In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
import time
import numpy as np
import matplotlib.pyplot as plt
from torchinfo import summary

# Set random seed for reproducibility
torch.manual_seed(42)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Define the Patch Embedding layer
class PatchEmbedding(nn.Module):
    def __init__(self, img_size=32, patch_size=8, in_channels=3, embed_dim=512):
        super().__init__()
        self.img_size = img_size
        self.patch_size = patch_size
        self.n_patches = (img_size // patch_size) ** 2

        # Linear projection
        self.proj = nn.Conv2d(
            in_channels, embed_dim,
            kernel_size=patch_size, stride=patch_size
        )

    def forward(self, x):
        # x: [B, C, H, W]
        x = self.proj(x)  # [B, embed_dim, H/patch_size, W/patch_size]
        x = x.flatten(2)  # [B, embed_dim, n_patches]
        x = x.transpose(1, 2)  # [B, n_patches, embed_dim]
        return x

# Define the Multi-Head Self-Attention module
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.head_dim = embed_dim // num_heads
        assert self.head_dim * num_heads == embed_dim, "embed_dim must be divisible

        self.qkv = nn.Linear(embed_dim, embed_dim * 3)
        self.proj = nn.Linear(embed_dim, embed_dim)

    def forward(self, x):
        # x: [B, n_patches + 1, embed_dim]
        B, N, C = x.shape
```

```
        qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, self.head_dim).permute(2
        q, k, v = qkv.unbind(0)  # Each has shape [B, num_heads, N, head_dim]

        # Scaled dot-product attention
        attn = (q @ k.transpose(-2, -1)) * (self.head_dim ** -0.5)  # [B, num_heads
        attn = attn.softmax(dim=-1)

        # Apply attention weights to values
        x = (attn @ v).transpose(1, 2).reshape(B, N, C)  # [B, N, C]
        x = self.proj(x)
        return x

# Define the MLP block
class MLP(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super().__init__()
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = nn.GELU()
        self.fc2 = nn.Linear(hidden_features, out_features)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.fc2(x)
        return x

# Define a Transformer Encoder Block
class TransformerEncoderBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, mlp_ratio=4.0):
        super().__init__()
        self.norm1 = nn.LayerNorm(embed_dim)
        self.attn = MultiHeadSelfAttention(embed_dim, num_heads)
        self.norm2 = nn.LayerNorm(embed_dim)
        self.mlp = MLP(
            in_features=embed_dim,
            hidden_features=int(embed_dim * mlp_ratio),
            out_features=embed_dim
        )

    def forward(self, x):
        # Pre-norm architecture
        x = x + self.attn(self.norm1(x))
        x = x + self.mlp(self.norm2(x))
        return x

# Define the Vision Transformer (ViT) model
class VisionTransformer(nn.Module):
    def __init__(
        self,
        img_size=32,
        patch_size=8,
        in_channels=3,
        num_classes=100,
        embed_dim=512,
        depth=6,
        num_heads=8,
```

```
        mlp_ratio=4.0
    ):
        super().__init__()
        self.patch_embed = PatchEmbedding(
            img_size=img_size,
            patch_size=patch_size,
            in_channels=in_channels,
            embed_dim=embed_dim
        )

        num_patches = self.patch_embed.n_patches

        # Class token and position embeddings
        self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
        self.pos_embed = nn.Parameter(torch.zeros(1, num_patches + 1, embed_dim))

        # Transformer encoder blocks
        self.blocks = nn.ModuleList([
            TransformerEncoderBlock(embed_dim, num_heads, mlp_ratio)
            for _ in range(depth)
        ])

        # Classification head
        self.norm = nn.LayerNorm(embed_dim)
        self.head = nn.Linear(embed_dim, num_classes)

        # Initialize parameters
        self._init_weights()

    def _init_weights(self):
        # Initialize patch embedding, class token, and position embedding
        nn.init.normal_(self.cls_token, std=0.02)
        nn.init.normal_(self.pos_embed, std=0.02)

        # Initialize transformer blocks
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)
                if m.bias is not None:
                    nn.init.zeros_(m.bias)
            elif isinstance(m, nn.LayerNorm):
                nn.init.ones_(m.weight)
                nn.init.zeros_(m.bias)

    def forward(self, x):
        # x: [B, C, H, W]
        batch_size = x.shape[0]

