 0.0571
Training Accuracy   : 0.9824
Validation Loss     : 0.2132
Validation Accuracy : 0.9323
Learning Rate      : 0.00003000
Early Stopping Counter : 5/7
=====
Epoch [18/30]
=====
Training Loss      : 0.0158
Training Accuracy   : 0.9949
Validation Loss     : 0.1022
Validation Accuracy : 0.9681
Learning Rate      : 0.00003000
Best model saved successfully.
Early Stopping Counter : 6/7
=====
Epoch [19/30]
=====
Training Loss      : 0.0118
Training Accuracy   : 0.9955
Validation Loss     : 0.0721
Validation Accuracy : 0.9801
Learning Rate      : 0.00003000
Best model saved successfully.
=====
Epoch [20/30]
=====
Training Loss      : 0.0060
Training Accuracy   : 0.9989
Validation Loss     : 0.0596
Validation Accuracy : 0.9841
Learning Rate      : 0.00003000
Best model saved successfully.
=====
Epoch [21/30]
=====
Training Loss      : 0.0011
Training Accuracy   : 1.0000
Validation Loss     : 0.0548
Validation Accuracy : 0.9801
Learning Rate      : 0.00003000
=====
Epoch [22/30]
=====
Training Loss      : 0.0021
Training Accuracy   : 0.9994
Validation Loss     : 0.0783
Validation Accuracy : 0.9841
Learning Rate      : 0.00003000
Early Stopping Counter : 1/7
=====
Epoch [23/30]
=====
Training Loss      : 0.0032
Training Accuracy   : 0.9989
Validation Loss     : 0.0878
```

```
Validation Accuracy : 0.9841
Learning Rate       : 0.00003000
Early Stopping Counter : 2/7
=====
Epoch [24/30]
=====
Training Loss       : 0.0066
Training Accuracy   : 0.9977
Validation Loss     : 0.0570
Validation Accuracy : 0.9880
Learning Rate       : 0.00003000
Best model saved successfully.
Early Stopping Counter : 3/7
=====
Epoch [25/30]
=====
Training Loss       : 0.0028
Training Accuracy   : 0.9994
Validation Loss     : 0.0661
Validation Accuracy : 0.9880
Learning Rate       : 0.00003000
Early Stopping Counter : 4/7
=====
Epoch [26/30]
=====
Training Loss       : 0.0075
Training Accuracy   : 0.9977
Validation Loss     : 0.0942
Validation Accuracy : 0.9681
Learning Rate       : 0.00000900
Early Stopping Counter : 5/7
=====
Epoch [27/30]
=====
Training Loss       : 0.0024
Training Accuracy   : 0.9994
Validation Loss     : 0.0620
Validation Accuracy : 0.9841
Learning Rate       : 0.00000900
Early Stopping Counter : 6/7
=====
Epoch [28/30]
=====
Training Loss       : 0.0006
Training Accuracy   : 1.0000
Validation Loss     : 0.0601
Validation Accuracy : 0.9841
Learning Rate       : 0.00000900
Early Stopping Counter : 7/7
=====
Early Stopping Activated
=====
Model Training Completed Successfully
=====
Best Validation Accuracy : 0.9880
Best model weights restored successfully.
=====
```

```
In [15]: # =====
# Step 15: Evaluate the Vision Transformer Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

(
    test_loss,
    test_accuracy,
    y_true,
    y_pred
) = evaluate_model(
    model,
    test_loader,
    criterion,
    DEVICE
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
=====
Model Evaluation
=====
Test Loss      : 0.0427
Test Accuracy : 0.9881
=====
```

```
In [16]: # =====
# Step 16: Generate Predictions and Classification Report
# This step displays the precision, recall, F1-score,
# and support values for all ASL classes.
# =====

report = classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    digits=4,
    zero_division=0
)

print("=" * 60)
print("Vision Transformer Classification Report")
print("=" * 60)
print(report)
print("=" * 60)
```

```
=====
Vision Transformer Classification Report
=====
```

	precision	recall	f1-score	support
0	0.8462	0.7857	0.8148	14
1	1.0000	1.0000	1.0000	14
2	1.0000	1.0000	1.0000	14
3	1.0000	1.0000	1.0000	14
4	1.0000	1.0000	1.0000	14
5	1.0000	1.0000	1.0000	14
6	1.0000	0.9286	0.9630	14
7	1.0000	1.0000	1.0000	14
8	1.0000	1.0000	1.0000	14
9	1.0000	1.0000	1.0000	14
a	1.0000	1.0000	1.0000	14
b	1.0000	1.0000	1.0000	14
c	1.0000	1.0000	1.0000	14
d	1.0000	1.0000	1.0000	14
e	1.0000	1.0000	1.0000	14
f	1.0000	1.0000	1.0000	14
g	1.0000	1.0000	1.0000	14
h	1.0000	1.0000	1.0000	14
i	1.0000	1.0000	1.0000	14
j	1.0000	1.0000	1.0000	14
k	1.0000	1.0000	1.0000	14
l	1.0000	1.0000	1.0000	14
m	1.0000	1.0000	1.0000	14
n	1.0000	1.0000	1.0000	14
o	0.8000	0.8571	0.8276	14
p	1.0000	1.0000	1.0000	14
q	1.0000	1.0000	1.0000	14
r	1.0000	1.0000	1.0000	14
s	1.0000	1.0000	1.0000	14
t	1.0000	1.0000	1.0000	14
u	1.0000	1.0000	1.0000	14
v	1.0000	1.0000	1.0000	14
w	0.9333	1.0000	0.9655	14
x	1.0000	1.0000	1.0000	14
y	1.0000	1.0000	1.0000	14
z	1.0000	1.0000	1.0000	14
accuracy			0.9881	504
macro avg	0.9883	0.9881	0.9881	504
weighted avg	0.9883	0.9881	0.9881	504

```
=====
```

```
In [17]: # =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "vit_confusion_matrix.png"
)

cm = confusion_matrix(
    y_true,
    y_pred
)

plt.figure(
    figsize=(16, 14)
)

plt.imshow(
    cm,
    cmap="Blues"
)

plt.title(
    "Vision Transformer Confusion Matrix",
    fontsize=18
)

plt.xlabel(
    "Predicted Label",
    fontsize=14
)

plt.ylabel(
    "True Label",
    fontsize=14
)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

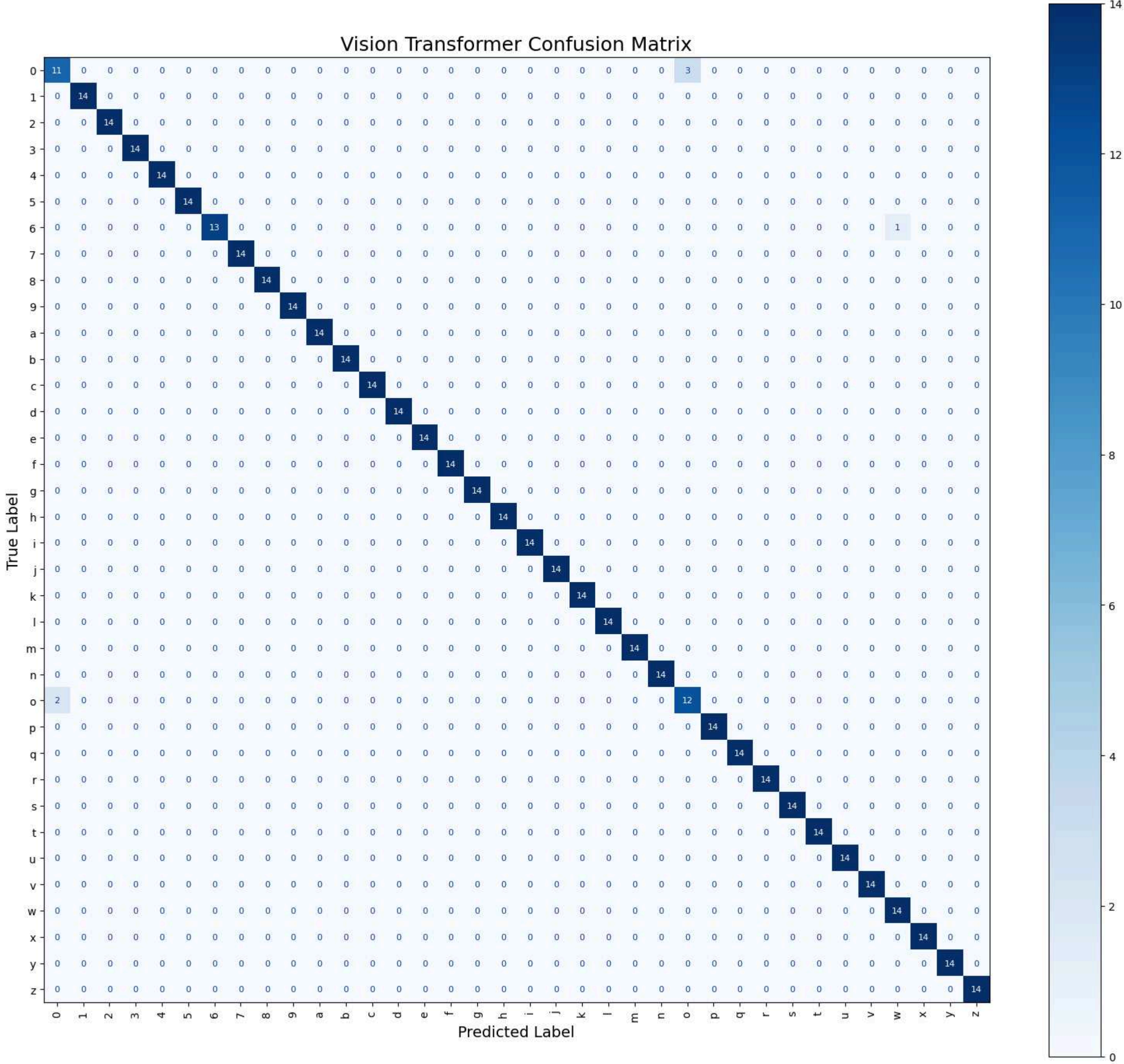
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



=====
Confusion Matrix Saved Successfully
=====
Saved Path : ./content/drive/MyDrive/ASU_Final_Project/03_Results/Confusion_Matrices/vit_confusion_matrix.png
=====

```
In [18]: # =====
# Step 18: Plot Training History
# This step plots the Training and Validation Accuracy and
# Loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "vit_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "vit_loss_curve.png"
)

# Accuracy Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    train_accuracies,
    label="Training Accuracy"
)

plt.plot(
    validation_accuracies,
    label="Validation Accuracy"
)

plt.title(
    "Vision Transformer Accuracy Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

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

plt.show()

# Loss Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    train_losses,
    label="Training Loss"
)

plt.plot(
    validation_losses,
    label="Validation Loss"
)

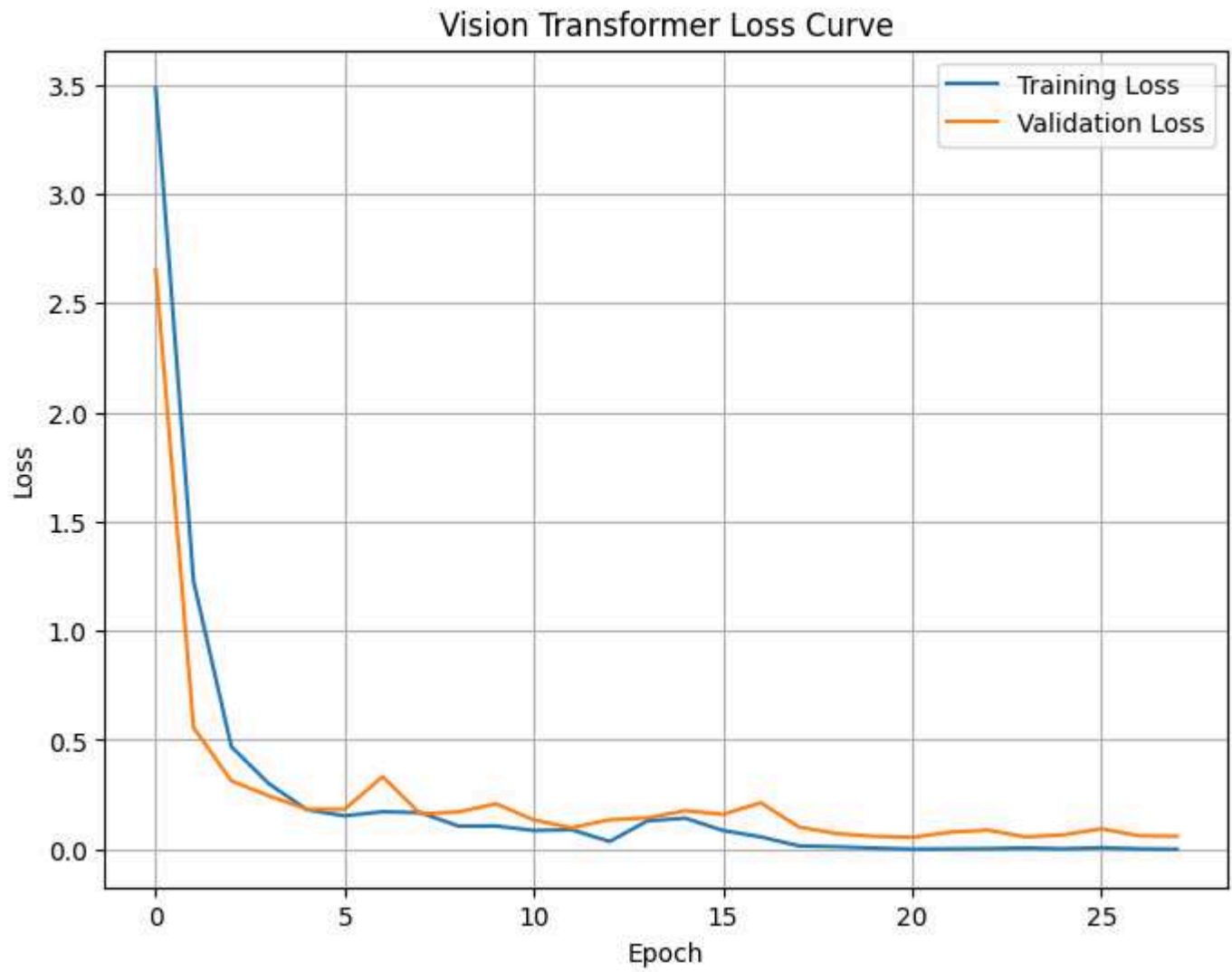
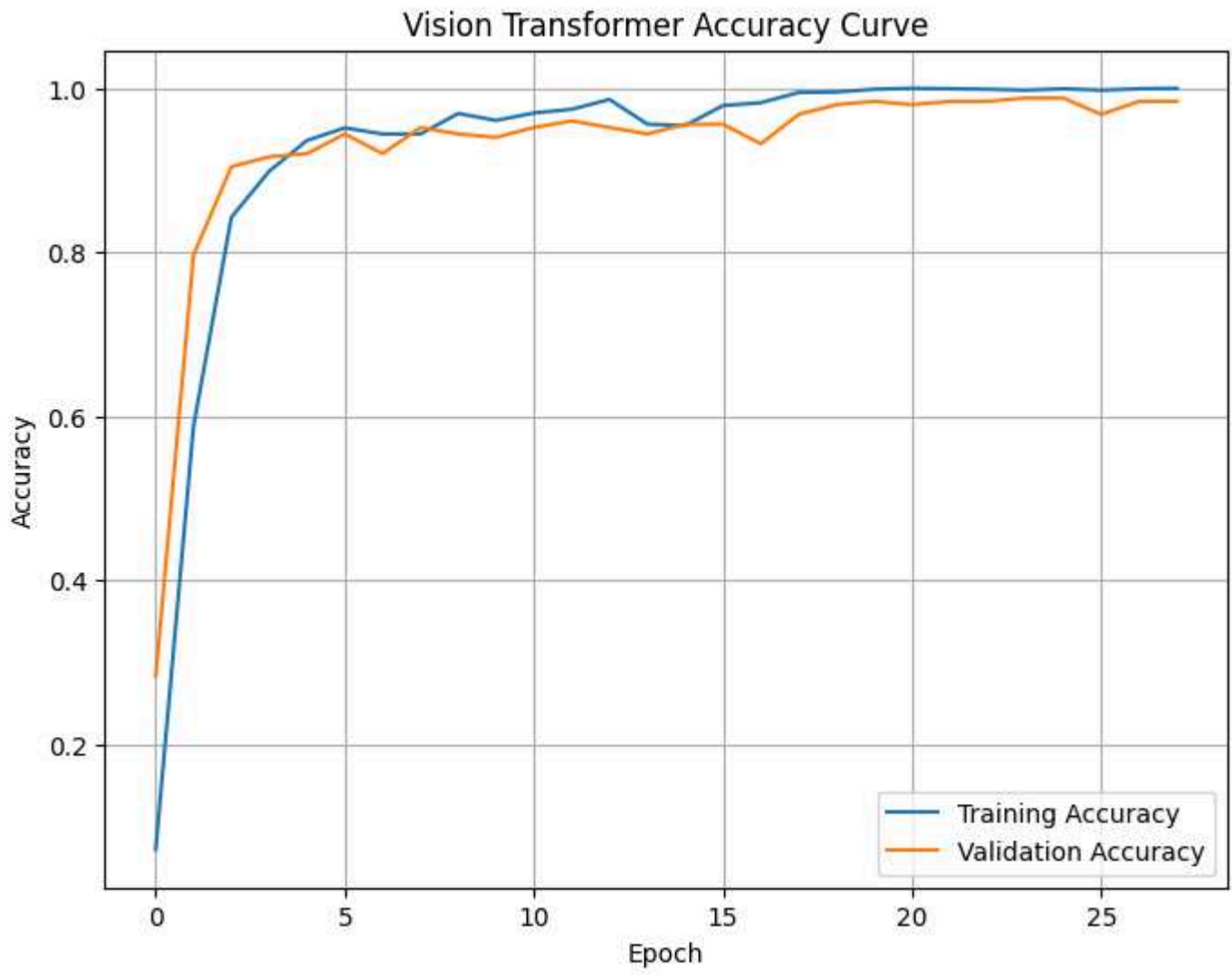
plt.title(
    "Vision Transformer Loss Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

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

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/vit_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/vit_loss_curve.png
=====
```

In [19]:

```
# =====
# Step 19: Save the Final Vision Transformer Model
# This step saves the trained Vision Transformer model
# and the class names for future inference and deployment.
# =====

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_vit_asl_model.pth"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "vit_class_names.txt"
)

torch.save(
    model.state_dict(),
    FINAL_MODEL_PATH
)

with open(
    CLASS_NAMES_PATH,
    "w"
) as file:

    for class_name in CLASS_NAMES:
        file.write(
            f"{class_name}\n"
        )

print("=" * 60)
print("Model Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print("=" * 60)

=====
Model Saved Successfully
=====

Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/VisionTransformer/final_vit_asl_model.pth
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/VisionTransformer/vit_class_names.txt
=====
```

In [20]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures Vision Transformer inference time on the
# complete test dataset using a warm-up run and repeated tests.
# =====

INFERENCE_REPEATS = 3

model.eval()

# Warm-up run
with torch.no_grad():

    for images, _ in test_loader:

        images = images.to(
            DEVICE,
            non_blocking=True
        )

        _ = model(images)

    break

if torch.cuda.is_available():
    torch.cuda.synchronize()

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    start_time = time.perf_counter()

    with torch.no_grad():

        for images, _ in test_loader:

            images = images.to(
                DEVICE,
                non_blocking=True
            )

            _ = model(images)

        if torch.cuda.is_available():
            torch.cuda.synchronize()

        end_time = time.perf_counter()

        inference_times.append(
            end_time - start_time
        )

average_inference_time = np.mean(
    inference_times
)

std_inference_time = np.std(
    inference_times
)

average_time_per_image = (
    average_inference_time
    / len(test_dataset)
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print("Model           : Vision Transformer")
print(f"Test Images       : {len(test_dataset)}")
print(f"Repeated Measurements : {INFERENCE_REPEATS}")
print(f"Average Total Time   : {average_inference_time:.4f} seconds")
print(f"Standard Deviation   : {std_inference_time:.4f} seconds")
print(f"Average Time per Image : {average_time_per_image:.6f} seconds")
print("=" * 60)

=====
Standardized Inference Time
=====
Model           : Vision Transformer
Test Images     : 504
Repeated Measurements : 3
Average Total Time   : 3.8516 seconds
Standard Deviation   : 3.1241 seconds
Average Time per Image : 0.007642 seconds
=====
```

In [21]:

```
# =====
# Step 21: Save Final Vision Transformer Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "vit"
MODEL_DISPLAY_NAME = "Vision Transformer"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/VisionTransformer/final_vit_asl_model.pth"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "vit_metrics.json"
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpu_name = (
    torch.cuda.get_device_name(0)
    if torch.cuda.is_available()
    else "CPU"
)

total_parameters = sum(
    parameter.numel()
    for parameter in model.parameters()
)

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "PyTorch",
    "gpu": gpu_name,
    "test_images": int(len(test_dataset)),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(total_parameters),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("Vision Transformer Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size     : {model_size_mb:.2f} MB")
print(f"Parameters     : {total_parameters:,}")
print(f"Metrics File   : {METRICS_PATH}")
print("=" * 60)
```

```
=====
Vision Transformer Metrics Saved Successfully
=====
Accuracy      : 0.9881
Precision     : 0.9883
Recall        : 0.9881
F1-Score      : 0.9881
Model Size    : 327.46 MB
Parameters    : 85,826,340
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/vit_metrics.json
=====
```

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs the additional libraries required for
# Swin Transformer training and spelling correction.
# =====

!pip -q install timm
!pip -q install textblob
```

```
In [3]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data loading,
# preprocessing, model training, evaluation, visualization,
# and result analysis.
# =====

import os
import random
import time
import warnings

import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim

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

import timm

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

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("PyTorch Version :", torch.__version__)
print("TIMM Version    :", timm.__version__)
print("=" * 60)
```

```
=====
PyTorch Version : 2.11.0+cu128
TIMM Version    : 1.0.27
=====
```

```
In [4]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies GPU availability.
# =====

SEED = 42

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)

if torch.cuda.is_available():
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)

DEVICE = torch.device(
    "cuda" if torch.cuda.is_available() else "cpu"
)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed : {SEED}")
print(f"Device      : {DEVICE}")

if torch.cuda.is_available():
    print("GPU Name      :", torch.cuda.get_device_name(0))
else:
    raise RuntimeError(
        "GPU was not detected. Enable A100 GPU before training."
    )

print("=" * 60)
```

```
=====
Project Configuration
=====
Random Seed : 42
Device      : cuda
GPU Name     : NVIDIA A100-SXM4-40GB
=====
```

```
In [5]: # =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive
# and verifies that all class folders are available.
# =====

DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Original/asl_dataset"
)

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):

    class_names = sorted([
        folder_name
        for folder_name in os.listdir(DATASET_DIR)
        if os.path.isdir(
            os.path.join(DATASET_DIR, folder_name)
        )
    ])

    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")
    print(f"Total Classes  : {len(class_names)}")

else:
    raise FileNotFoundError(
        f"Dataset not found:\n{DATASET_DIR}"
    )

print("=" * 60)
```

```
=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
=====
```

```
In [6]: # =====
# Step 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_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"
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(
        os.path.join(DATASET_DIR, folder_name)
    )
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes     : {len(EXPECTED_CLASSES)}")
print(f"Available Classes    : {len(available_classes)}")
print(f"Missing Classes      : {missing_classes}")
print(f"Unexpected Classes   : {unexpected_classes}")
print("=" * 60)

if not dataset_ready:
    raise ValueError(
        "Dataset class verification failed."
    )
```

```
=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes     : 36
Available Classes    : 36
Missing Classes      : []
Unexpected Classes   : []
=====
```

```
In [7]: # =====
# Step 7: Load Final Dataset Split
# This step loads the same fixed Train, Validation, and Test
# split used by all models for a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

TRAIN_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "train"
)

VALIDATION_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "val"
)

TEST_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "test"
)

required_paths = [
    TRAIN_DIR,
    VALIDATION_DIR,
    TEST_DIR
]

if not all(
    os.path.isdir(path)
    for path in required_paths
):
    raise FileNotFoundError(
        "The final dataset split was not found."
    )

print("=" * 60)
print("Final Dataset Split Loaded Successfully")
print("=" * 60)
print(f"Training Path      : {TRAIN_DIR}")
print(f"Validation Path    : {VALIDATION_DIR}")
print(f"Testing Path       : {TEST_DIR}")
print("=" * 60)
```

```
=====
Final Dataset Split Loaded Successfully
=====
Training Path      : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/train
Validation Path    : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/val
Testing Path       : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/test
=====
```

```
In [8]: # =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total.
# =====

VALID_EXTENSIONS = (
    ".jpg",
    ".jpeg",
    ".png",
    ".bmp",
    ".webp"
)

split_paths = {
    "Training": TRAIN_DIR,
    "Validation": VALIDATION_DIR,
    "Testing": TEST_DIR
}

grand_total = 0

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

for split_name, split_path in split_paths.items():

    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(
            split_path,
            class_name
        )

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(
                VALID_EXTENSIONS
            )
        )

    grand_total += total_images

    print(
        f"{split_name:<12}: "
        f"{total_images} images"
    )

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training    : 1760 images
Validation  : 251 images
Testing     : 504 images
-----
Total Images : 2515
=====
```

```
In [9]: # =====
# Step 9: Create Image Transformations
# This step applies Swin Transformer preprocessing and data
# augmentation to the Training, Validation, and Testing images.
# =====

IMAGE_SIZE = 224
BATCH_SIZE = 16

train_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.RandomRotation(15),

    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

validation_test_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

print("=" * 60)
print("Image Transformations Created Successfully")
print("=" * 60)
print(f"Image Size : {IMAGE_SIZE} x {IMAGE_SIZE}")
print(f"Batch Size : {BATCH_SIZE}")
print("=" * 60)
```

```
=====
Image Transformations Created Successfully
=====
Image Size : 224 x 224
Batch Size : 16
=====
```

```
In [10]: # =====
# Step 10: Load Datasets and Create DataLoaders
# This step loads the Training, Validation, and Testing
# datasets and creates DataLoaders for model training.
# =====

train_dataset = datasets.ImageFolder(
    root=TRAIN_DIR,
    transform=train_transform
)

validation_dataset = datasets.ImageFolder(
    root=VALIDATION_DIR,
    transform=validation_test_transform
)

test_dataset = datasets.ImageFolder(
    root=TEST_DIR,
    transform=validation_test_transform
)

CLASS_NAMES = train_dataset.classes
NUM_CLASSES = len(CLASS_NAMES)

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2,
    pin_memory=True
)

validation_loader = DataLoader(
    validation_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

print("=" * 60)
print("Datasets and DataLoaders Created Successfully")
print("=" * 60)
print(f"Training Samples   : {len(train_dataset)}")
print(f"Validation Samples : {len(validation_dataset)}")
print(f"Testing Samples    : {len(test_dataset)}")
print(f"Number of Classes  : {NUM_CLASSES}")
print(f"Training Batches   : {len(train_loader)}")
print(f"Validation Batches : {len(validation_loader)}")
print(f"Testing Batches    : {len(test_loader)}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Datasets and DataLoaders Created Successfully
=====
Training Samples   : 1760
Validation Samples : 251
Testing Samples    : 504
Number of Classes  : 36
Training Batches   : 110
Validation Batches : 16
Testing Batches    : 32
=====
Class Names
=====
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 [11]: # =====
# Step 11: Build the Swin Transformer Model
# This step loads the pretrained Swin Transformer model
# and replaces its classification head for 36 ASL classes.
# =====

MODEL_NAME = "swin_tiny_patch4_window7_224"

model = timm.create_model(
    MODEL_NAME,
    pretrained=True,
    num_classes=NUM_CLASSES
)

model = model.to(DEVICE)

print("=" * 60)
print("Swin Transformer Model Created Successfully")
print("=" * 60)
print(f"Model Name       : {MODEL_NAME}")
print(f"Number of Classes : {NUM_CLASSES}")
print(
    f"Total Parameters : "
    f"{sum(p.numel() for p in model.parameters()):,}"
)
print(f"Device           : {DEVICE}")
print("=" * 60)
```

Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.

```
model.safetensors:   0%|          | 0.00/114M [00:00<?, ?B/s]

=====
Swin Transformer Model Created Successfully
=====
Model Name       : swin_tiny_patch4_window7_224
Number of Classes : 36
Total Parameters : 27,547,038
Device           : cuda
=====
```

In [12]:

```
# =====
# Step 12: Configure the Swin Transformer Model
# This step defines the loss function, optimizer, learning
# rate scheduler, and model saving path.
# =====

LEARNING_RATE = 0.0001

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/SwinTransformer"
)

os.makedirs(
    MODEL_DIR,
    exist_ok=True
)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_swin_asl_model.pth"
)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(
    model.parameters(),
    lr=LEARNING_RATE,
    weight_decay=0.01
)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode="min",
    factor=0.3,
    patience=4,
    min_lr=1e-6
)

print("=" * 60)
print("Training Setup Configured Successfully")
print("=" * 60)
print("Optimizer      : AdamW")
print(f"Learning Rate   : {LEARNING_RATE}")
print("Loss Function   : Cross Entropy Loss")
print("Weight Decay    : 0.01")
print(f"Best Model Path : {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Setup Configured Successfully
=====
Optimizer      : AdamW
Learning Rate   : 0.0001
Loss Function   : Cross Entropy Loss
Weight Decay    : 0.01
Best Model Path : /content/drive/MyDrive/ASL_Final_Project/02_Models/SwinTransformer/best_swin_asl_model.pth
=====
```

```
In [13]: # =====
# Step 13: Create Training and Evaluation Functions
# This step creates the functions used to train and evaluate
# the Swin Transformer model during each epoch.
# =====

def train_one_epoch(
    model,
    data_loader,
    criterion,
    optimizer,
    device
):
    model.train()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for images, labels in data_loader:

        images = images.to(
            device,
            non_blocking=True
        )

        labels = labels.to(
            device,
            non_blocking=True
        )

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(
            outputs,
            labels
        )

        loss.backward()

        optimizer.step()

        running_loss += (
            loss.item() * images.size(0)
        )

        predicted_labels = outputs.argmax(
            dim=1
        )

        correct_predictions += (
            predicted_labels == labels
        ).sum().item()

        total_samples += labels.size(0)

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return epoch_loss, epoch_accuracy

def evaluate_model(
    model,
    data_loader,
    criterion,
    device
):
    model.eval()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    true_labels = []
    predicted_labels = []

    with torch.no_grad():

        for images, labels in data_loader:

            images = images.to(
                device,
                non_blocking=True
            )

            labels = labels.to(
                device,
                non_blocking=True
            )

            outputs = model(images)

            loss = criterion(
                outputs,
                labels
            )

            running_loss += (
                loss.item() * images.size(0)
            )

            predictions = outputs.argmax(
                dim=1
            )

            correct_predictions += (
                predictions == labels
            ).sum().item()

            total_samples += labels.size(0)

            true_labels.extend(
                labels.cpu().numpy()
            )

            predicted_labels.extend(
                predictions.cpu().numpy()
            )

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return (
        epoch_loss,
        epoch_accuracy,
        true_labels,
        predicted_labels
    )

print("=" * 60)
print("Training and Evaluation Functions Created Successfully")
print("=" * 60)
```

```
=====
Training and Evaluation Functions Created Successfully
=====
```

In [14]:

```
# =====
# Step 14: Train the Swin Transformer Model
# This step trains the model, monitors validation performance,
# applies Early Stopping, and saves the best model.
# =====

EPOCHS = 30
EARLY_STOPPING_PATIENCE = 7

train_losses = []
validation_losses = []

train_accuracies = []
validation_accuracies = []

best_validation_accuracy = 0.0
best_validation_loss = float("inf")

early_stopping_counter = 0

for epoch in range(EPOCHS):

    train_loss, train_accuracy = train_one_epoch(
        model,
        train_loader,
        criterion,
        optimizer,
        DEVICE
    )

    (
        validation_loss,
        validation_accuracy,
        _
    ) = evaluate_model(
        model,
        validation_loader,
        criterion,
        DEVICE
    )

    scheduler.step(
        validation_loss
    )

    train_losses.append(train_loss)
    validation_losses.append(validation_loss)

    train_accuracies.append(train_accuracy)
    validation_accuracies.append(validation_accuracy)

    current_learning_rate = (
        optimizer.param_groups[0]["lr"]
    )

    print("=" * 60)
    print(f"Epoch [{epoch + 1}/{EPOCHS}]")
    print("=" * 60)
    print(f"Training Loss      : {train_loss:.4f}")
    print(f"Training Accuracy     : {train_accuracy:.4f}")
    print(f"Validation Loss        : {validation_loss:.4f}")
    print(f"Validation Accuracy     : {validation_accuracy:.4f}")
    print(f"Learning Rate          : {current_learning_rate:.8f}")

    if validation_accuracy > best_validation_accuracy:

        best_validation_accuracy = (
            validation_accuracy
        )

        torch.save(
            model.state_dict(),
            BEST_MODEL_PATH
        )

        print("Best model saved successfully.")

    if validation_loss < best_validation_loss:

        best_validation_loss = (
            validation_loss
        )

        early_stopping_counter = 0

    else:

        early_stopping_counter += 1

        print(
            "Early Stopping Counter : "
            f"{early_stopping_counter}/"
            f"{EARLY_STOPPING_PATIENCE}"
        )

    if (
        early_stopping_counter
        >= EARLY_STOPPING_PATIENCE
    ):

        print("=" * 60)
        print("Early Stopping Activated")
        print("=" * 60)

        break

model.load_state_dict(
    torch.load(
        BEST_MODEL_PATH,
        map_location=DEVICE
    )
)

model = model.to(DEVICE)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
print(
    f"Best Validation Accuracy : "
    f"{best_validation_accuracy:.4f}"
)
print("Best model weights restored successfully.")
print("=" * 60)
```

```
=====
Epoch [1/30]
=====
Training Loss      : 1.8314
Training Accuracy  : 0.4886
Validation Loss    : 0.4039
Validation Accuracy : 0.8566
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [2/30]
=====
Training Loss      : 0.3478
Training Accuracy  : 0.8824
Validation Loss    : 0.2360
Validation Accuracy : 0.9124
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [3/30]
=====
Training Loss      : 0.1916
Training Accuracy  : 0.9364
Validation Loss    : 0.2586
Validation Accuracy : 0.9084
Learning Rate      : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [4/30]
=====
Training Loss      : 0.1978
Training Accuracy  : 0.9352
Validation Loss    : 0.1971
Validation Accuracy : 0.9482
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [5/30]
=====
Training Loss      : 0.1513
Training Accuracy  : 0.9483
Validation Loss    : 0.1504
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
```

```
=====
Epoch [6/30]
=====
Training Loss      : 0.1071
Training Accuracy  : 0.9693
Validation Loss    : 0.1064
Validation Accuracy: 0.9641
Learning Rate     : 0.00010000
Best model saved successfully.
=====
Epoch [7/30]
=====
Training Loss      : 0.0772
Training Accuracy  : 0.9778
Validation Loss    : 0.1553
Validation Accuracy: 0.9363
Learning Rate     : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [8/30]
=====
Training Loss      : 0.0913
Training Accuracy  : 0.9722
Validation Loss    : 0.1420
Validation Accuracy: 0.9482
Learning Rate     : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [9/30]
=====
Training Loss      : 0.0760
Training Accuracy  : 0.9767
Validation Loss    : 0.0975
Validation Accuracy: 0.9641
Learning Rate     : 0.00010000
=====
Epoch [10/30]
=====
Training Loss      : 0.0975
Training Accuracy  : 0.9699
Validation Loss    : 0.0858
Validation Accuracy: 0.9641
Learning Rate     : 0.00010000
=====
Epoch [11/30]
=====
Training Loss      : 0.0607
Training Accuracy  : 0.9795
Validation Loss    : 0.1364
Validation Accuracy: 0.9482
Learning Rate     : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [12/30]
=====
Training Loss      : 0.0561
Training Accuracy  : 0.9852
Validation Loss    : 0.0886
Validation Accuracy: 0.9761
Learning Rate     : 0.00010000
Best model saved successfully.
Early Stopping Counter : 2/7
=====
Epoch [13/30]
=====
Training Loss      : 0.0279
Training Accuracy  : 0.9915
Validation Loss    : 0.0926
Validation Accuracy: 0.9721
Learning Rate     : 0.00010000
Early Stopping Counter : 3/7
=====
Epoch [14/30]
=====
Training Loss      : 0.0748
Training Accuracy  : 0.9784
Validation Loss    : 0.1312
Validation Accuracy: 0.9522
Learning Rate     : 0.00010000
Early Stopping Counter : 4/7
=====
Epoch [15/30]
=====
Training Loss      : 0.0346
Training Accuracy  : 0.9920
Validation Loss    : 0.0969
Validation Accuracy: 0.9721
Learning Rate     : 0.00003000
Early Stopping Counter : 5/7
=====
Epoch [16/30]
=====
Training Loss      : 0.0233
Training Accuracy  : 0.9960
Validation Loss    : 0.0619
Validation Accuracy: 0.9761
Learning Rate     : 0.00003000
=====
Epoch [17/30]
=====
Training Loss      : 0.0139
Training Accuracy  : 0.9949
Validation Loss    : 0.0266
Validation Accuracy: 0.9880
Learning Rate     : 0.00003000
Best model saved successfully.
=====
Epoch [18/30]
=====
Training Loss      : 0.0067
Training Accuracy  : 0.9989
Validation Loss    : 0.0242
Validation Accuracy: 0.9880
Learning Rate     : 0.00003000
=====
Epoch [19/30]
=====
Training Loss      : 0.0140
Training Accuracy  : 0.9960
Validation Loss    : 0.0379
Validation Accuracy: 0.9880
Learning Rate     : 0.00003000
Early Stopping Counter : 1/7
=====
Epoch [20/30]
=====
Training Loss      : 0.0082
Training Accuracy  : 0.9989
Validation Loss    : 0.0271
Validation Accuracy: 0.9880
Learning Rate     : 0.00003000
Early Stopping Counter : 2/7
=====
Epoch [21/30]
=====
Training Loss      : 0.0054
Training Accuracy  : 0.9994
Validation Loss    : 0.0348
Validation Accuracy: 0.9841
Learning Rate     : 0.00003000
Early Stopping Counter : 3/7
=====
Epoch [22/30]
=====
Training Loss      : 0.0061
Training Accuracy  : 0.9994
Validation Loss    : 0.0295
Validation Accuracy: 0.9880
Learning Rate     : 0.00003000
Early Stopping Counter : 4/7
=====
Epoch [23/30]
=====
Training Loss      : 0.0057
Training Accuracy  : 0.9989
Validation Loss    : 0.0570
Validation Accuracy: 0.9841
Learning Rate     : 0.00000900
Early Stopping Counter : 5/7
=====
Epoch [24/30]
=====
Training Loss      : 0.0032
Training Accuracy  : 1.0000
Validation Loss    : 0.0582
Validation Accuracy: 0.9841
Learning Rate     : 0.00000900
Early Stopping Counter : 6/7
=====
Epoch [25/30]
```

```
=====
Training Loss      : 0.0044
Training Accuracy  : 0.9994
Validation Loss    : 0.0403
Validation Accuracy: 0.9880
Learning Rate     : 0.00000900
Early Stopping Counter : 7/7
=====
Early Stopping Activated
=====
Model Training Completed Successfully
=====
Best Validation Accuracy : 0.9880
Best model weights restored successfully.
=====
```

```
In [15]: # =====
# Step 15: Evaluate the Swin Transformer Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

(
    test_loss,
    test_accuracy,
    y_true,
    y_pred
) = evaluate_model(
    model,
    test_loader,
    criterion,
    DEVICE
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
=====
Model Evaluation
=====
Test Loss      : 0.0189
Test Accuracy  : 0.9960
=====
```

```
In [16]: # =====
# Step 16: Generate Predictions and Classification Report
# This step displays the precision, recall, F1-score,
# and support values for all ASL classes.
# =====

report = classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    digits=4,
    zero_division=0
)

print("=" * 60)
print("Swin Transformer Classification Report")
print("=" * 60)
print(report)
print("=" * 60)
```

```
=====
Swin Transformer Classification Report
=====
              precision    recall  f1-score   support

    0   0.9286   0.9286   0.9286     14
    1   1.0000   1.0000   1.0000     14
    2   1.0000   1.0000   1.0000     14
    3   1.0000   1.0000   1.0000     14
    4   1.0000   1.0000   1.0000     14
    5   1.0000   1.0000   1.0000     14
    6   1.0000   1.0000   1.0000     14
    7   1.0000   1.0000   1.0000     14
    8   1.0000   1.0000   1.0000     14
    9   1.0000   1.0000   1.0000     14
    a   1.0000   1.0000   1.0000     14
    b   1.0000   1.0000   1.0000     14
    c   1.0000   1.0000   1.0000     14
    d   1.0000   1.0000   1.0000     14
    e   1.0000   1.0000   1.0000     14
    f   1.0000   1.0000   1.0000     14
    g   1.0000   1.0000   1.0000     14
    h   1.0000   1.0000   1.0000     14
    i   1.0000   1.0000   1.0000     14
    j   1.0000   1.0000   1.0000     14
    k   1.0000   1.0000   1.0000     14
    l   1.0000   1.0000   1.0000     14
    m   1.0000   1.0000   1.0000     14
    n   1.0000   1.0000   1.0000     14
    o   0.9286   0.9286   0.9286     14
    p   1.0000   1.0000   1.0000     14
    q   1.0000   1.0000   1.0000     14
    r   1.0000   1.0000   1.0000     14
    s   1.0000   1.0000   1.0000     14
    t   1.0000   1.0000   1.0000     14
    u   1.0000   1.0000   1.0000     14
    v   1.0000   1.0000   1.0000     14
    w   1.0000   1.0000   1.0000     14
    x   1.0000   1.0000   1.0000     14
    y   1.0000   1.0000   1.0000     14
    z   1.0000   1.0000   1.0000     14

 accuracy         0.9960
 macro avg       0.9960
 weighted avg    0.9960

=====
```

```
In [17]: # =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "swin_confusion_matrix.png"
)

cm = confusion_matrix(
    y_true,
    y_pred
)

plt.figure(
    figsize=(16, 14)
)

plt.imshow(
    cm,
    cmap="Blues"
)

plt.title(
    "Swin Transformer Confusion Matrix",
    fontsize=18
)

plt.xlabel(
    "Predicted Label",
    fontsize=14
)

plt.ylabel(
    "True Label",
    fontsize=14
)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

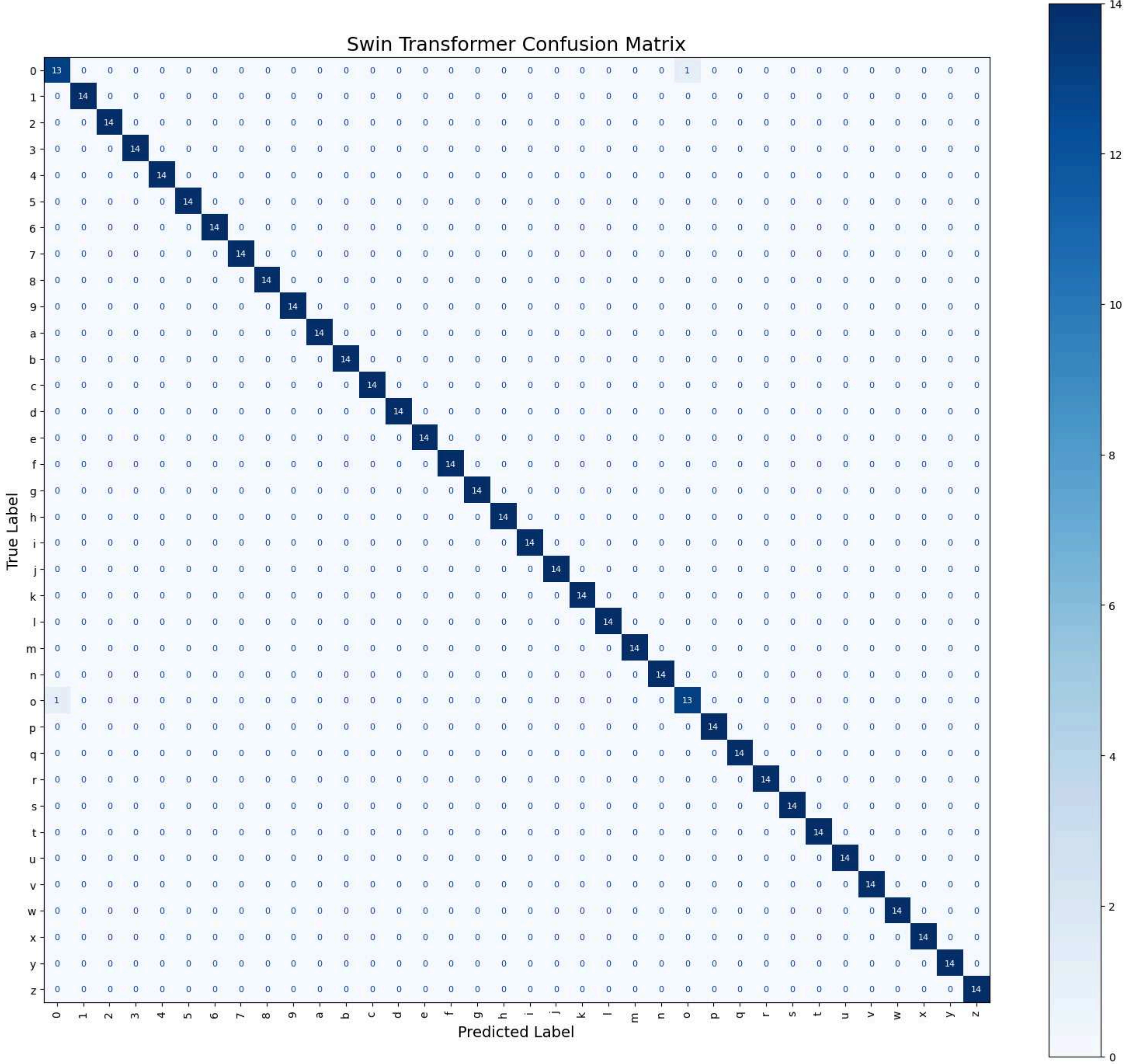
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