        # Patch embedding
        x = self.patch_embed(x)  # [B, n_patches, embed_dim]

        # Add class token
        cls_token = self.cls_token.expand(batch_size, -1, -1)  # [B, 1, embed_dim]
        x = torch.cat((cls_token, x), dim=1)  # [B, n_patches + 1, embed_dim]

        # Add position embeddings
```

```
        x = x + self.pos_embed  # [B, n_patches + 1, embed_dim]

        # Apply Transformer blocks
        for block in self.blocks:
            x = block(x)

        # Classification head
        x = self.norm(x)
        x = x[:, 0]  # Only use the class token for classification
        x = self.head(x)

        return x

# Helper function to count parameters and calculate FLOPs
def count_params_and_flops(model, input_size=(1, 3, 32, 32)):
    """Count parameters and estimate FLOPs using torchinfo."""
    model_summary = summary(model, input_size=input_size, verbose=0)
    params = model_summary.total_params
    flops = model_summary.total_mult_adds
    return params, flops

# Training function
def train(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    start_time = time.time()

    for batch_idx, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = 100.0 * correct / total
    epoch_time = time.time() - start_time

    return train_loss, train_acc, epoch_time

# Evaluation function
def evaluate(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
```

```
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(test_loader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    test_loss = running_loss / len(test_loader)
    test_acc = 100.0 * correct / total

    return test_loss, test_acc

# Function to train and evaluate a model configuration
def run_experiment(model, model_name, train_loader, test_loader, num_epochs=10):
    print(f"Running experiment with {model_name}")

    # Calculate parameters and FLOPs
    params, flops = count_params_and_flops(model)
    print(f"Number of parameters: {params:,}")
    print(f"Estimated FLOPs per forward pass: {flops:,}")

    # Set up training
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Training loop
    model.to(device)
    results = {
        "model_name": model_name,
        "params": params,
        "flops": flops,
        "train_time_per_epoch": [],
        "train_loss": [],
        "train_acc": [],
        "test_loss": [],
        "test_acc": []
    }

    for epoch in range(num_epochs):
        # Train
        train_loss, train_acc, epoch_time = train(model, train_loader, criterion, o

        # Evaluate
        test_loss, test_acc = evaluate(model, test_loader, criterion, device)

        # Record results
        results["train_time_per_epoch"].append(epoch_time)
        results["train_loss"].append(train_loss)
        results["train_acc"].append(train_acc)
        results["test_loss"].append(test_loss)
        results["test_acc"].append(test_acc)
```

```
        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train A
              f"Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%, Time: {epoch

    # Calculate average metrics
    results["avg_train_time"] = np.mean(results["train_time_per_epoch"])
    results["final_test_acc"] = results["test_acc"][-1]

    return results

# Main function to run the experiments
def main():
    # Data preprocessing
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761)),
    ])

    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761)),
    ])

    # Load CIFAR-100 dataset
    trainset = torchvision.datasets.CIFAR100(
        root='./data', train=True, download=True, transform=transform_train)
    trainloader = torch.utils.data.DataLoader(
        trainset, batch_size=64, shuffle=True, num_workers=2)

    testset = torchvision.datasets.CIFAR100(
        root='./data', train=False, download=True, transform=transform_test)
    testloader = torch.utils.data.DataLoader(
        testset, batch_size=64, shuffle=False, num_workers=2)

    # Define different ViT configurations
    vit_configs = [
        # Config 1: Small ViT (Patch=8x8, Embed=256, Layers=4, Heads=2, MLP=512)
        {
            "name": "ViT-Small (P8-E256-L4-H2)",
            "patch_size": 8,
            "embed_dim": 256,
            "depth": 4,
            "num_heads": 2,
            "mlp_ratio": 2.0
        },
        # Config 2: Medium ViT (Patch=8x8, Embed=512, Layers=4, Heads=4, MLP=2048)
        {
            "name": "ViT-Medium (P8-E512-L4-H4)",
            "patch_size": 8,
            "embed_dim": 512,
            "depth": 4,
            "num_heads": 4,
            "mlp_ratio": 4.0
        },
        # Config 3: Medium-Deep ViT (Patch=4x4, Embed=256, Layers=8, Heads=2, MLP=5
```