```
=====
Confusion Matrix Saved Successfully
=====
Saved Path : ./content/drive/MyDrive/ASU_Final_Project/03_Results/Confusion_Matrices/swin_confusion_matrix.png
=====
```

```
In [18]: # =====
# Step 18: Plot Training History
# This step plots the Training and Validation Accuracy and
# Loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "swin_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "swin_loss_curve.png"
)

# Accuracy Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    train_accuracies,
    label="Training Accuracy"
)

plt.plot(
    validation_accuracies,
    label="Validation Accuracy"
)

plt.title(
    "Swin Transformer Accuracy Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

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

plt.show()

# Loss Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    train_losses,
    label="Training Loss"
)

plt.plot(
    validation_losses,
    label="Validation Loss"
)

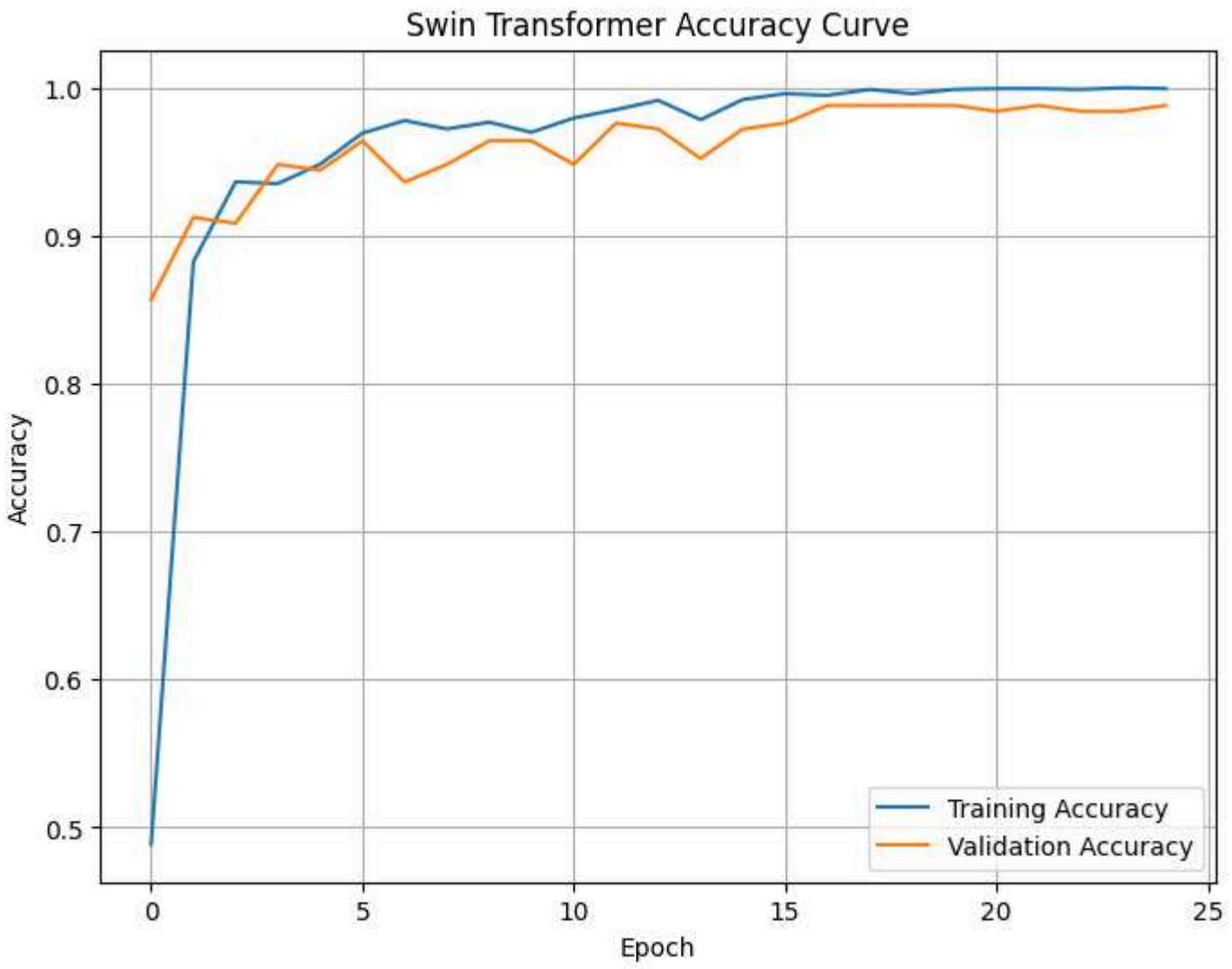
plt.title(
    "Swin Transformer Loss Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

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

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/swin_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/swin_loss_curve.png
=====
```

In [19]:

```
# =====
# Step 19: Save the Final Swin Transformer Model
# This step saves the trained Swin Transformer model
# and the class names for future inference and deployment.
# =====

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_swin_asl_model.pth"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "swin_class_names.txt"
)

torch.save(
    model.state_dict(),
    FINAL_MODEL_PATH
)

with open(
    CLASS_NAMES_PATH,
    "w"
) as file:

    for class_name in CLASS_NAMES:
        file.write(
            f"{class_name}\n"
        )

print("=" * 60)
print("Model Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print("=" * 60)

=====
Model Saved Successfully
=====

Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/SwinTransformer/final_swin_asl_model.pth
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/SwinTransformer/swin_class_names.txt
=====
```

In [20]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures Swin Transformer inference time on the
# complete test dataset using a warm-up run and repeated tests.
# =====

INFERENCE_REPEATS = 3

model.eval()

# Warm-up run
with torch.no_grad():

    for images, _ in test_loader:

        images = images.to(
            DEVICE,
            non_blocking=True
        )

        _ = model(images)

    break

if torch.cuda.is_available():
    torch.cuda.synchronize()

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    start_time = time.perf_counter()

    with torch.no_grad():

        for images, _ in test_loader:

            images = images.to(
                DEVICE,
                non_blocking=True
            )

            _ = model(images)

        if torch.cuda.is_available():
            torch.cuda.synchronize()

        end_time = time.perf_counter()

        inference_times.append(
            end_time - start_time
        )

average_inference_time = np.mean(
    inference_times
)

std_inference_time = np.std(
    inference_times
)

average_time_per_image = (
    average_inference_time
    / len(test_dataset)
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print("Model                : Swin Transformer")
print(f"Test Images             : {len(test_dataset)}")
print(f"Repeated Measurements    : {INFERENCE_REPEATS}")
print(f"Average Total Time      : {average_inference_time:.4f} seconds")
print(f"Standard Deviation       : {std_inference_time:.4f} seconds")
print(f"Average Time per Image  : {average_time_per_image:.6f} seconds")
print("=" * 60)

=====
Standardized Inference Time
=====
Model                : Swin Transformer
Test Images             : 504
Repeated Measurements    : 3
Average Total Time      : 1.6083 seconds
Standard Deviation       : 0.0310 seconds
Average Time per Image  : 0.003191 seconds
=====
```

In [21]:

```
# =====
# Step 21: Save Final Swin Transformer Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "swin"
MODEL_DISPLAY_NAME = "Swin Transformer"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/SwinTransformer/final_swin_asl_model.pth"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "swin_metrics.json"
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpu_name = (
    torch.cuda.get_device_name(0)
    if torch.cuda.is_available()
    else "CPU"
)

total_parameters = sum(
    parameter.numel()
    for parameter in model.parameters()
)

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "PyTorch",
    "gpu": gpu_name,
    "test_images": int(len(test_dataset)),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(total_parameters),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("Swin Transformer Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size     : {model_size_mb:.2f} MB")
print(f"Parameters     : {total_parameters:,}")
print(f"Metrics File   : {METRICS_PATH}")
print("=" * 60)
```

```
=====
Swin Transformer Metrics Saved Successfully
=====
Accuracy      : 0.9960
Precision     : 0.9960
Recall        : 0.9960
F1-Score      : 0.9960
Model Size    : 105.15 MB
Parameters    : 27,547,038
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/swin_metrics.json
=====
```

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs the additional libraries required for
# Hybrid model training and spelling correction.
# =====

!pip -q install timm
!pip -q install textblob
```

```
In [3]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data loading,
# model training, evaluation, visualization, and analysis.
# =====

import os
import random
import time
import warnings

import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim

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

import timm

from sklearn.metrics import (
    classification_report,
    confusion_matrix
)

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("PyTorch Version :", torch.__version__)
print("TIMM Version    :", timm.__version__)
print("=" * 60)
```

```
=====
PyTorch Version : 2.11.0+cu128
TIMM Version    : 1.0.27
=====
```

```
In [4]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies GPU availability.
# =====

SEED = 42

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)

if torch.cuda.is_available():
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)

DEVICE = torch.device(
    "cuda" if torch.cuda.is_available() else "cpu"
)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed : {SEED}")
print(f"Device      : {DEVICE}")

if torch.cuda.is_available():
    print("GPU Name    :", torch.cuda.get_device_name(0))
else:
    raise RuntimeError(
        "GPU was not detected. Enable A100 GPU before training."
    )

print("=" * 60)
```

```
=====
Project Configuration
=====
Random Seed : 42
Device      : cuda
GPU Name    : NVIDIA A100-SXM4-40GB
=====
```

```
In [5]: # =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive
# and verifies that all class folders are available.
# =====

DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Original/asl_dataset"
)

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):

    class_names = sorted([
        folder_name
        for folder_name in os.listdir(DATASET_DIR)
        if os.path.isdir(
            os.path.join(DATASET_DIR, folder_name)
        )
    ])

    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")
    print(f"Total Classes  : {len(class_names)}")

else:
    raise FileNotFoundError(
        f"Dataset not found:\n{DATASET_DIR}"
    )

print("=" * 60)
```

```
=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
=====
```

In [6]:

```
# =====
# Step 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_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"
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(
        os.path.join(DATASET_DIR, folder_name)
    )
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes     : {len(EXPECTED_CLASSES)}")
print(f"Available Classes    : {len(available_classes)}")
print(f"Missing Classes      : {missing_classes}")
print(f"Unexpected Classes   : {unexpected_classes}")
print("=" * 60)

if not dataset_ready:
    raise ValueError(
        "Dataset class verification failed."
    )
```

```
=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes     : 36
Available Classes    : 36
Missing Classes      : []
Unexpected Classes   : []
=====
```

In [7]:

```
# =====
# Step 7: Load Final Dataset Split
# This step loads the same fixed Train, Validation, and Test
# split used by all models for a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

TRAIN_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "train"
)

VALIDATION_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "val"
)

TEST_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "test"
)

required_paths = [
    TRAIN_DIR,
    VALIDATION_DIR,
    TEST_DIR
]

if not all(
    os.path.isdir(path)
    for path in required_paths
):
    raise FileNotFoundError(
        "The final dataset split was not found."
    )

print("=" * 60)
print("Final Dataset Split Loaded Successfully")
print("=" * 60)
print(f"Training Path      : {TRAIN_DIR}")
print(f"Validation Path    : {VALIDATION_DIR}")
print(f"Testing Path       : {TEST_DIR}")
print("=" * 60)
```

```
=====
Final Dataset Split Loaded Successfully
=====
Training Path      : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/train
Validation Path    : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/val
Testing Path       : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/test
=====
```

```
In [8]: # =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total.
# =====

VALID_EXTENSIONS = (
    ".jpg",
    ".jpeg",
    ".png",
    ".bmp",
    ".webp"
)

split_paths = {
    "Training": TRAIN_DIR,
    "Validation": VALIDATION_DIR,
    "Testing": TEST_DIR
}

grand_total = 0

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

for split_name, split_path in split_paths.items():

    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(
            split_path,
            class_name
        )

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(
                VALID_EXTENSIONS
            )
        )

    grand_total += total_images

    print(
        f"{split_name:<12}: "
        f"{total_images} images"
    )

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training    : 1760 images
Validation  : 251 images
Testing     : 504 images
-----
Total Images : 2515
=====
```

```
In [9]: # =====
# Step 9: Create Image Transformations
# This step applies preprocessing and data augmentation to
# prepare images for both Hybrid feature extractors.
# =====

IMAGE_SIZE = 224
BATCH_SIZE = 16

train_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.RandomRotation(15),

    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

validation_test_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

print("=" * 60)
print("Image Transformations Created Successfully")
print("=" * 60)
print(f"Image Size : {IMAGE_SIZE} x {IMAGE_SIZE}")
print(f"Batch Size : {BATCH_SIZE}")
print("=" * 60)
```

```
=====
Image Transformations Created Successfully
=====
Image Size : 224 x 224
Batch Size : 16
=====
```

In [10]:

```
# =====
# Step 10: Load Datasets and Create DataLoaders
# This step loads the Training, Validation, and Testing
# datasets and creates DataLoaders for Hybrid model training.
# =====

train_dataset = datasets.ImageFolder(
    root=TRAIN_DIR,
    transform=train_transform
)

validation_dataset = datasets.ImageFolder(
    root=VALIDATION_DIR,
    transform=validation_test_transform
)

test_dataset = datasets.ImageFolder(
    root=TEST_DIR,
    transform=validation_test_transform
)

CLASS_NAMES = train_dataset.classes
NUM_CLASSES = len(CLASS_NAMES)

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2,
    pin_memory=True
)

validation_loader = DataLoader(
    validation_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

print("=" * 60)
print("Datasets and DataLoaders Created Successfully")
print("=" * 60)
print(f"Training Samples   : {len(train_dataset)}")
print(f"Validation Samples : {len(validation_dataset)}")
print(f"Testing Samples    : {len(test_dataset)}")
print(f"Number of Classes  : {NUM_CLASSES}")
print(f"Training Batches   : {len(train_loader)}")
print(f"Validation Batches : {len(validation_loader)}")
print(f"Testing Batches    : {len(test_loader)}")
print("=" * 60)
```

```
=====
Datasets and DataLoaders Created Successfully
=====
Training Samples   : 1760
Validation Samples : 251
Testing Samples    : 504
Number of Classes  : 36
Training Batches   : 110
Validation Batches : 16
Testing Batches    : 32
=====
```

```
In [11]: # =====
# Step 11: Build the Hybrid MobileNetV2-Swin Model
# This step combines MobileNetV2 and Swin Transformer feature
# vectors and adds a new classification head for 36 classes.
# =====

MOBILENET_MODEL_NAME = "mobilenetv2_100"
SWIN_MODEL_NAME = "swin_tiny_patch4_window7_224"

class HybridMobileNetSwin(nn.Module):

    def __init__(self, num_classes):

        super().__init__()

        self.mobilenet = timm.create_model(
            MOBILENET_MODEL_NAME,
            pretrained=True,
            num_classes=0,
            global_pool="avg"
        )

        self.swin = timm.create_model(
            SWIN_MODEL_NAME,
            pretrained=True,
            num_classes=0,
            global_pool="avg"
        )

        mobilenet_features = (
            self.mobilenet.num_features
        )

        swin_features = (
            self.swin.num_features
        )

        combined_features = (
            mobilenet_features
            + swin_features
        )

        self.classifier = nn.Sequential(
            nn.Linear(
                combined_features,
                512
            ),

            nn.ReLU(),

            nn.Dropout(0.5),

            nn.Linear(
                512,
                num_classes
            )
        )

        self.mobilenet_feature_size = (
            mobilenet_features
        )

        self.swin_feature_size = (
            swin_features
        )

        self.combined_feature_size = (
            combined_features
        )

    def forward(self, images):

        mobilenet_features = self.mobilenet(
            images
        )

        swin_features = self.swin(
            images
        )

        fused_features = torch.cat(
            (
                mobilenet_features,
                swin_features
            ),
            dim=1
        )

        output = self.classifier(
            fused_features
        )

        return output

model = HybridMobileNetSwin(
    num_classes=NUM_CLASSES
)

model = model.to(DEVICE)

print("=" * 60)
print("Hybrid Model Created Successfully")
print("=" * 60)
print(f"MobileNetV2 Model : {MOBILENET_MODEL_NAME}")
print(f"Swin Model : {SWIN_MODEL_NAME}")
print(f"MobileNet Features : {model.mobilenet_feature_size}")
print(f"Swin Features : {model.swin_feature_size}")
print(f"Combined Features : {model.combined_feature_size}")
print(f"Number of Classes : {NUM_CLASSES}")
print(
    f"Total Parameters : "
    f"{sum(p.numel() for p in model.parameters()):,}"
)
print(f"Device : {DEVICE}")
print("=" * 60)
```

```
model.safetensors:  0%|          | 0.00/14.2M [00:00<?, ?B/s]

model.safetensors:  0%|          | 0.00/114M [00:00<?, ?B/s]

=====
Hybrid Model Created Successfully
=====
MobileNetV2 Model : mobilenetv2_100
Swin Model : swin_tiny_patch4_window7_224
MobileNet Features : 1280
Swin Features : 768
Combined Features : 2048
Number of Classes : 36
Total Parameters : 30,810,782
Device : cuda
=====
```

In [12]:

```
# =====
# Step 12: Configure the Hybrid Model
# This step defines the loss function, optimizer, learning
# rate scheduler, and model saving path.
# =====

LEARNING_RATE = 0.0001

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/Hybrid"
)

os.makedirs(
    MODEL_DIR,
    exist_ok=True
)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_hybrid_mobilenetv2_swin_asl_model.pth"
)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(
    model.parameters(),
    lr=LEARNING_RATE,
    weight_decay=0.01
)

scheduler = optim.lr_scheduler.ReduceLR0nPlateau(
    optimizer,
    mode="min",
    factor=0.3,
    patience=4,
    min_lr=1e-6
)

print("=" * 60)
print("Training Setup Configured Successfully")
print("=" * 60)
print("Optimizer      : AdamW")
print(f"Learning Rate   : {LEARNING_RATE}")
print("Loss Function   : Cross Entropy Loss")
print("Weight Decay    : 0.01")
print(f"Best Model Path : {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Setup Configured Successfully
=====
Optimizer      : AdamW
Learning Rate   : 0.0001
Loss Function   : Cross Entropy Loss
Weight Decay    : 0.01
Best Model Path : /content/drive/MyDrive/ASL_Final_Project/02_Models/Hybrid/best_hybrid_mobilenetv2_swin_asl_model.pth
=====
```

```
In [13]: # =====
# Step 13: Create Training and Evaluation Functions
# This step creates the functions used to train and evaluate
# the Hybrid model during each epoch.
# =====

def train_one_epoch(
    model,
    data_loader,
    criterion,
    optimizer,
    device
):
    model.train()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for images, labels in data_loader:

        images = images.to(
            device,
            non_blocking=True
        )

        labels = labels.to(
            device,
            non_blocking=True
        )

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(
            outputs,
            labels
        )

        loss.backward()

        optimizer.step()

        running_loss += (
            loss.item() * images.size(0)
        )

        predicted_labels = outputs.argmax(
            dim=1
        )

        correct_predictions += (
            predicted_labels == labels
        ).sum().item()

        total_samples += labels.size(0)

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return epoch_loss, epoch_accuracy

def evaluate_model(
    model,
    data_loader,
    criterion,
    device
):
    model.eval()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    true_labels = []
    predicted_labels = []

    with torch.no_grad():

        for images, labels in data_loader:

            images = images.to(
                device,
                non_blocking=True
            )

            labels = labels.to(
                device,
                non_blocking=True
            )

            outputs = model(images)

            loss = criterion(
                outputs,
                labels
            )

            running_loss += (
                loss.item() * images.size(0)
            )

            predictions = outputs.argmax(
                dim=1
            )

            correct_predictions += (
                predictions == labels
            ).sum().item()

            total_samples += labels.size(0)

            true_labels.extend(
                labels.cpu().numpy()
            )

            predicted_labels.extend(
                predictions.cpu().numpy()
            )

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return (
        epoch_loss,
        epoch_accuracy,
        true_labels,
        predicted_labels
    )

print("=" * 60)
print("Training and Evaluation Functions Created Successfully")
print("=" * 60)
```

```
=====
Training and Evaluation Functions Created Successfully
=====
```

In [14]:

```
# =====
# Step 14: Train the Hybrid MobileNetV2-Swin Model
# This step trains the model, monitors validation performance,
# applies Early Stopping, and saves the best model.
# =====

EPOCHS = 30
EARLY_STOPPING_PATIENCE = 7

train_losses = []
validation_losses = []

train_accuracies = []
validation_accuracies = []

best_validation_accuracy = 0.0
best_validation_loss = float("inf")

early_stopping_counter = 0

for epoch in range(EPOCHS):

    train_loss, train_accuracy = train_one_epoch(
        model,
        train_loader,
        criterion,
        optimizer,
        DEVICE
    )

    (
        validation_loss,
        validation_accuracy,
        _
    ) = evaluate_model(
        model,
        validation_loader,
        criterion,
        DEVICE
    )

    scheduler.step(
        validation_loss
    )

    train_losses.append(train_loss)
    validation_losses.append(validation_loss)

    train_accuracies.append(train_accuracy)
    validation_accuracies.append(validation_accuracy)

    current_learning_rate = (
        optimizer.param_groups[0]["lr"]
    )

    print("=" * 60)
    print(f"Epoch [{epoch + 1}/{EPOCHS}]")
    print("=" * 60)
    print(f"Training Loss      : {train_loss:.4f}")
    print(f"Training Accuracy     : {train_accuracy:.4f}")
    print(f"Validation Loss        : {validation_loss:.4f}")
    print(f"Validation Accuracy    : {validation_accuracy:.4f}")
    print(f"Learning Rate         : {current_learning_rate:.8f}")

    if validation_accuracy > best_validation_accuracy:

        best_validation_accuracy = (
            validation_accuracy
        )

        torch.save(
            model.state_dict(),
            BEST_MODEL_PATH
        )

        print("Best model saved successfully.")

    if validation_loss < best_validation_loss:

        best_validation_loss = (
            validation_loss
        )

        early_stopping_counter = 0

    else:

        early_stopping_counter += 1

        print(
            "Early Stopping Counter : "
            f"{early_stopping_counter}/"
            f"{EARLY_STOPPING_PATIENCE}"
        )

    if (
        early_stopping_counter
        >= EARLY_STOPPING_PATIENCE
    ):

        print("=" * 60)
        print("Early Stopping Activated")
        print("=" * 60)

        break

model.load_state_dict(
    torch.load(
        BEST_MODEL_PATH,
        map_location=DEVICE
    )
)

model = model.to(DEVICE)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
print(
    f"Best Validation Accuracy : "
    f"{best_validation_accuracy:.4f}"
)
print("Best model weights restored successfully.")
print("=" * 60)
```

```
=====
Epoch [1/30]
=====
Training Loss      : 2.3869
Training Accuracy  : 0.3665
Validation Loss    : 0.8062
Validation Accuracy : 0.7291
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [2/30]
=====
Training Loss      : 0.6259
Training Accuracy  : 0.8091
Validation Loss    : 0.2472
Validation Accuracy : 0.9203
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [3/30]
=====
Training Loss      : 0.3188
Training Accuracy  : 0.8960
Validation Loss    : 0.1909
Validation Accuracy : 0.9243
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [4/30]
=====
Training Loss      : 0.2255
Training Accuracy  : 0.9193
Validation Loss    : 0.1735
Validation Accuracy : 0.9602
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [5/30]
=====
Training Loss      : 0.2183
Training Accuracy  : 0.9233
Validation Loss    : 0.1247
Validation Accuracy : 0.9482
Learning Rate      : 0.00010000
```

```
=====
Epoch [6/30]
=====
Training Loss      : 0.1660
Training Accuracy  : 0.9500
Validation Loss    : 0.1181
Validation Accuracy: 0.9482
Learning Rate     : 0.00010000
=====
Epoch [7/30]
=====
Training Loss      : 0.1140
Training Accuracy  : 0.9659
Validation Loss    : 0.0973
Validation Accuracy: 0.9562
Learning Rate     : 0.00010000
=====
Epoch [8/30]
=====
Training Loss      : 0.0667
Training Accuracy  : 0.9807
Validation Loss    : 0.0645
Validation Accuracy: 0.9761
Learning Rate     : 0.00010000
Best model saved successfully.
=====
Epoch [9/30]
=====
Training Loss      : 0.0802
Training Accuracy  : 0.9790
Validation Loss    : 0.1174
Validation Accuracy: 0.9562
Learning Rate     : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [10/30]
=====
Training Loss      : 0.0797
Training Accuracy  : 0.9756
Validation Loss    : 0.0348
Validation Accuracy: 0.9841
Learning Rate     : 0.00010000
Best model saved successfully.
=====
Epoch [11/30]
=====
Training Loss      : 0.0955
Training Accuracy  : 0.9705
Validation Loss    : 0.0993
Validation Accuracy: 0.9681
Learning Rate     : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [12/30]
=====
Training Loss      : 0.0777
Training Accuracy  : 0.9761
Validation Loss    : 0.0584
Validation Accuracy: 0.9761
Learning Rate     : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [13/30]
=====
Training Loss      : 0.0470
Training Accuracy  : 0.9875
Validation Loss    : 0.0512
Validation Accuracy: 0.9841
Learning Rate     : 0.00010000
Early Stopping Counter : 3/7
=====
Epoch [14/30]
=====
Training Loss      : 0.0437
Training Accuracy  : 0.9864
Validation Loss    : 0.0370
Validation Accuracy: 0.9880
Learning Rate     : 0.00010000
Best model saved successfully.
Early Stopping Counter : 4/7
=====
Epoch [15/30]
=====
Training Loss      : 0.0386
Training Accuracy  : 0.9881
Validation Loss    : 0.0809
Validation Accuracy: 0.9721
Learning Rate     : 0.00003000
Early Stopping Counter : 5/7
=====
Epoch [16/30]
=====
Training Loss      : 0.0179
Training Accuracy  : 0.9932
Validation Loss    : 0.0305
Validation Accuracy: 0.9841
Learning Rate     : 0.00003000
=====
Epoch [17/30]
=====
Training Loss      : 0.0126
Training Accuracy  : 0.9977
Validation Loss    : 0.0167
Validation Accuracy: 0.9960
Learning Rate     : 0.00003000
Best model saved successfully.
=====
Epoch [18/30]
=====
Training Loss      : 0.0085
Training Accuracy  : 0.9989
Validation Loss    : 0.0295
Validation Accuracy: 0.9920
Learning Rate     : 0.00003000
Early Stopping Counter : 1/7
=====
Epoch [19/30]
=====
Training Loss      : 0.0105
Training Accuracy  : 0.9977
Validation Loss    : 0.0341
Validation Accuracy: 0.9920
Learning Rate     : 0.00003000
Early Stopping Counter : 2/7
=====
Epoch [20/30]
=====
Training Loss      : 0.0134
Training Accuracy  : 0.9966
Validation Loss    : 0.0294
Validation Accuracy: 0.9920
Learning Rate     : 0.00003000
Early Stopping Counter : 3/7
=====
Epoch [21/30]
=====
Training Loss      : 0.0071
Training Accuracy  : 0.9989
Validation Loss    : 0.0364
Validation Accuracy: 0.9920
Learning Rate     : 0.00003000
Early Stopping Counter : 4/7
=====
Epoch [22/30]
=====
Training Loss      : 0.0061
Training Accuracy  : 0.9989
Validation Loss    : 0.0179
Validation Accuracy: 0.9920
Learning Rate     : 0.00000900
Early Stopping Counter : 5/7
=====
Epoch [23/30]
=====
Training Loss      : 0.0091
Training Accuracy  : 0.9977
Validation Loss    : 0.0254
Validation Accuracy: 0.9920
Learning Rate     : 0.00000900
Early Stopping Counter : 6/7
=====
Epoch [24/30]
=====
Training Loss      : 0.0072
Training Accuracy  : 0.9983
Validation Loss    : 0.0174
Validation Accuracy: 0.9920
Learning Rate     : 0.00000900
Early Stopping Counter : 7/7
=====
```

```
Early Stopping Activated
=====

Model Training Completed Successfully
=====

Best Validation Accuracy : 0.9960
Best model weights restored successfully.
=====
```

```
In [15]: # =====
# Step 15: Evaluate the Hybrid MobileNetV2-Swin Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

(
    test_loss,
    test_accuracy,
    y_true,
    y_pred
) = evaluate_model(
    model,
    test_loader,
    criterion,
    DEVICE
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
=====
Model Evaluation
=====

Test Loss      : 0.0112
Test Accuracy : 0.9980
=====
```

```
In [16]: # =====
# Step 16: Generate Predictions and Classification Report
# This step displays the precision, recall, F1-score,
# and support values for all ASL classes.
# =====

report = classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    digits=4,
    zero_division=0
)

print("=" * 60)
print("Hybrid MobileNetV2-Swin Classification Report")
print("=" * 60)
print(report)
print("=" * 60)
```

```
=====
Hybrid MobileNetV2-Swin Classification Report
=====
              precision    recall  f1-score   support

0           1.0000      0.9286      0.9630         14
1           1.0000      1.0000      1.0000         14
2           1.0000      1.0000      1.0000         14
3           1.0000      1.0000      1.0000         14
4           1.0000      1.0000      1.0000         14
5           1.0000      1.0000      1.0000         14
6           1.0000      1.0000      1.0000         14
7           1.0000      1.0000      1.0000         14
8           1.0000      1.0000      1.0000         14
9           1.0000      1.0000      1.0000         14
a           1.0000      1.0000      1.0000         14
b           1.0000      1.0000      1.0000         14
c           1.0000      1.0000      1.0000         14
d           1.0000      1.0000      1.0000         14
e           1.0000      1.0000      1.0000         14
f           1.0000      1.0000      1.0000         14
g           1.0000      1.0000      1.0000         14
h           1.0000      1.0000      1.0000         14
i           1.0000      1.0000      1.0000         14
j           1.0000      1.0000      1.0000         14
k           1.0000      1.0000      1.0000         14
l           1.0000      1.0000      1.0000         14
m           1.0000      1.0000      1.0000         14
n           1.0000      1.0000      1.0000         14
o           0.9333      1.0000      0.9655         14
p           1.0000      1.0000      1.0000         14
q           1.0000      1.0000      1.0000         14
r           1.0000      1.0000      1.0000         14
s           1.0000      1.0000      1.0000         14
t           1.0000      1.0000      1.0000         14
u           1.0000      1.0000      1.0000         14
v           1.0000      1.0000      1.0000         14
w           1.0000      1.0000      1.0000         14
x           1.0000      1.0000      1.0000         14
y           1.0000      1.0000      1.0000         14
z           1.0000      1.0000      1.0000         14

accuracy              0.9980         504
macro avg           0.9981      0.9980      0.9980         504
weighted avg        0.9981      0.9980      0.9980         504
=====
```

```
In [17]: # =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "hybrid_mobilenetv2_swin_confusion_matrix.png"
)

cm = confusion_matrix(
    y_true,
    y_pred
)

plt.figure(
    figsize=(16, 14)
)

plt.imshow(
    cm,
    cmap="Blues"
)

plt.title(
    "Hybrid MobileNetV2-Swin Confusion Matrix",
    fontsize=18
)

plt.xlabel(
    "Predicted Label",
    fontsize=14
)

plt.ylabel(
    "True Label",
    fontsize=14
)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