```
    {
        "name": "ViT-Medium-Deep (P4-E256-L8-H2)",
        "patch_size": 4,
        "embed_dim": 256,
        "depth": 8,
        "num_heads": 2,
        "mlp_ratio": 2.0
    },
    # Config 4: Large ViT (Patch=4x4, Embed=512, Layers=8, Heads=4, MLP=2048)
    {
        "name": "ViT-Large (P4-E512-L8-H4)",
        "patch_size": 4,
        "embed_dim": 512,
        "depth": 8,
        "num_heads": 4,
        "mlp_ratio": 4.0
    }
]

# Create models and run experiments
num_epochs = 30
all_results = []

# Run ViT experiments
for config in vit_configs:
    model = VisionTransformer(
        img_size=32,
        patch_size=config["patch_size"],
        in_channels=3,
        num_classes=100,
        embed_dim=config["embed_dim"],
        depth=config["depth"],
        num_heads=config["num_heads"],
        mlp_ratio=config["mlp_ratio"]
    )

    results = run_experiment(
        model=model,
        model_name=config["name"],
        train_loader=trainloader,
        test_loader=testloader,
        num_epochs=num_epochs
    )

    all_results.append(results)

# Run ResNet-18 experiment
resnet18 = torchvision.models.resnet18(weights=None)
# Modify the first layer for CIFAR-100 (smaller images)
resnet18.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=Fals
resnet18.maxpool = nn.Identity()  # Remove maxpool layer
# Modify the final FC layer for 100 classes
resnet18.fc = nn.Linear(512, 100)

resnet_results = run_experiment(
    model=resnet18,
```

```
        model_name="ResNet-18",
        train_loader=trainloader,
        test_loader=testloader,
        num_epochs=num_epochs
    )

    all_results.append(resnet_results)

    # Print results summary table
    print("\nResults Summary:")
    print("-" * 100)
    print(f"{'Model':<25} | {'Params':<12} | {'FLOPs':<15} | {'Avg Train Time':<15}
    print("-" * 100)

    for result in all_results:
        print(f"{result['model_name']:<25} | {result['params']:,} | {result['flops'

    # Visualize accuracy comparison
    plt.figure(figsize=(12, 5))

    # Plot test accuracy
    plt.subplot(1, 2, 1)
    for result in all_results:
        plt.plot(range(1, num_epochs+1), result['test_acc'], label=result['model_na
    plt.xlabel('Epoch')
    plt.ylabel('Test Accuracy (%)')
    plt.title('Test Accuracy Comparison')
    plt.legend()
    plt.grid(True)

    # Plot training time
    plt.subplot(1, 2, 2)
    model_names = [result['model_name'] for result in all_results]
    avg_times = [result['avg_train_time'] for result in all_results]
    plt.bar(model_names, avg_times)
    plt.xlabel('Model')
    plt.ylabel('Average Training Time per Epoch (s)')
    plt.title('Training Time Comparison')
    plt.xticks(rotation=45)
    plt.tight_layout()

    plt.savefig('results_comparison.png')
    plt.show()