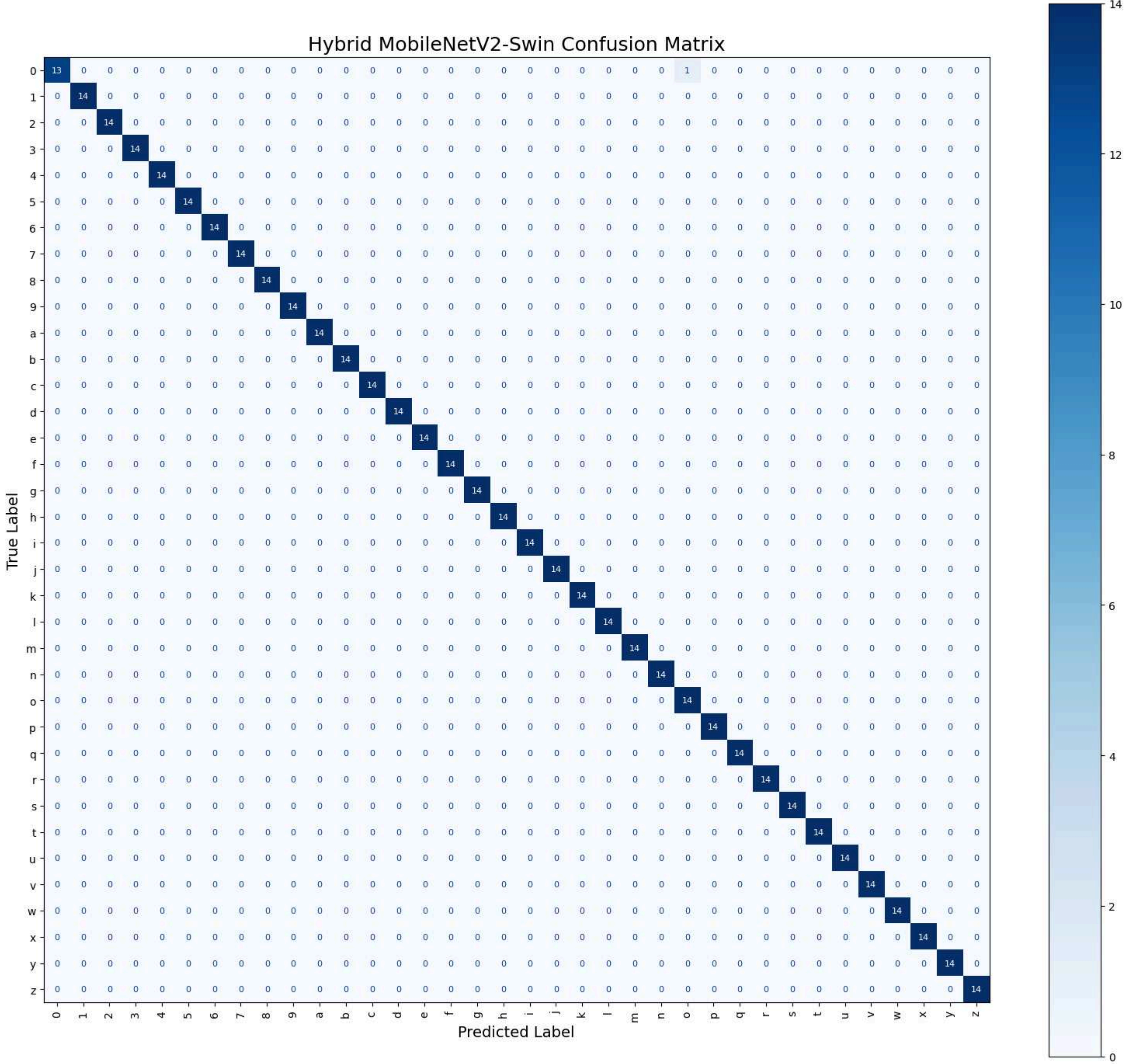
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



```
=====
Confusion Matrix Saved Successfully
=====
Saved Path : ./content/drive/MyDrive/ASU_Final_Project/03_Results/Confusion_Matrices/hybrid_mobilenetv2_swin_confusion_matrix.png
=====
```

```
In [18]: # =====
# Step 18: Plot Training History
# This step plots the Training and Validation Accuracy and
# Loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "hybrid_mobilenetv2_swin_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "hybrid_mobilenetv2_swin_loss_curve.png"
)

# Accuracy Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    train_accuracies,
    label="Training Accuracy"
)

plt.plot(
    validation_accuracies,
    label="Validation Accuracy"
)

plt.title(
    "Hybrid MobileNetV2-Swin Accuracy Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

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

plt.show()

# Loss Curve
plt.figure(
    figsize=(8, 6)
)

plt.plot(
    train_losses,
    label="Training Loss"
)

plt.plot(
    validation_losses,
    label="Validation Loss"
)

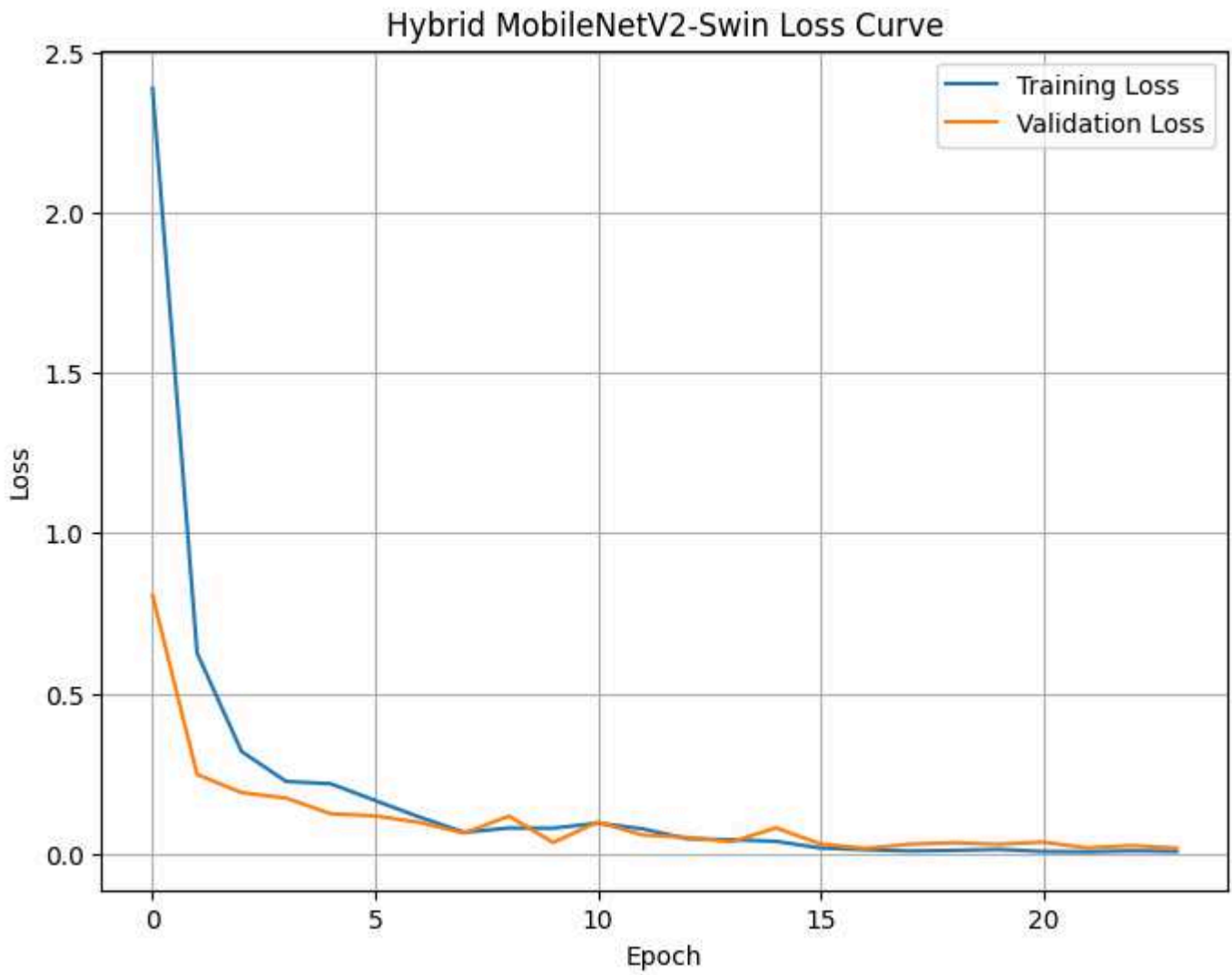
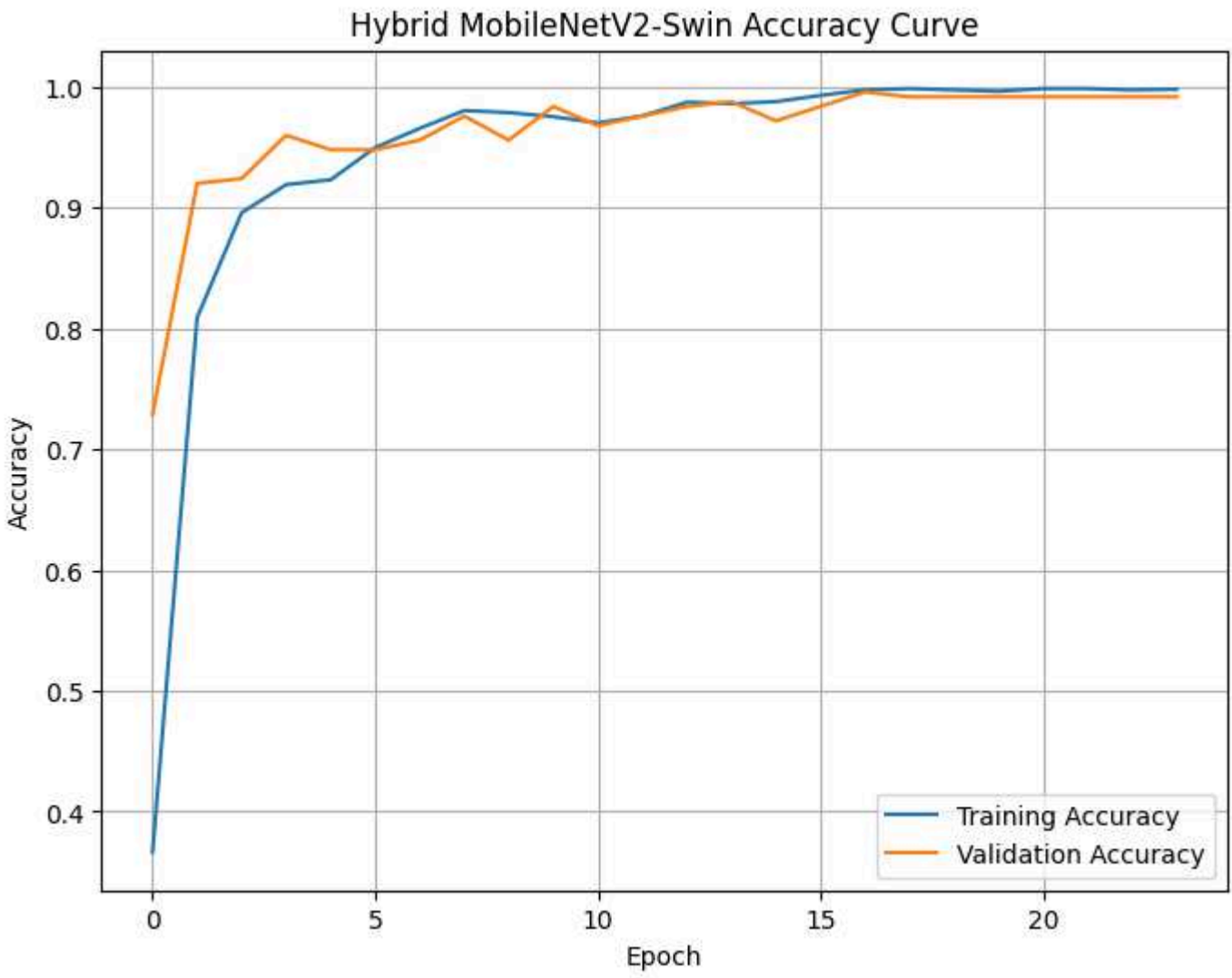
plt.title(
    "Hybrid MobileNetV2-Swin Loss Curve"
)

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

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

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/hybrid_mobilenetv2_swin_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/hybrid_mobilenetv2_swin_loss_curve.png
=====
```

In [19]:

```
# =====
# Step 19: Save the Final Hybrid MobileNetV2-Swin Model
# This step saves the trained Hybrid model and class names
# for future inference and deployment.
# =====

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_hybrid_mobilenetv2_swin_asl_model.pth"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "hybrid_mobilenetv2_swin_class_names.txt"
)

torch.save(
    model.state_dict(),
    FINAL_MODEL_PATH
)

with open(
    CLASS_NAMES_PATH,
    "w"
) as file:

    for class_name in CLASS_NAMES:
        file.write(
            f"{class_name}\n"
        )

print("=" * 60)
print("Model Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print("=" * 60)

=====
Model Saved Successfully
=====

Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/Hybrid/final_hybrid_mobilenetv2_swin_asl_model.pth
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/Hybrid/hybrid_mobilenetv2_swin_class_names.txt
=====
```

In [20]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures Hybrid model inference time on the
# complete test dataset using a warm-up run and repeated tests.
# =====

INFERENCE_REPEATS = 3

model.eval()

# Warm-up run
with torch.no_grad():

    for images, _ in test_loader:

        images = images.to(
            DEVICE,
            non_blocking=True
        )

        _ = model(images)

        break

if torch.cuda.is_available():
    torch.cuda.synchronize()

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    start_time = time.perf_counter()

    with torch.no_grad():

        for images, _ in test_loader:

            images = images.to(
                DEVICE,
                non_blocking=True
            )

            _ = model(images)

        if torch.cuda.is_available():
            torch.cuda.synchronize()

        end_time = time.perf_counter()

        inference_times.append(
            end_time - start_time
        )

average_inference_time = np.mean(
    inference_times
)

std_inference_time = np.std(
    inference_times
)

average_time_per_image = (
    average_inference_time
    / len(test_dataset)
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print("Model                : Hybrid MobileNetV2-Swin")
print(f"Test Images              : {len(test_dataset)}")
print(f"Repeated Measurements    : {INFERENCE_REPEATS}")
print(f"Average Total Time      : {average_inference_time:.4f} seconds")
print(f"Standard Deviation       : {std_inference_time:.4f} seconds")
print(f"Average Time per Image  : {average_time_per_image:.6f} seconds")
print("=" * 60)

=====
Standardized Inference Time
=====
Model                : Hybrid MobileNetV2-Swin
Test Images          : 504
Repeated Measurements : 3
Average Total Time    : 2.5131 seconds
Standard Deviation    : 1.1216 seconds
Average Time per Image : 0.004986 seconds
=====
```

In [21]:

```
# =====
# Step 21: Save Final Hybrid MobileNetV2-Swin Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "hybrid"
MODEL_DISPLAY_NAME = "Hybrid MobileNetV2-Swin"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/Hybrid/"
    "final_hybrid_mobilenetv2_swin_asl_model.pth"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "hybrid_metrics.json"
)

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

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

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpu_name = (
    torch.cuda.get_device_name(0)
    if torch.cuda.is_available()
    else "CPU"
)

total_parameters = sum(
    parameter.numel()
    for parameter in model.parameters()
)

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "PyTorch",
    "gpu": gpu_name,
    "test_images": int(len(test_dataset)),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(total_parameters),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("Hybrid Model Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size    : {model_size_mb:.2f} MB")
print(f"Parameters    : {total_parameters:,}")
print(f"Metrics File  : {METRICS_PATH}")
print("=" * 60)

=====
Hybrid Model Metrics Saved Successfully
=====
Accuracy      : 0.9980
Precision     : 0.9981
Recall        : 0.9980
F1-Score      : 0.9980
Model Size    : 117.86 MB
Parameters    : 30,810,702
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/hybrid_metrics.json
=====
```

```
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
):

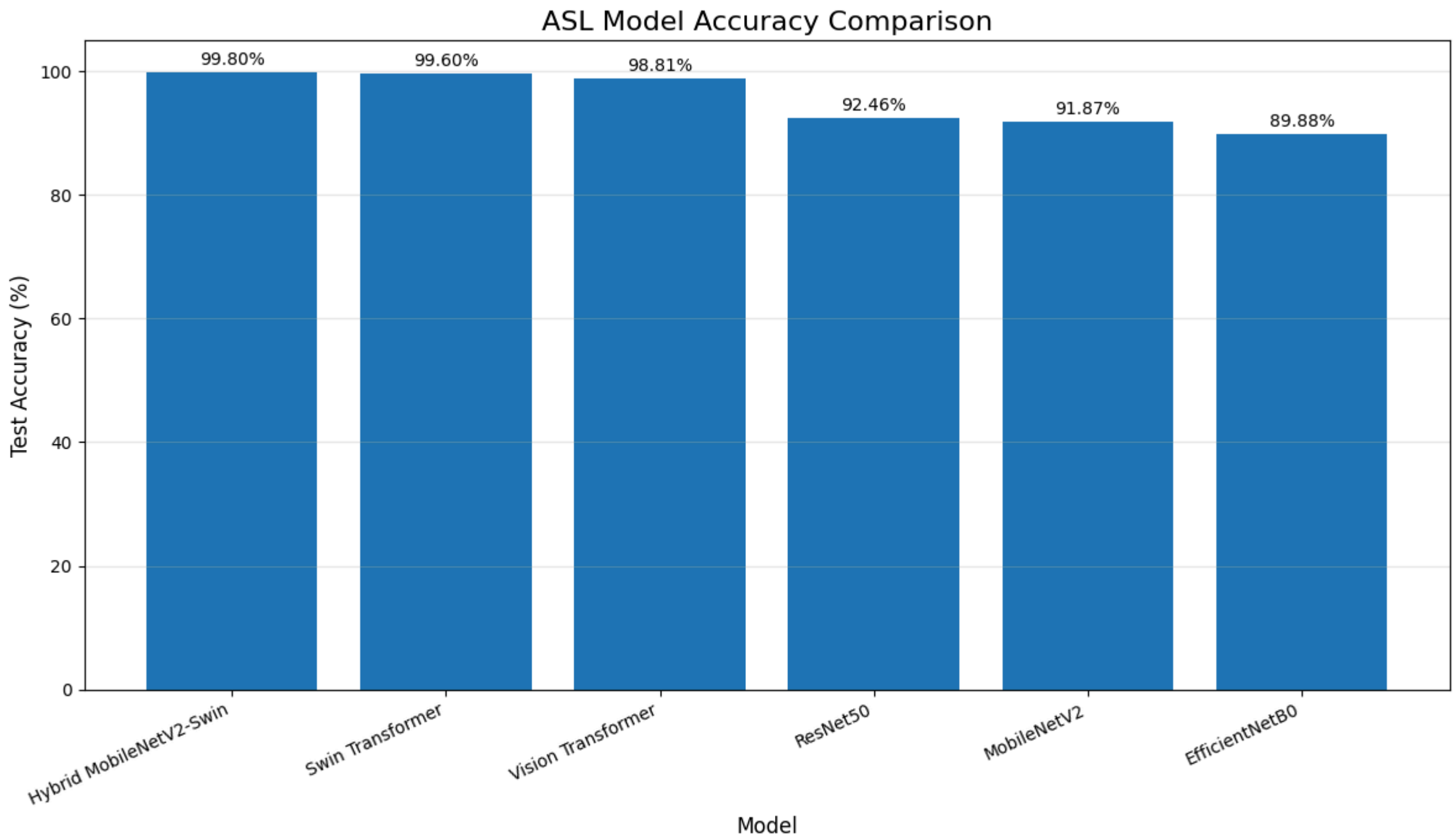
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.6,
        f"{value:.2f}%",
        ha="center",
        va="bottom",
        fontsize=10
    )

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()

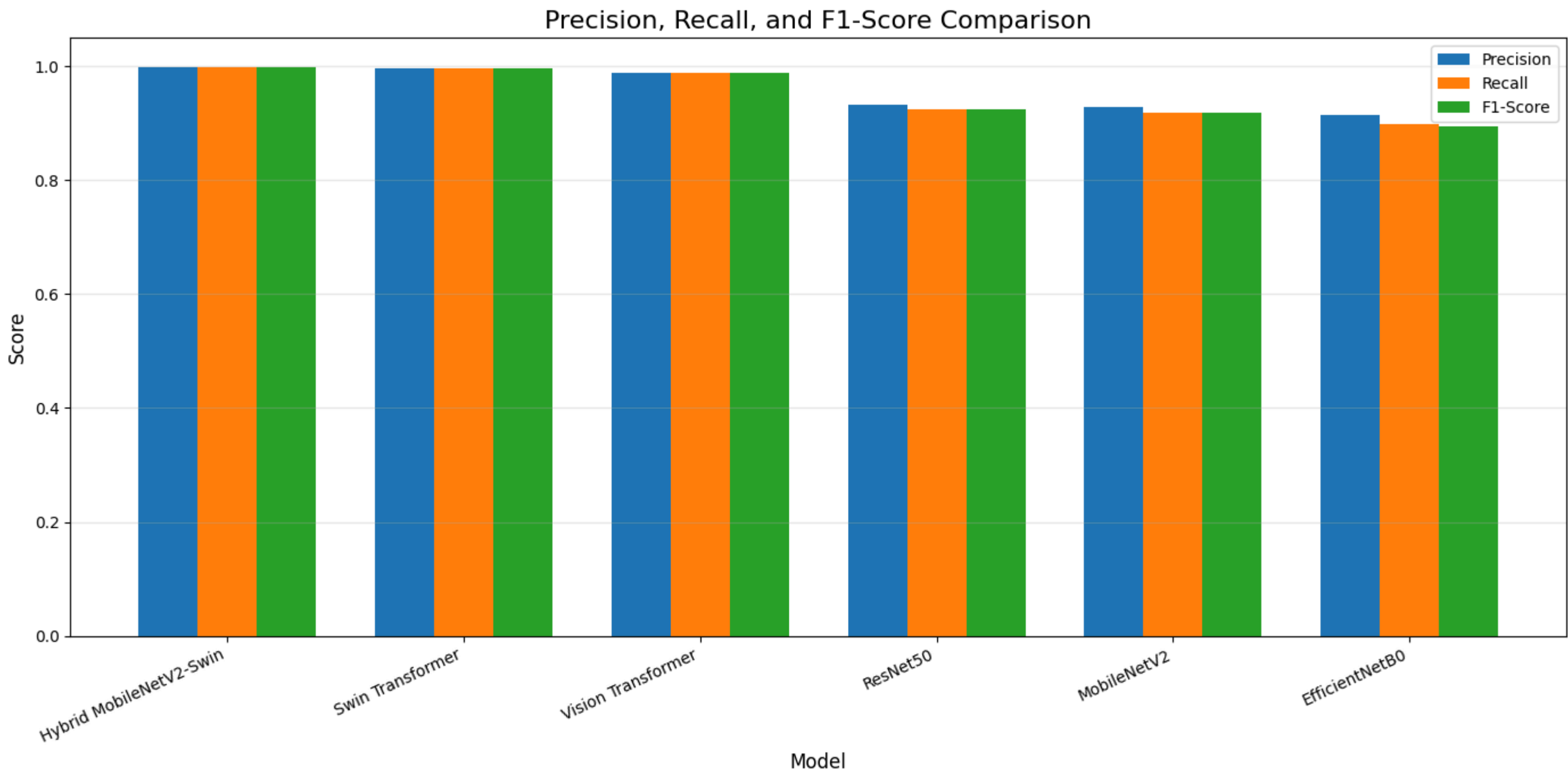
plt.grid(
    axis="y",
    alpha=0.3
)

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.
# =====

INFERENCE_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)"]
):

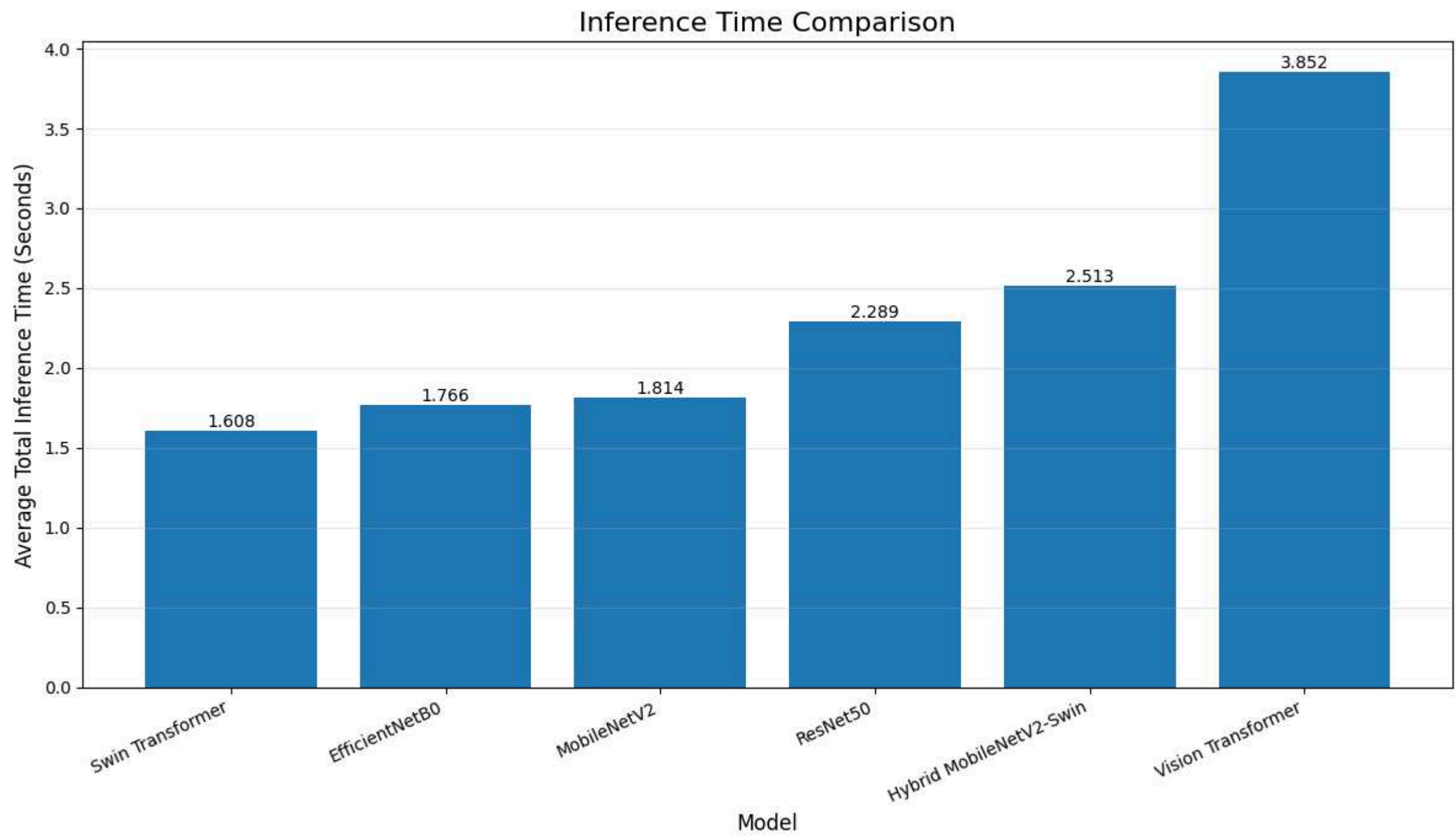
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.3f}",
        ha="center",
        va="bottom",
        fontsize=10
    )

plt.tight_layout()

plt.savefig(
    INFERENCE_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)"]
):

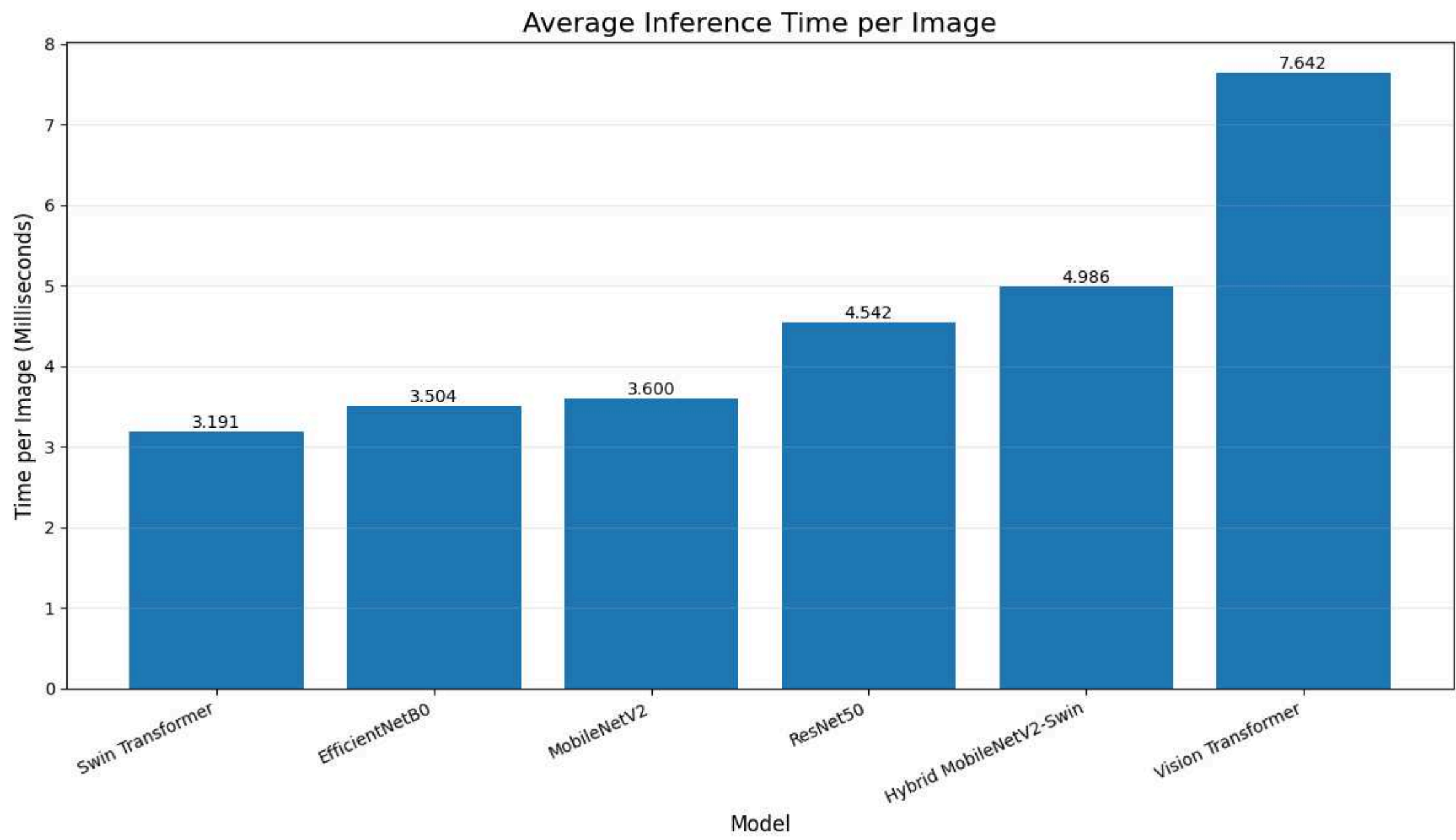
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.3f}",
        ha="center",
        va="bottom",
        fontsize=10
    )

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)"]
):

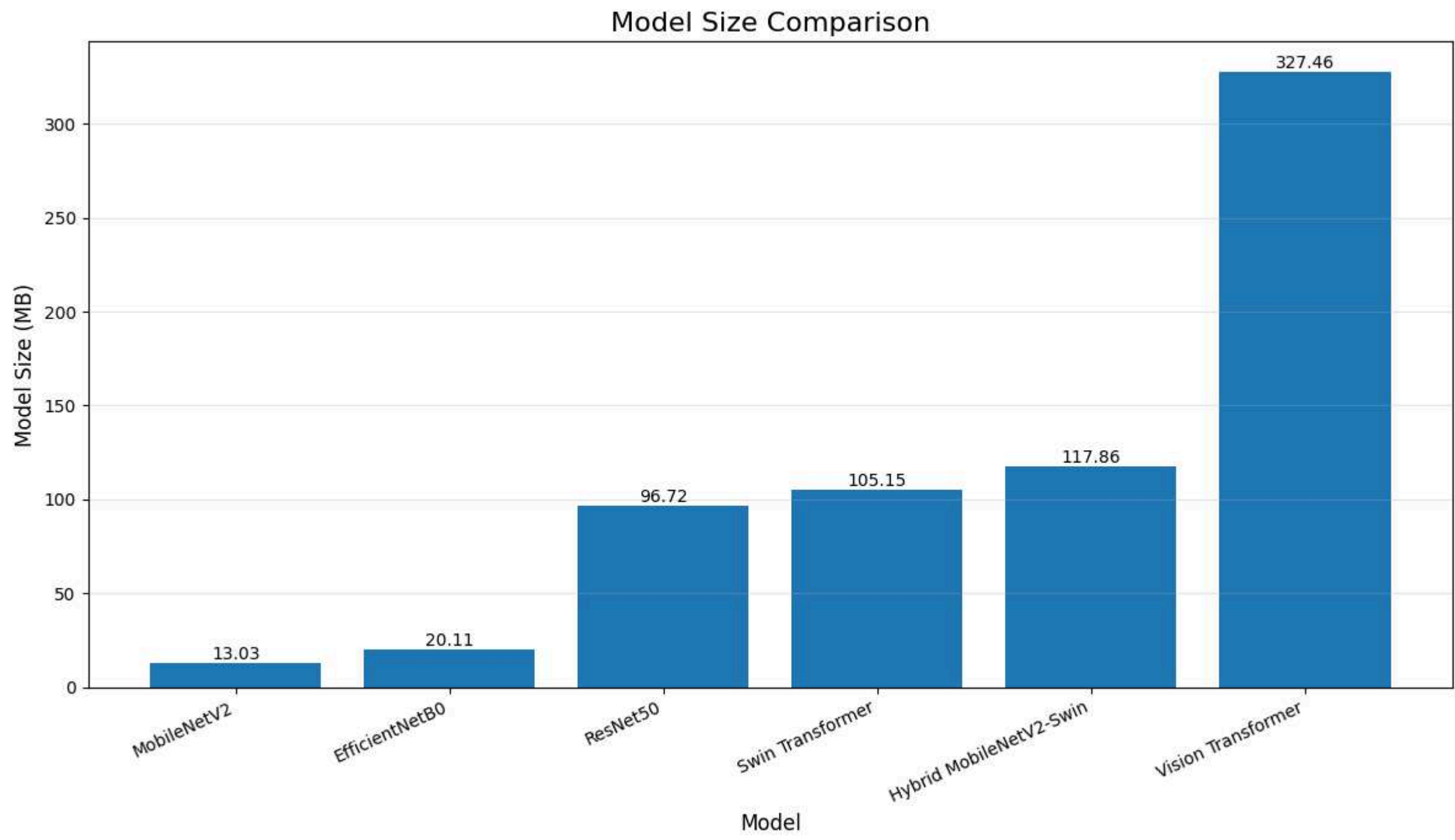
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.2f}",
        ha="center",
        va="bottom",
        fontsize=10
    )

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)"]
):

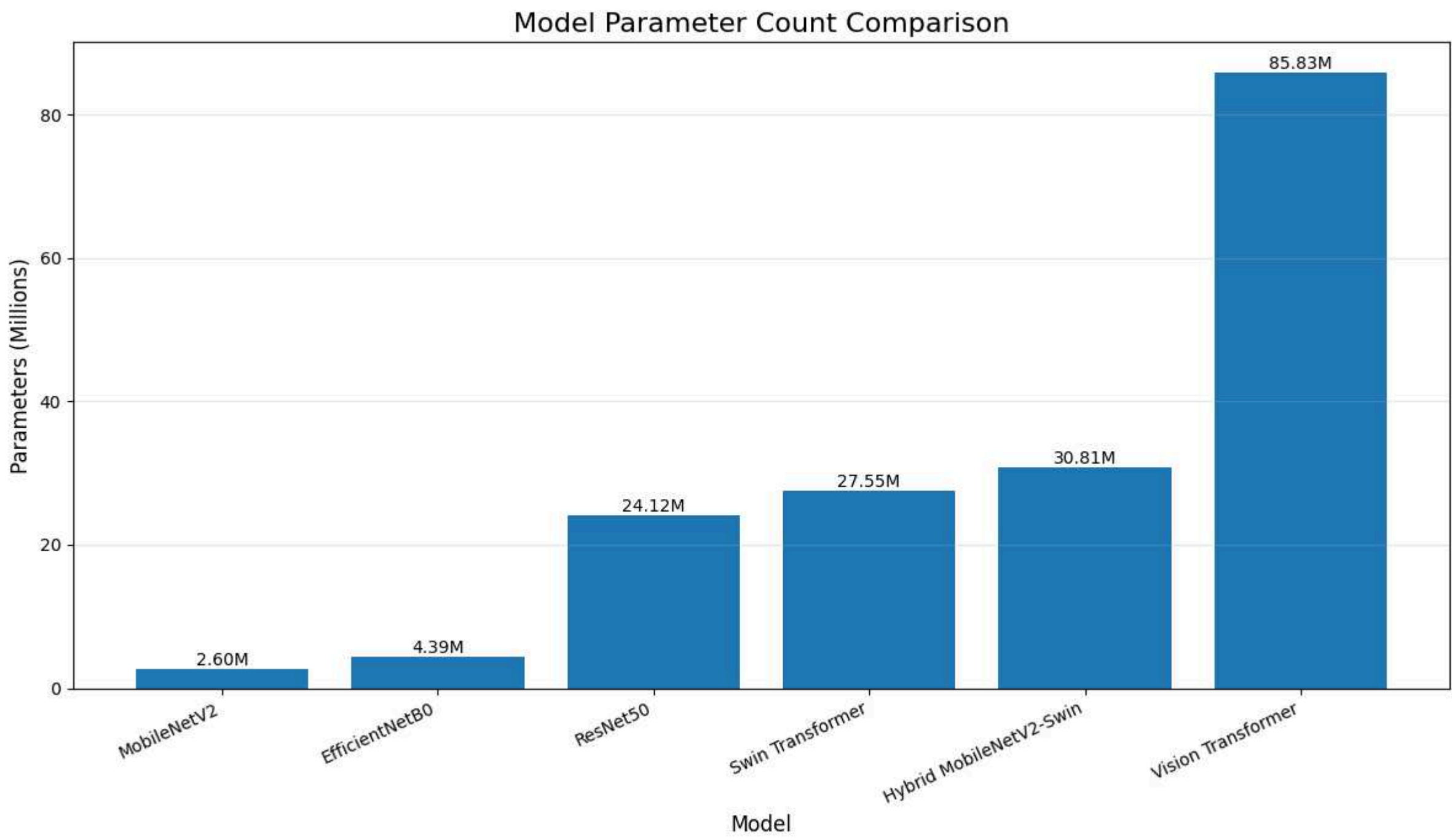
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f"{value:.2f}M",
        ha="center",
        va="bottom",
        fontsize=10
    )

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
)