if __name__ == "__main__":
    main()
```

```
Using device: cuda
Files already downloaded and verified
Files already downloaded and verified
Running experiment with ViT-Small (P8-E256-L4-H2)
Number of parameters: 2,188,644
Estimated FLOPs per forward pass: 2,925,156
Epoch 1/30, Train Loss: 4.2010, Train Acc: 6.15%, Test Loss: 4.1312, Test Acc: 6.5
5%, Time: 9.11s
Epoch 2/30, Train Loss: 3.9875, Train Acc: 8.16%, Test Loss: 4.0016, Test Acc: 8.3
7%, Time: 8.69s
Epoch 3/30, Train Loss: 3.9071, Train Acc: 9.33%, Test Loss: 3.8513, Test Acc: 10.2
8%, Time: 8.62s
Epoch 4/30, Train Loss: 3.8515, Train Acc: 10.24%, Test Loss: 3.8787, Test Acc: 9.9
7%, Time: 8.57s
Epoch 5/30, Train Loss: 3.9194, Train Acc: 9.33%, Test Loss: 3.9233, Test Acc: 9.1
0%, Time: 8.61s
Epoch 6/30, Train Loss: 3.8935, Train Acc: 9.49%, Test Loss: 3.9976, Test Acc: 8.1
4%, Time: 8.56s
Epoch 7/30, Train Loss: 3.8618, Train Acc: 10.00%, Test Loss: 3.8007, Test Acc: 11.3
7%, Time: 8.65s
Epoch 8/30, Train Loss: 3.8628, Train Acc: 10.14%, Test Loss: 3.9030, Test Acc: 9.6
9%, Time: 8.59s
Epoch 9/30, Train Loss: 4.0125, Train Acc: 8.07%, Test Loss: 4.0816, Test Acc: 7.6
9%, Time: 8.70s
Epoch 10/30, Train Loss: 4.0273, Train Acc: 7.99%, Test Loss: 3.9361, Test Acc: 9.0
0%, Time: 8.67s
Epoch 11/30, Train Loss: 3.9417, Train Acc: 9.16%, Test Loss: 3.8717, Test Acc: 10.2
5%, Time: 8.68s
Epoch 12/30, Train Loss: 3.8724, Train Acc: 9.97%, Test Loss: 3.8019, Test Acc: 10.8
0%, Time: 8.60s
Epoch 13/30, Train Loss: 3.7854, Train Acc: 11.11%, Test Loss: 3.7558, Test Acc: 1
1.58%, Time: 8.69s
Epoch 14/30, Train Loss: 3.7373, Train Acc: 11.83%, Test Loss: 3.7246, Test Acc: 1
2.62%, Time: 8.53s
Epoch 15/30, Train Loss: 3.7162, Train Acc: 12.33%, Test Loss: 3.7462, Test Acc: 1
2.08%, Time: 9.16s
Epoch 16/30, Train Loss: 3.7500, Train Acc: 11.79%, Test Loss: 3.7805, Test Acc: 1
1.31%, Time: 9.02s
Epoch 17/30, Train Loss: 3.7584, Train Acc: 11.82%, Test Loss: 3.8534, Test Acc: 9.9
5%, Time: 8.63s
Epoch 18/30, Train Loss: 3.8350, Train Acc: 10.55%, Test Loss: 3.8248, Test Acc: 1
0.62%, Time: 8.69s
Epoch 19/30, Train Loss: 3.8273, Train Acc: 10.58%, Test Loss: 3.8523, Test Acc: 9.9
4%, Time: 8.70s
Epoch 20/30, Train Loss: 3.8174, Train Acc: 10.88%, Test Loss: 3.8153, Test Acc: 1
0.39%, Time: 8.67s
Epoch 21/30, Train Loss: 3.7542, Train Acc: 11.67%, Test Loss: 3.8356, Test Acc: 1
0.96%, Time: 15.40s
Epoch 22/30, Train Loss: 3.7824, Train Acc: 11.54%, Test Loss: 3.8061, Test Acc: 1
1.07%, Time: 8.84s
Epoch 23/30, Train Loss: 3.7394, Train Acc: 12.10%, Test Loss: 3.7039, Test Acc: 1
2.29%, Time: 8.70s
Epoch 24/30, Train Loss: 3.7117, Train Acc: 12.45%, Test Loss: 3.7317, Test Acc: 1
2.15%, Time: 8.84s
Epoch 25/30, Train Loss: 3.7799, Train Acc: 11.61%, Test Loss: 3.7687, Test Acc: 1
1.72%, Time: 8.79s
```

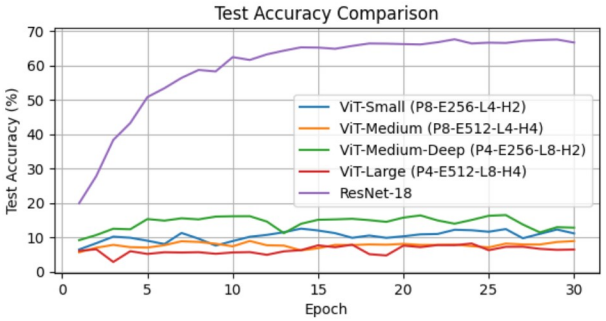
```
Epoch 26/30, Train Loss: 3.7662, Train Acc: 11.78%, Test Loss: 3.7388, Test Acc: 1
2.49%, Time: 8.75s
Epoch 27/30, Train Loss: 3.7663, Train Acc: 11.57%, Test Loss: 3.8808, Test Acc: 9.8
2%, Time: 8.72s
Epoch 28/30, Train Loss: 3.7160, Train Acc: 12.42%, Test Loss: 3.8197, Test Acc: 1
1.11%, Time: 8.66s
Epoch 29/30, Train Loss: 3.7159, Train Acc: 12.41%, Test Loss: 3.7398, Test Acc: 1
2.36%, Time: 8.76s
Epoch 30/30, Train Loss: 3.7390, Train Acc: 12.01%, Test Loss: 3.8046, Test Acc: 1
1.24%, Time: 8.57s
Running experiment with ViT-Medium (P8-E512-L4-H4)
Number of parameters: 12,769,892
Estimated FLOPs per forward pass: 14,242,916
Epoch 1/30, Train Loss: 4.3483, Train Acc: 4.72%, Test Loss: 4.2418, Test Acc: 5.7
5%, Time: 11.66s
Epoch 2/30, Train Loss: 4.1685, Train Acc: 6.19%, Test Loss: 4.1308, Test Acc: 7.0
6%, Time: 11.61s
Epoch 3/30, Train Loss: 4.0863, Train Acc: 7.27%, Test Loss: 4.0396, Test Acc: 7.9
0%, Time: 11.63s
Epoch 4/30, Train Loss: 4.0563, Train Acc: 7.56%, Test Loss: 4.0675, Test Acc: 7.2
3%, Time: 11.58s
Epoch 5/30, Train Loss: 4.1102, Train Acc: 7.07%, Test Loss: 4.1058, Test Acc: 7.1
1%, Time: 11.61s
Epoch 6/30, Train Loss: 4.0431, Train Acc: 7.94%, Test Loss: 4.0172, Test Acc: 7.8
1%, Time: 11.62s
Epoch 7/30, Train Loss: 4.0136, Train Acc: 8.21%, Test Loss: 3.9831, Test Acc: 8.9
5%, Time: 11.57s
Epoch 8/30, Train Loss: 4.0191, Train Acc: 8.18%, Test Loss: 3.9912, Test Acc: 8.7
5%, Time: 11.53s
Epoch 9/30, Train Loss: 4.0517, Train Acc: 7.80%, Test Loss: 4.0267, Test Acc: 8.2
3%, Time: 11.53s
Epoch 10/30, Train Loss: 4.0485, Train Acc: 7.65%, Test Loss: 4.0645, Test Acc: 7.4
8%, Time: 11.66s
Epoch 11/30, Train Loss: 4.0003, Train Acc: 8.26%, Test Loss: 3.9478, Test Acc: 9.0
3%, Time: 11.65s
Epoch 12/30, Train Loss: 4.0229, Train Acc: 7.90%, Test Loss: 4.0409, Test Acc: 7.7
8%, Time: 11.58s
Epoch 13/30, Train Loss: 4.0902, Train Acc: 7.10%, Test Loss: 4.0872, Test Acc: 7.6
8%, Time: 11.58s
Epoch 14/30, Train Loss: 4.1296, Train Acc: 6.49%, Test Loss: 4.1546, Test Acc: 6.3
0%, Time: 11.62s
Epoch 15/30, Train Loss: 4.1179, Train Acc: 6.71%, Test Loss: 4.1102, Test Acc: 6.9
3%, Time: 11.62s
Epoch 16/30, Train Loss: 4.0962, Train Acc: 6.92%, Test Loss: 4.0328, Test Acc: 7.9
0%, Time: 11.59s
Epoch 17/30, Train Loss: 4.0761, Train Acc: 7.26%, Test Loss: 4.0614, Test Acc: 7.8
5%, Time: 11.54s
Epoch 18/30, Train Loss: 4.0425, Train Acc: 7.78%, Test Loss: 4.0297, Test Acc: 8.0
4%, Time: 11.56s
Epoch 19/30, Train Loss: 4.0559, Train Acc: 7.70%, Test Loss: 4.0422, Test Acc: 7.9
5%, Time: 11.57s
Epoch 20/30, Train Loss: 4.0375, Train Acc: 7.73%, Test Loss: 4.0072, Test Acc: 8.1
9%, Time: 11.55s
Epoch 21/30, Train Loss: 4.0664, Train Acc: 7.61%, Test Loss: 4.0395, Test Acc: 7.8
9%, Time: 11.56s
Epoch 22/30, Train Loss: 4.0428, Train Acc: 7.87%, Test Loss: 4.0371, Test Acc: 7.8
```