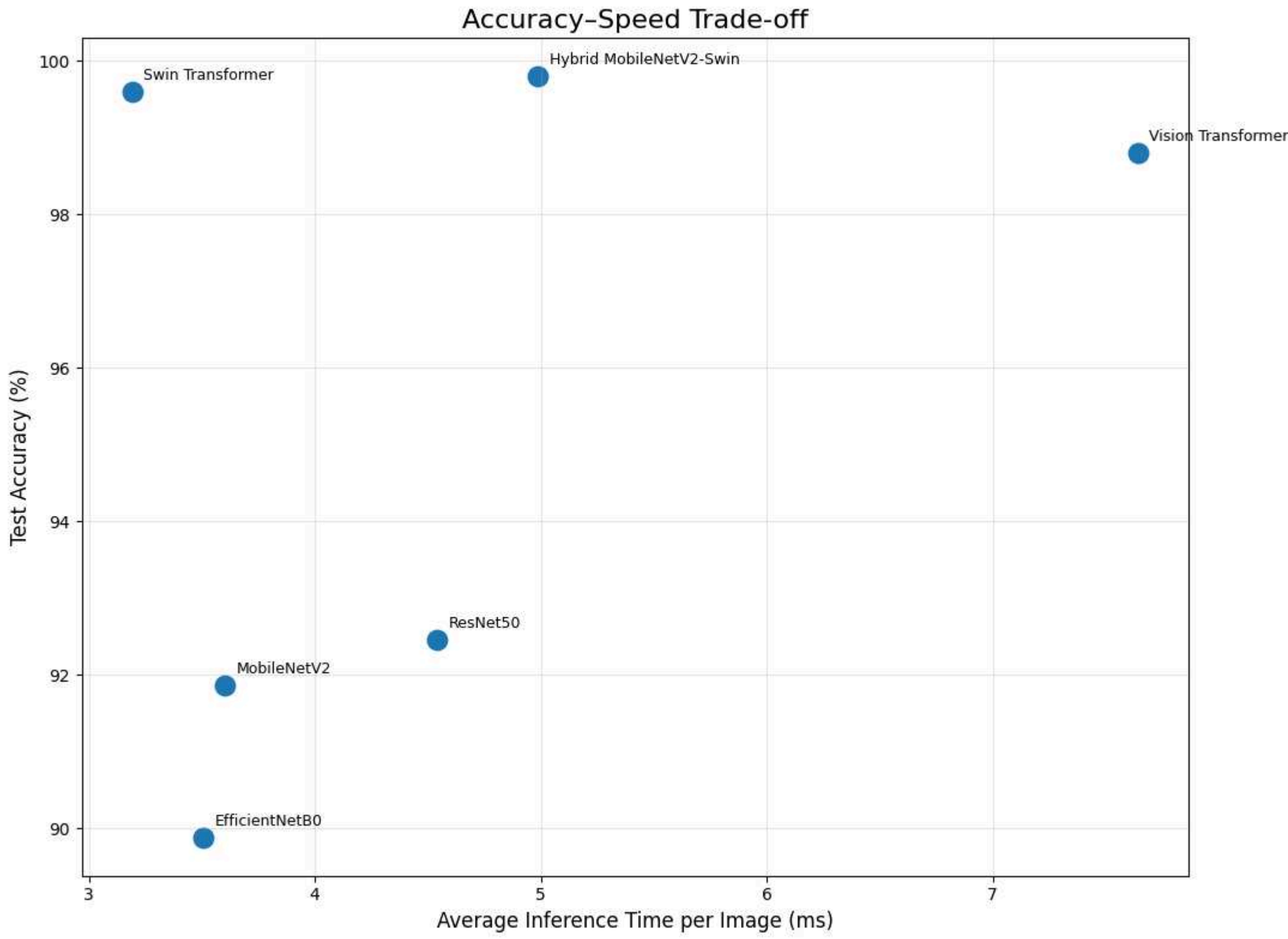
plt.grid(
    alpha=0.3
)

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
=====
```

In [1]:

```
# =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the saved models,
# class names, metrics files, and application resources.
# =====

from google.colab import drive

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

Mounted at /content/drive

In [2]:

```
# =====
# Step 2: Install Required Libraries
# This step installs Gradio, TIMM, and TextBlob for the
# multi-model ASL web application and spelling correction.
# =====

!pip -q install --upgrade gradio
!pip -q install timm
!pip -q install textblob

print("=" * 60)
print("Required Libraries Installed Successfully")
print("=" * 60)
```

```

[?]251  [90m-----[0m [32m0.0/31.0 MB[0m [31m?[0m eta [36m:--:--[0m
[2K  [91m-----[0m[90m-[0m[90m-----[0m [32m9.4/31.0 MB[0m [31m282.1 MB/s[0m eta [36m0:00:01[0m
[2K  [91m-----[0m[90m-[0m[90m-----[0m [32m19.6/31.0 MB[0m [31m290.1 MB/s[0m eta [36m0:00:01[0m
[2K  [91m-----[0m[90m-[0m[90m-[0m[90m-[0m [32m29.8/31.0 MB[0m [31m291.0 MB/s[0m eta [36m0:00:01[0m
[2K  [91m-----[0m[91m-[0m [32m31.0/31.0 MB[0m [31m294.0 MB/s[0m eta [36m0:00:01[0m
[2K  [91m-----[0m[91m-[0m [32m31.0/31.0 MB[0m [31m294.0 MB/s[0m eta [36m0:00:01[0m
[2K  [90m-----[0m [32m31.0/31.0 MB[0m [31m94.9 MB/s[0m eta [36m0:00:00[0m
[?]25h=====
Required Libraries Installed Successfully
=====
```

In [3]:

```
# =====
# Step 3: Import Required Libraries
# This step imports all libraries required for loading models,
# preprocessing images, prediction, spelling correction,
# metrics display, and Gradio interface development.
# =====

import os
import json
import gc
import warnings

import numpy as np
import pandas as pd

from PIL import Image

import tensorflow as tf

import torch
import torch.nn as nn

from torchvision import transforms

import timm
import gradio as gr

from textblob import TextBlob

from tensorflow.keras.applications.resnet50 import (
    preprocess_input as resnet50_preprocess
)

from tensorflow.keras.applications.mobilenet_v2 import (
    preprocess_input as mobilenetv2_preprocess
)

from tensorflow.keras.applications.efficientnet import (
    preprocess_input as efficientnetb0_preprocess
)

warnings.filterwarnings("ignore")

print("=" * 60)
print("Libraries Imported Successfully")
print("=" * 60)
print(f"Gradio Version      : {gr.__version__}")
print(f"TensorFlow Version: {tf.__version__}")
print(f"PyTorch Version      : {torch.__version__}")
print(f"TIMM Version         : {timm.__version__}")
print("=" * 60)
```

```

=====
Libraries Imported Successfully
=====
Gradio Version      : 6.20.0
TensorFlow Version: 2.20.0
PyTorch Version     : 2.11.0+cu128
TIMM Version        : 1.0.27
=====
```

In [4]:

```
# =====
# Step 4: Configure Device and Application Settings
# This step configures the GPU or CPU device, image size,
# class labels, and default application settings.
# =====

IMAGE_SIZE = 224

CLASS_NAMES = [
    "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"
]

NUM_CLASSES = len(CLASS_NAMES)

TORCH_DEVICE = torch.device(
    "cuda"
    if torch.cuda.is_available()
    else "cpu"
)

tensorflow_gpu = tf.config.list_physical_devices("GPU")

print("=" * 60)
print("Application Configuration")
print("=" * 60)
print(f"Image Size           : {IMAGE_SIZE} x {IMAGE_SIZE}")
print(f"Number of Classes     : {NUM_CLASSES}")
print(f"PyTorch Device        : {TORCH_DEVICE}")
print(f"TensorFlow GPU Found  : {'Yes' if tensorflow_gpu else 'No'}")

if torch.cuda.is_available():
    print(
        "GPU Name           :",
        torch.cuda.get_device_name(0)
    )
else:
    print("GPU Name           : CPU")

print("=" * 60)
```

```

=====
Application Configuration
=====
Image Size           : 224 x 224
Number of Classes     : 36
PyTorch Device        : cuda
TensorFlow GPU Found  : Yes
GPU Name             : NVIDIA A100-SXM4-40GB
=====
```

```
In [5]: # =====
# Step 5: Define Model, Class, and Metrics Paths
# This step defines the saved files used by all six models
# and verifies that the required application files exist.
# =====

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

MODELS_DIR = os.path.join(
    PROJECT_DIR,
    "02_Models"
)

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

MODEL_CONFIGS = {
    "ResNet50": {
        "framework": "tensorflow",
        "model_path": os.path.join(
            MODELS_DIR,
            "ResNet50",
            "final_resnet50_asl_model.keras"
        ),
        "class_names_path": os.path.join(
            MODELS_DIR,
            "ResNet50",
            "asl_class_names.txt"
        ),
        "metrics_path": os.path.join(
            METRICS_DIR,
            "resnet50_metrics.json"
        )
    },

    "MobileNetV2": {
        "framework": "tensorflow",
        "model_path": os.path.join(
            MODELS_DIR,
            "MobileNetV2",
            "final_mobilenetv2_asl_model.keras"
        ),
        "class_names_path": os.path.join(
            MODELS_DIR,
            "MobileNetV2",
            "mobilenetv2_class_names.txt"
        ),
        "metrics_path": os.path.join(
            METRICS_DIR,
            "mobilenetv2_metrics.json"
        )
    },

    "EfficientNetB0": {
        "framework": "tensorflow",
        "model_path": os.path.join(
            MODELS_DIR,
            "EfficientNetB0",
            "final_efficientnetb0_asl_model.keras"
        ),
        "class_names_path": os.path.join(
            MODELS_DIR,
            "EfficientNetB0",
            "efficientnetb0_class_names.txt"
        ),
        "metrics_path": os.path.join(
            METRICS_DIR,
            "efficientnetb0_metrics.json"
        )
    },

    "Vision Transformer": {
        "framework": "pytorch",
        "architecture": "vit_base_patch16_224",
        "model_path": os.path.join(
            MODELS_DIR,
            "VisionTransformer",
            "final_vit_asl_model.pth"
        ),
        "class_names_path": os.path.join(
            MODELS_DIR,
            "VisionTransformer",
            "vit_class_names.txt"
        ),
        "metrics_path": os.path.join(
            METRICS_DIR,
            "vit_metrics.json"
        )
    },

    "Swin Transformer": {
        "framework": "pytorch",
        "architecture": "swin_tiny_patch4_window7_224",
        "model_path": os.path.join(
            MODELS_DIR,
            "SwinTransformer",
            "final_swin_asl_model.pth"
        ),
        "class_names_path": os.path.join(
            MODELS_DIR,
            "SwinTransformer",
            "swin_class_names.txt"
        ),
        "metrics_path": os.path.join(
            METRICS_DIR,
            "swin_metrics.json"
        )
    },

    "Hybrid MobileNetV2-Swin": {
        "framework": "hybrid",
        "model_path": os.path.join(
            MODELS_DIR,
            "Hybrid",
            "final_hybrid_mobilenetv2_swin_asl_model.pth"
        ),
        "class_names_path": os.path.join(
            MODELS_DIR,
            "Hybrid",
            "hybrid_mobilenetv2_swin_class_names.txt"
        ),
        "metrics_path": os.path.join(
            METRICS_DIR,
            "hybrid_metrics.json"
        )
    }
}

missing_required_files = []

print("=" * 80)
print("Application Files Verification")
print("=" * 80)

for model_name, config in MODEL_CONFIGS.items():

    print(f"\nModel: {model_name}")

    model_exists = os.path.isfile(
        config["model_path"]
    )

    classes_exist = os.path.isfile(
        config["class_names_path"]
    )

    metrics_exist = os.path.isfile(
        config["metrics_path"]
    )

    print(
        f"Model File   : "
        f"{'Found' if model_exists else 'Missing'}"
    )

    print(
        f"Class Names   : "
        f"{'Found' if classes_exist else 'Missing'}"
    )

    print(
        f"Metrics File  : "
        f"{'Found' if metrics_exist else 'Missing'}"
    )
```

```

    if not model_exists:
        missing_required_files.append(
            config["model_path"]
        )

    if not classes_exist:
        missing_required_files.append(
            config["class_names_path"]
        )

    if not metrics_exist:
        missing_required_files.append(
            config["metrics_path"]
        )

print("\n" + "=" * 80)

if missing_required_files:

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

    for missing_file in missing_required_files:
        print(f"- {missing_file}")

    raise FileNotFoundError(
        "Some required application files are missing. "
        "Check the paths displayed above."
    )

else:

    print("Application Status: READY")
    print("All model, class, and metrics files were found.")

print("=" * 80)
```

```

=====
Application Files Verification
=====

Model: ResNet50
Model File   : Found
Class Names  : Found
Metrics File : Found

Model: MobileNetV2
Model File   : Found
Class Names  : Found
Metrics File : Found

Model: EfficientNetB0
Model File   : Found
Class Names  : Found
Metrics File : Found

Model: Vision Transformer
Model File   : Found
Class Names  : Found
Metrics File : Found

Model: Swin Transformer
Model File   : Found
Class Names  : Found
Metrics File : Found

Model: Hybrid MobileNetV2-Swin
Model File   : Found
Class Names  : Found
Metrics File : Found

=====
Application Status: READY
All model, class, and metrics files were found.
=====
```

In [6]:

```

# =====
# Step 6: Load Class Names and Model Metrics
# This step loads the class names and evaluation metrics
# associated with each trained ASL recognition model.
# =====

MODEL_CLASS_NAMES = {}
MODEL_METRICS = {}

def load_class_names(
    class_names_path
):

    with open(
        class_names_path,
        "r"
    ) as class_file:

        loaded_class_names = [
            line.strip()
            for line in class_file.readlines()
            if line.strip()
        ]

    if len(loaded_class_names) != NUM_CLASSES:

        raise ValueError(
            f"Expected {NUM_CLASSES} class names, "
            f"but found {len(loaded_class_names)} in:\n"
            f"{class_names_path}"
        )

    return loaded_class_names

for model_name, config in MODEL_CONFIGS.items():

    MODEL_CLASS_NAMES[model_name] = (
        load_class_names(
            config["class_names_path"]
        )
    )

    with open(
        config["metrics_path"],
        "r"
    ) as metrics_file:

        MODEL_METRICS[model_name] = json.load(
            metrics_file
        )

print("=" * 70)
print("Class Names and Model Metrics Loaded Successfully")
print("=" * 70)

for model_name in MODEL_CONFIGS:

    accuracy = (
        MODEL_METRICS[model_name]["test_accuracy"]
        * 100
    )

    print(
        f"{model_name:<28}: "
        f"{len(MODEL_CLASS_NAMES[model_name])} classes | "
        f"Accuracy: {accuracy:.2f}%"
    )

print("=" * 70)
```

```

=====
Class Names and Model Metrics Loaded Successfully
=====
ResNet50           : 36 classes | Accuracy: 92.46%
MobileNetV2        : 36 classes | Accuracy: 91.87%
EfficientNetB0      : 36 classes | Accuracy: 89.88%
Vision Transformer  : 36 classes | Accuracy: 98.81%
Swin Transformer    : 36 classes | Accuracy: 99.60%
Hybrid MobileNetV2-Swin : 36 classes | Accuracy: 99.80%
=====
```

```
In [7]: # =====
# Step 7: Define the Hybrid Model Architecture
# This step recreates the Hybrid MobileNetV2-Swin architecture
# required to load and use the saved model weights.
# =====

class HybridMobileNetSwin(nn.Module):

    def __init__(
        self,
        num_classes
    ):

        super().__init__()

        self.mobilenet = timm.create_model(
            "mobilenetv2_100",
            pretrained=False,
            num_classes=0,
            global_pool="avg"
        )

        self.swin = timm.create_model(
            "swin_tiny_patch4_window7_224",
            pretrained=False,
            num_classes=0,
            global_pool="avg"
        )

        mobilenet_features = (
            self.mobilenet.num_features
        )

        swin_features = (
            self.swin.num_features
        )

        combined_features = (
            mobilenet_features
            + swin_features
        )

        self.classifier = nn.Sequential(
            nn.Linear(
                combined_features,
                512
            ),

            nn.ReLU(),

            nn.Dropout(0.5),

            nn.Linear(
                512,
                num_classes
            )
        )

    def forward(
        self,
        images
    ):

        mobilenet_features = self.mobilenet(
            images
        )

        swin_features = self.swin(
            images
        )

        fused_features = torch.cat(
            (
                mobilenet_features,
                swin_features
            ),
            dim=1
        )

        outputs = self.classifier(
            fused_features
        )

        return outputs

print("=" * 60)
print("Hybrid Model Architecture Defined Successfully")
print("=" * 60)

=====
Hybrid Model Architecture Defined Successfully
=====
```

```
In [8]: # =====
# Step 8: Create Image Preprocessing Functions
# This step creates the TensorFlow and PyTorch preprocessing
# pipelines required by the six ASL recognition models.
# =====

PYTORCH_IMAGE_TRANSFORM = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[
            0.485,
            0.456,
            0.406
        ],

        std=[
            0.229,
            0.224,
            0.225
        ]
    )
])

def prepare_tensorflow_image(
    image,
    model_name
):

    if not isinstance(
        image,
        Image.Image
    ):

        image = Image.fromarray(
            np.asarray(image)
        )

    image = image.convert(
        "RGB"
    )

    image = image.resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    )

    image_array = np.asarray(
        image,
        dtype=np.float32
    )

    image_array = np.expand_dims(
        image_array,
        axis=0
    )

    if model_name == "ResNet50":

        image_array = resnet50_preprocess(
            image_array
        )

    elif model_name == "MobileNetV2":

        image_array = mobilenetv2_preprocess(
            image_array
        )

    elif model_name == "EfficientNetB0":

        image_array = efficientnetb0_preprocess(
            image_array
        )

    else:

        raise ValueError(
            f"Unsupported TensorFlow model: "
            f"{model_name}"
        )

    return image_array

def prepare_pytorch_image(
    image
):

    if not isinstance(
        image,
        Image.Image
    ):

        image = Image.fromarray(
            np.asarray(image)
        )

    image = image.convert(
        "RGB"
    )

    image_tensor = PYTORCH_IMAGE_TRANSFORM(
        image
    )

    image_tensor = image_tensor.unsqueeze(
        0
    )

    image_tensor = image_tensor.to(
        TORCH_DEVICE
    )

    return image_tensor

print("=" * 60)
print("Image Preprocessing Functions Created Successfully")
print("=" * 60)
```

```
=====
Image Preprocessing Functions Created Successfully
=====
```

```
In [9]: # =====
# Step 9: Create the Smart Model Loader
# This step loads only the selected model and releases the
# previous model to reduce RAM and GPU memory consumption.
# =====

ACTIVE_MODEL = None
ACTIVE_MODEL_NAME = None
ACTIVE_FRAMEWORK = None

def clear_active_model():

    global ACTIVE_MODEL
    global ACTIVE_MODEL_NAME
    global ACTIVE_FRAMEWORK

    ACTIVE_MODEL = None
    ACTIVE_MODEL_NAME = None
    ACTIVE_FRAMEWORK = None

    tf.keras.backend.clear_session()

    gc.collect()

    if torch.cuda.is_available():
        torch.cuda.empty_cache()

def load_torch_checkpoint(
    checkpoint_path
):

    try:

        checkpoint = torch.load(
            checkpoint_path,
            map_location="cpu",
            weights_only=True
        )

    except TypeError:

        checkpoint = torch.load(
            checkpoint_path,
            map_location="cpu"
        )

    if (
        isinstance(checkpoint, dict)
        and "state_dict" in checkpoint
    ):

        checkpoint = checkpoint[
            "state_dict"
        ]

    cleaned_checkpoint = {}

    for key, value in checkpoint.items():

        cleaned_key = key

        if cleaned_key.startswith(
            "module."
        ):

            cleaned_key = cleaned_key[
                len("module.") :
            ]

        cleaned_checkpoint[
            cleaned_key
        ] = value

    return cleaned_checkpoint

def create_pytorch_model(
    model_name
):

    if model_name == "Vision Transformer":

        pytorch_model = timm.create_model(
            "vit_base_patch16_224",
            pretrained=False,
            num_classes=NUM_CLASSES
        )

    elif model_name == "Swin Transformer":

        pytorch_model = timm.create_model(
            "swin_tiny_patch4_window7_224",
            pretrained=False,
            num_classes=NUM_CLASSES
        )

    elif model_name == (
        "Hybrid MobileNetV2-Swin"
    ):

        pytorch_model = HybridMobileNetSwin(
            num_classes=NUM_CLASSES
        )

    else:

        raise ValueError(
            f"Unsupported PyTorch model: "
            f"{model_name}"
        )

    return pytorch_model

def load_selected_model(
    model_name
):

    global ACTIVE_MODEL
    global ACTIVE_MODEL_NAME
    global ACTIVE_FRAMEWORK

    if (
        ACTIVE_MODEL is not None
        and ACTIVE_MODEL_NAME == model_name
    ):

        return (
            ACTIVE_MODEL,
            ACTIVE_FRAMEWORK
        )

    if model_name not in MODEL_CONFIGS:

        raise ValueError(
            f"Unknown model: {model_name}"
        )

    clear_active_model()

    config = MODEL_CONFIGS[
        model_name
    ]

    framework = config[
        "framework"
    ]

    model_path = config[
        "model_path"
    ]

    print(
        f>Loading {model_name}..."
    )

    if framework == "tensorflow":

        loaded_model = tf.keras.models.load_model(
            model_path,
            compile=False
        )

    elif framework in [
        "pytorch",
        "hybrid"
    ]:
```

```
loaded_model = create_pytorch_model(
    model_name
)

state_dictionary = load_torch_checkpoint(
    model_path
)

loaded_model.load_state_dict(
    state_dictionary,
    strict=True
)

loaded_model = loaded_model.to(
    TORCH_DEVICE
)

loaded_model.eval()

else:

    raise ValueError(
        f"Unsupported framework: "
        f"{framework}"
    )

ACTIVE_MODEL = loaded_model
ACTIVE_MODEL_NAME = model_name
ACTIVE_FRAMEWORK = framework

print(
    f"{model_name} Loaded Successfully"
)

return (
    ACTIVE_MODEL,
    ACTIVE_FRAMEWORK
)

print("=" * 60)
print("Smart Model Loader Created Successfully")
print("=" * 60)
```

```
=====
Smart Model Loader Created Successfully
=====
```