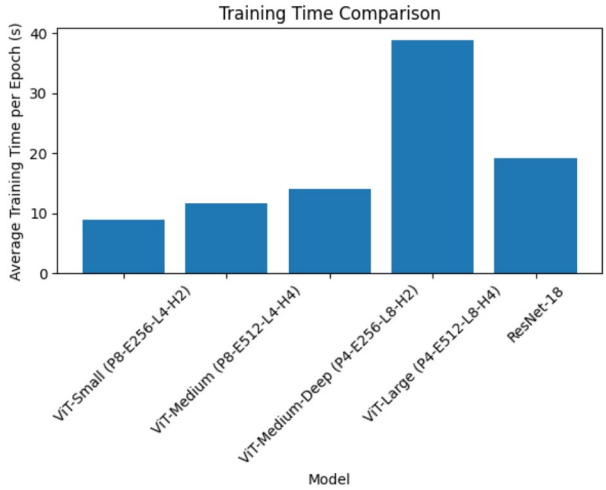
```
9%, Time: 11.53s
Epoch 23/30, Train Loss: 4.0380, Train Acc: 7.95%, Test Loss: 4.0632, Test Acc: 7.9
7%, Time: 11.54s
Epoch 24/30, Train Loss: 4.0630, Train Acc: 7.58%, Test Loss: 4.0842, Test Acc: 7.5
3%, Time: 11.57s
Epoch 25/30, Train Loss: 4.0448, Train Acc: 7.75%, Test Loss: 4.0969, Test Acc: 7.1
7%, Time: 11.55s
Epoch 26/30, Train Loss: 4.0231, Train Acc: 8.22%, Test Loss: 4.0179, Test Acc: 8.2
7%, Time: 11.58s
Epoch 27/30, Train Loss: 4.0316, Train Acc: 8.05%, Test Loss: 4.0289, Test Acc: 8.0
3%, Time: 11.51s
Epoch 28/30, Train Loss: 4.0239, Train Acc: 8.22%, Test Loss: 4.0348, Test Acc: 8.0
1%, Time: 11.54s
Epoch 29/30, Train Loss: 3.9919, Train Acc: 8.58%, Test Loss: 4.0009, Test Acc: 8.7
3%, Time: 11.50s
Epoch 30/30, Train Loss: 3.9809, Train Acc: 8.86%, Test Loss: 3.9753, Test Acc: 9.0
1%, Time: 11.54s
Running experiment with ViT-Medium-Deep (P4-E256-L8-H2)
Number of parameters: 4,272,484
Estimated FLOPs per forward pass: 5,045,860
Epoch 1/30, Train Loss: 4.0675, Train Acc: 7.38%, Test Loss: 3.9098, Test Acc: 9.2
6%, Time: 13.67s
Epoch 2/30, Train Loss: 3.8451, Train Acc: 10.39%, Test Loss: 3.7841, Test Acc: 10.7
5%, Time: 13.90s
Epoch 3/30, Train Loss: 3.8186, Train Acc: 10.46%, Test Loss: 3.7202, Test Acc: 12.5
9%, Time: 14.20s
Epoch 4/30, Train Loss: 3.7263, Train Acc: 12.02%, Test Loss: 3.7261, Test Acc: 12.4
4%, Time: 13.89s
Epoch 5/30, Train Loss: 3.6258, Train Acc: 13.88%, Test Loss: 3.5399, Test Acc: 15.4
1%, Time: 13.93s
Epoch 6/30, Train Loss: 3.6230, Train Acc: 13.89%, Test Loss: 3.6113, Test Acc: 14.9
5%, Time: 14.07s
Epoch 7/30, Train Loss: 3.5864, Train Acc: 14.39%, Test Loss: 3.5090, Test Acc: 15.6
3%, Time: 14.07s
Epoch 8/30, Train Loss: 3.6295, Train Acc: 13.87%, Test Loss: 3.5533, Test Acc: 15.3
2%, Time: 14.03s
Epoch 9/30, Train Loss: 3.5531, Train Acc: 15.39%, Test Loss: 3.4908, Test Acc: 16.1
2%, Time: 14.10s
Epoch 10/30, Train Loss: 3.4939, Train Acc: 16.17%, Test Loss: 3.5055, Test Acc: 1
6.22%, Time: 13.96s
Epoch 11/30, Train Loss: 3.5107, Train Acc: 16.02%, Test Loss: 3.5041, Test Acc: 1
6.24%, Time: 14.08s
Epoch 12/30, Train Loss: 3.5706, Train Acc: 14.95%, Test Loss: 3.5471, Test Acc: 1
4.68%, Time: 14.10s
Epoch 13/30, Train Loss: 3.6751, Train Acc: 13.35%, Test Loss: 3.8163, Test Acc: 1
1.30%, Time: 14.13s
Epoch 14/30, Train Loss: 3.6297, Train Acc: 14.16%, Test Loss: 3.6097, Test Acc: 1
4.03%, Time: 14.11s
Epoch 15/30, Train Loss: 3.5700, Train Acc: 15.04%, Test Loss: 3.5778, Test Acc: 1
5.20%, Time: 14.14s
Epoch 16/30, Train Loss: 3.5382, Train Acc: 15.64%, Test Loss: 3.5821, Test Acc: 1
5.32%, Time: 14.14s
Epoch 17/30, Train Loss: 3.5198, Train Acc: 15.61%, Test Loss: 3.5361, Test Acc: 1
5.49%, Time: 13.97s
Epoch 18/30, Train Loss: 3.5341, Train Acc: 15.53%, Test Loss: 3.5711, Test Acc: 1
5.09%, Time: 14.09s
```