In [10]:

```
# =====
# Step 10: Create Prediction Functions
# This step predicts the ASL class of an uploaded image and
# returns the predicted letter, confidence, and top results.
# =====

def predict_probabilities(
    image,
    model_name
):

    if image is None:

        raise ValueError(
            "Please upload an image or use the camera."
        )

    if model_name not in MODEL_CONFIGS:

        raise ValueError(
            "Please select a valid model."
        )

    model, framework = load_selected_model(
        model_name
    )

    model_class_names = MODEL_CLASS_NAMES[
        model_name
    ]

    if framework == "tensorflow":

        prepared_image = (
            prepare_tensorflow_image(
                image,
                model_name
            )
        )

        probabilities = model.predict(
            prepared_image,
            verbose=0
        )[0]

    else:

        prepared_image = (
            prepare_pytorch_image(
                image
            )
        )

        model.eval()

        with torch.inference_mode():

            logits = model(
                prepared_image
            )

            probabilities = torch.softmax(
                logits,
                dim=1
            )[0]

            probabilities = (
                probabilities
                .detach()
                .cpu()
                .numpy()
            )

        probabilities = np.asarray(
            probabilities,
            dtype=np.float64
        )

    if len(probabilities) != len(
        model_class_names
    ):

        raise ValueError(
            "The number of model outputs does not match "
            "the number of class names."
        )

    return (
        probabilities,
        model_class_names
    )

def predict_single_image(
    image,
    model_name
):

    try:

        probabilities, model_class_names = (
            predict_probabilities(
                image,
                model_name
            )
        )

        predicted_index = int(
            np.argmax(probabilities)
        )

        predicted_letter = (
            model_class_names[
                predicted_index
            ].upper()
        )

        confidence = float(
            probabilities[
                predicted_index
            ]
        )

        top_indices = np.argsort(
            probabilities
        )[-5:][::-1]

        top_predictions = {
            model_class_names[
                index
            ].upper():
            float(
                probabilities[index]
            )
            for index in top_indices
        }

        status_message = (
            f"Prediction completed successfully "
            f"using {model_name}."
        )

        return (
            top_predictions,
            predicted_letter,
            f"{confidence * 100:.2f}%",
            status_message
        )

    except Exception as error:

        return (
            {},
            "",
            "",
            f"Prediction Error: {error}"
        )

print("=" * 60)
print("Prediction Functions Created Successfully")
print("=" * 60)
```

```
=====
Prediction Functions Created Successfully
=====
```

```
In [11]: # =====
# Step 11: Create Word Formation and Spelling Correction
# This step creates functions to build a word from predicted
# letters, remove letters, clear the word, and correct spelling.
# =====

def correct_spelling(
    raw_word
):
    raw_word = (
        raw_word or ""
    ).strip()

    if not raw_word:
        return ""

    # Do not modify sequences containing numbers.
    if any(
        character.isdigit()
        for character in raw_word
    ):
        return raw_word.upper()

    try:
        corrected_word = str(
            TextBlob(
                raw_word.lower()
            ).correct()
        )

        return corrected_word.upper()

    except Exception:
        return raw_word.upper()

def append_predicted_letter(
    predicted_letter,
    current_word
):
    predicted_letter = (
        predicted_letter or ""
    ).strip().upper()

    current_word = (
        current_word or ""
    ).strip().upper()

    if not predicted_letter:

        corrected_word = correct_spelling(
            current_word
        )

        return (
            current_word,
            current_word,
            corrected_word,
            "No predicted letter is available."
        )

    updated_word = (
        current_word
        + predicted_letter
    )

    corrected_word = correct_spelling(
        updated_word
    )

    return (
        updated_word,
        updated_word,
        corrected_word,
        f"Letter {predicted_letter} added successfully."
    )

def remove_last_letter(
    current_word
):
    current_word = (
        current_word or ""
    ).strip().upper()

    updated_word = current_word[:-1]

    corrected_word = correct_spelling(
        updated_word
    )

    return (
        updated_word,
        updated_word,
        corrected_word,
        "Last letter removed."
    )

def clear_current_word():
    return (
        "",
        "",
        "",
        "Word cleared successfully."
    )

def correct_current_word(
    current_word
):
    current_word = (
        current_word or ""
    ).strip().upper()

    corrected_word = correct_spelling(
        current_word
    )

    return (
        corrected_word,
        "Spelling correction completed."
    )

print("=" * 60)
print("Word Formation and Spelling Functions Created Successfully")
print("=" * 60)
```

```
=====
Word Formation and Spelling Functions Created Successfully
=====
```

```
In [12]: # =====
# Step 12: Create Multiple-Image Word Prediction
# This step predicts multiple uploaded images, combines their
# predicted classes, and applies spelling correction.
# =====

def get_uploaded_file_path(
    uploaded_file
):
    if isinstance(
        uploaded_file,
        str
    ):
        return uploaded_file

    if hasattr(
        uploaded_file,
        "name"
    ):
        return uploaded_file.name

    if hasattr(
        uploaded_file,
        "path"
    ):
        return uploaded_file.path

    raise ValueError(
        "Unsupported uploaded file format."
    )

def predict_multiple_images(
    uploaded_files,
    model_name,
    progress=gr.Progress()
):
    if not uploaded_files:
        return (
            "",
            "",
            [],
            "Please upload one or more ASL images."
        )

    predicted_letters = []
    gallery_results = []

    total_files = len(
        uploaded_files
    )

    for image_index, uploaded_file in enumerate(
        uploaded_files,
        start=1
    ):
        progress(
            image_index / total_files,
            desc=(
                f"Processing image "
                f"{image_index} of {total_files}"
            )
        )

        image_path = get_uploaded_file_path(
            uploaded_file
        )

        image = Image.open(
            image_path
        ).convert(
            "RGB"
        )

        (
            probabilities,
            model_class_names
        ) = predict_probabilities(
            image,
            model_name
        )

        predicted_index = int(
            np.argmax(
                probabilities
            )
        )

        predicted_letter = (
            model_class_names[
                predicted_index
            ].upper()
        )

        confidence = float(
            probabilities[
                predicted_index
            ]
        )

        predicted_letters.append(
            predicted_letter
        )

        caption = (
            f"Image {image_index}: "
            f"{predicted_letter} "
            f"({confidence * 100:.2f}%)"
        )

        gallery_results.append(
            (
                image,
                caption
            )
        )

    raw_word = "".join(
        predicted_letters
    )

    corrected_word = correct_spelling(
        raw_word
    )

    status_message = (
        f"{total_files} images processed successfully "
        f"using {model_name}."
    )

    return (
        raw_word,
        corrected_word,
        gallery_results,
        status_message
    )

print("=" * 60)
print("Multiple-Image Prediction Created Successfully")
print("=" * 60)
```

```
=====
Multiple-Image Prediction Created Successfully
=====
```

In [13]:

```
# =====
# Step 13: Create Model Information Functions
# This step creates automatic model information and comparison
# tables using the saved metrics JSON files.
# =====

def create_model_information(
    model_name
):
    if model_name not in MODEL_METRICS:
        return (
            "### Model information is unavailable."
        )

    metrics = MODEL_METRICS[
        model_name
    ]

    accuracy = (
        metrics["test_accuracy"]
        * 100
    )

    precision = (
        metrics["weighted_precision"]
        * 100
    )

    recall = (
        metrics["weighted_recall"]
        * 100
    )

    f1_score_value = (
        metrics["weighted_f1_score"]
        * 100
    )

    time_per_image_ms = (
        metrics[
            "average_time_per_image_seconds"
        ]
        * 1000
    )

    parameters = int(
        metrics["parameters"]
    )

    model_size = float(
        metrics["model_size_mb"]
    )

    test_images = int(
        metrics["test_images"]
    )

    gpu_name = metrics.get(
        "gpu",
        "Not recorded"
    )

    framework = metrics.get(
        "framework",
        "Not recorded"
    )

    model_information = f"""
### Selected Model Information

| Metric | Value |
|---|---|
| **Model** | {model_name} |
| **Framework** | {framework} |
| **Test Images** | {test_images:,} |
| **Test Accuracy** | {accuracy:.2f}% |
| **Weighted Precision** | {precision:.2f}% |
| **Weighted Recall** | {recall:.2f}% |
| **Weighted F1-Score** | {f1_score_value:.2f}% |
| **Test Loss** | {metrics["test_loss"]:.4f} |
| **Time per Image** | {time_per_image_ms:.3f} ms |
| **Parameters** | {parameters:,} |
| **Model Size** | {model_size:.2f} MB |
| **Evaluation Device** | {gpu_name} |

**Test Accuracy** is the model's performance on the complete test dataset.
**Prediction Confidence** is the confidence for the currently uploaded image.
"""

    return model_information

def create_models_comparison_table():

    comparison_rows = []

    for model_name, metrics in MODEL_METRICS.items():

        comparison_rows.append({
            "Model": model_name,

            "Accuracy (%)": round(
                metrics["test_accuracy"]
                * 100,
                2
            ),

            "Precision (%)": round(
                metrics["weighted_precision"]
                * 100,
                2
            ),

            "Recall (%)": round(
                metrics["weighted_recall"]
                * 100,
                2
            ),

            "F1-Score (%)": round(
                metrics["weighted_f1_score"]
                * 100,
                2
            ),

            "Test Loss": round(
                metrics["test_loss"],
                4
            ),

            "Time/Image (ms)": round(
                metrics[
                    "average_time_per_image_seconds"
                ]
                * 1000,
                3
            ),

            "Parameters": int(
                metrics["parameters"]
            ),

            "Model Size (MB)": round(
                metrics["model_size_mb"],
                2
            )
        })

    comparison_table = pd.DataFrame(
        comparison_rows
    )

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

    comparison_table.insert(
        0,
        "Rank",
        np.arange(
            1,
```

```
        len(comparison_table) + 1
    )
)

return comparison_table

def handle_model_change(
    model_name
):
    return (
        create_model_information(
            model_name
        ),
        (
            f"{model_name} selected. "
            f"The model will load when prediction starts."
        )
    )

print("=" * 60)
print("Model Information Functions Created Successfully")
print("=" * 60)

display(
    create_models_comparison_table()
)
```

=====
Model Information Functions Created Successfully
=====

	Rank	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Test Loss	Time/Image (ms)	Parameters	Model Size (MB)
0	1	Hybrid MobileNetV2-Swin	99.80	99.81	99.80	99.80	0.0112	4.986	30810782	117.86
1	2	Swin Transformer	99.60	99.60	99.60	99.60	0.0189	3.191	27547038	105.15
2	3	Vision Transformer	98.81	98.83	98.81	98.81	0.0427	7.642	85826340	327.46
3	4	ResNet50	92.46	93.36	92.46	92.42	0.3774	4.542	24121508	96.72
4	5	MobileNetV2	91.87	92.86	91.87	91.93	0.3565	3.600	2595172	13.03
5	6	EfficientNetB0	89.88	91.42	89.88	89.51	0.5200	3.504	4386759	20.11

```
In [14]: # =====
# Step 14: Define Application Styling
# This step defines the visual appearance of the multi-model
# ASL recognition web application.
# =====

CUSTOM_CSS = """
.gradio-container {
    max-width: 1280px !important;
    margin: auto !important;
}

#main-title {
    text-align: center;
    margin-bottom: 5px;
}

#main-subtitle {
    text-align: center;
    opacity: 0.85;
    margin-bottom: 22px;
}

.predicted-letter textarea {
    font-size: 34px !important;
    font-weight: bold !important;
    text-align: center !important;
}

.confidence-box textarea {
    font-size: 24px !important;
    font-weight: bold !important;
    text-align: center !important;
}

.word-output textarea {
    font-size: 28px !important;
    font-weight: bold !important;
    text-align: center !important;
}

.status-output textarea {
    font-size: 15px !important;
}

.footer-text {
    text-align: center;
    opacity: 0.75;
    margin-top: 25px;
}
"""

APPLICATION_TITLE = (
    "ASL Multi-Model Recognition System"
)

APPLICATION_DESCRIPTION = """
Select one of the trained models, upload an ASL image or use
the camera, and view the predicted letter with its confidence.
You may also combine several predictions into a word and apply
automatic English spelling correction.
"""

print("=" * 60)
print("Application Styling Created Successfully")
print("=" * 60)
```

=====
Application Styling Created Successfully
=====

In [15]:

```
# =====
# Step 15: Build the Gradio Web Application
# This step creates the complete multi-model ASL recognition
# website with image, camera, word, batch, and comparison modes.
# =====

with gr.Blocks(
    title=APPLICATION_TITLE,
    css=CUSTOM_CSS
) as demo:

    current_word_state = gr.State(
        value=""
    )

    gr.Markdown(
        f"""
# {APPLICATION_TITLE}
### American Sign Language Classification, Word Formation, and Spelling Correction
"""
    ,
    elem_id="main-title"
    )

    gr.Markdown(
        APPLICATION_DESCRIPTION,
        elem_id="main-subtitle"
    )

    with gr.Row():

        model_selector = gr.Dropdown(
            choices=list(
                MODEL_CONFIGS.keys()
            ),
            value="Swin Transformer",
            label="Select Recognition Model",
            interactive=True,
            scale=2
        )

        application_status = gr.Textbox(
            value="Application is ready.",
            label="Application Status",
            interactive=False,
            scale=2,
            elem_classes=[
                "status-output"
            ]
        )

        model_information = gr.Markdown(
            value=create_model_information(
                "Swin Transformer"
            )
        )

    with gr.Tabs():

        # =====
        # Tab 1: Single Image or Camera
        # =====

        with gr.Tab(
            "Single Image / Camera"
        ):

            with gr.Row():

                image_input = gr.Image(
                    sources=[
                        "upload",
                        "webcam"
                    ],
                    type="pil",
                    label="Upload an ASL Image or Use Camera",
                    height=430
                )

                with gr.Column():

                    top_predictions_output = gr.Label(
                        label="Top Five Predictions",
                        num_top_classes=5
                    )

                    predicted_letter_output = gr.Textbox(
                        label="Predicted Letter",
                        interactive=False,
                        elem_classes=[
                            "predicted-letter"
                        ]
                    )

                    confidence_output = gr.Textbox(
                        label="Prediction Confidence",
                        interactive=False,
                        elem_classes=[
                            "confidence-box"
                        ]
                    )

                    predict_button = gr.Button(
                        "Predict Image",
                        variant="primary",
                        size="lg"
                    )

                gr.Markdown(
                    "## Word Formation"
                )

            with gr.Row():

                raw_word_output = gr.Textbox(
                    label="Raw Predicted Word",
                    interactive=False,
                    elem_classes=[
                        "word-output"
                    ]
                )

                corrected_word_output = gr.Textbox(
                    label="Corrected Word",
                    interactive=False,
                    elem_classes=[
                        "word-output"
                    ]
                )

            with gr.Row():

                append_letter_button = gr.Button(
                    "Append Predicted Letter",
                    variant="primary"
                )

                remove_letter_button = gr.Button(
                    "Remove Last Letter"
                )

                correct_word_button = gr.Button(
                    "Correct Word"
                )

                clear_word_button = gr.Button(
                    "Clear Word",
                    variant="stop"
                )

        # =====
        # Tab 2: Multiple Images to Word
        # =====

        with gr.Tab(
            "Multiple Images to Word"
        ):

            gr.Markdown(
                """
Upload multiple ASL images in the correct letter order.
Each image will be classified, and the predicted letters will
be joined into one word.
"""
            )

            multiple_images_input = gr.File(
                label="Upload Ordered ASL Images",
                file_count="multiple",
                file_types=["image"],
                type="filepath"
            )

```

```
)

multiple_predict_button = gr.Button(
    "Predict Complete Word",
    variant="primary",
    size="lg"
)

with gr.Row():

    multiple_raw_word_output = gr.Textbox(
        label="Raw Predicted Word",
        interactive=False,
        elem_classes=[
            "word-output"
        ]
    )

    multiple_corrected_word_output = gr.Textbox(
        label="Corrected Word",
        interactive=False,
        elem_classes=[
            "word-output"
        ]
    )

    multiple_gallery_output = gr.Gallery(
        label="Individual Image Predictions",
        columns=4,
        rows=2,
        height="auto",
        object_fit="contain"
    )

    multiple_status_output = gr.Textbox(
        label="Multiple-Image Prediction Status",
        interactive=False,
        elem_classes=[
            "status-output"
        ]
    )

# =====
# Tab 3: Models Comparison
# =====

with gr.Tab(
    "Models Comparison"
):

    gr.Markdown(
        """
## Saved Model Performance

This table is generated automatically from the six JSON
metrics files created after model training and evaluation.
        """
    )

    comparison_table_output = gr.Dataframe(
        value=create_models_comparison_table(),
        label="ASL Models Comparison",
        interactive=False,
        wrap=True
    )

    gr.Markdown(
        """
---
**ASL Recognition Research Application**
Static image classification for 36 classes: A-Z and 0-9.
        """
    )

    elem_classes=[
        "footer-text"
    ]

# =====
# Event Connections
# =====

model_selector.change(
    fn=handle_model_change,
    inputs=[
        model_selector
    ],
    outputs=[
        model_information,
        application_status
    ]
)

predict_button.click(
    fn=predict_single_image,
    inputs=[
        image_input,
        model_selector
    ],
    outputs=[
        top_predictions_output,
        predicted_letter_output,
        confidence_output,
        application_status
    ]
)

append_letter_button.click(
    fn=append_predicted_letter,
    inputs=[
        predicted_letter_output,
        current_word_state
    ],
    outputs=[
        current_word_state,
        raw_word_output,
        corrected_word_output,
        application_status
    ]
)

remove_letter_button.click(
    fn=remove_last_letter,
    inputs=[
        current_word_state
    ],
    outputs=[
        current_word_state,
        raw_word_output,
        corrected_word_output,
        application_status
    ]
)

correct_word_button.click(
    fn=correct_current_word,
    inputs=[
        current_word_state
    ],
    outputs=[
        corrected_word_output,
        application_status
    ]
)

clear_word_button.click(
    fn=clear_current_word,
    inputs=[],
    outputs=[
        current_word_state,
        raw_word_output,
        corrected_word_output,
        application_status
    ]
)

multiple_predict_button.click(
    fn=predict_multiple_images,
    inputs=[
        multiple_images_input,
        model_selector
    ],
    outputs=[
        multiple_raw_word_output,
        multiple_corrected_word_output,
        multiple_gallery_output,
        multiple_status_output
    ]
)
```

```
print("=" * 60)
print("Gradio Web Application Built Successfully")
print("=" * 60)
```

```
=====
Gradio Web Application Built Successfully
=====
```

In []:

```
# =====
# Step 16: Launch the Gradio Website
# This step launches the multi-model ASL recognition website
# and creates a temporary public Gradio link from Google Colab.
# =====

demo.queue(
    default_concurrency_limit=1
)

demo.launch(
    share=True,
    debug=True,
    show_error=True
)
```

Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False in launch().
* Running on public URL: <https://ad503daab2525af179.gradio.live>

This share link is temporary and will last for up to 1 week (best effort). For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (<https://huggingface.co/spaces>)

Loading Swin Transformer...
Swin Transformer Loaded Successfully