```
Epoch 19/30, Train Loss: 3.5712, Train Acc: 14.69%, Test Loss: 3.5757, Test Acc: 1
4.60%, Time: 14.05s
Epoch 20/30, Train Loss: 3.5396, Train Acc: 15.59%, Test Loss: 3.5172, Test Acc: 1
5.83%, Time: 14.07s
Epoch 21/30, Train Loss: 3.4956, Train Acc: 16.12%, Test Loss: 3.4732, Test Acc: 1
6.47%, Time: 14.13s
Epoch 22/30, Train Loss: 3.4951, Train Acc: 16.09%, Test Loss: 3.5530, Test Acc: 1
5.01%, Time: 14.30s
Epoch 23/30, Train Loss: 3.6778, Train Acc: 13.30%, Test Loss: 3.6213, Test Acc: 1
4.06%, Time: 13.91s
Epoch 24/30, Train Loss: 3.6178, Train Acc: 14.41%, Test Loss: 3.5737, Test Acc: 1
5.15%, Time: 14.06s
Epoch 25/30, Train Loss: 3.5434, Train Acc: 15.44%, Test Loss: 3.5054, Test Acc: 1
6.37%, Time: 14.24s
Epoch 26/30, Train Loss: 3.5103, Train Acc: 15.92%, Test Loss: 3.4971, Test Acc: 1
6.57%, Time: 14.59s
Epoch 27/30, Train Loss: 3.5506, Train Acc: 15.29%, Test Loss: 3.6687, Test Acc: 1
3.82%, Time: 14.09s
Epoch 28/30, Train Loss: 3.6466, Train Acc: 13.69%, Test Loss: 3.8107, Test Acc: 1
1.52%, Time: 14.13s
Epoch 29/30, Train Loss: 3.7555, Train Acc: 12.25%, Test Loss: 3.7203, Test Acc: 1
3.04%, Time: 14.05s
Epoch 30/30, Train Loss: 3.7146, Train Acc: 12.49%, Test Loss: 3.7207, Test Acc: 1
2.90%, Time: 14.20s
Running experiment with ViT-Large (P4-E512-L8-H4)
Number of parameters: 25,330,276
Estimated FLOPs per forward pass: 26,877,028
Epoch 1/30, Train Loss: 4.3193, Train Acc: 4.90%, Test Loss: 4.1698, Test Acc: 6.1
3%, Time: 39.20s
Epoch 2/30, Train Loss: 4.2181, Train Acc: 5.25%, Test Loss: 4.1293, Test Acc: 6.6
4%, Time: 39.37s
Epoch 3/30, Train Loss: 4.2113, Train Acc: 5.38%, Test Loss: 4.5338, Test Acc: 2.9
3%, Time: 39.29s
Epoch 4/30, Train Loss: 4.2884, Train Acc: 4.46%, Test Loss: 4.1937, Test Acc: 6.0
6%, Time: 38.93s
Epoch 5/30, Train Loss: 4.2627, Train Acc: 5.00%, Test Loss: 4.2424, Test Acc: 5.2
6%, Time: 38.88s
Epoch 6/30, Train Loss: 4.2862, Train Acc: 4.81%, Test Loss: 4.2363, Test Acc: 5.7
5%, Time: 38.93s
Epoch 7/30, Train Loss: 4.2157, Train Acc: 5.63%, Test Loss: 4.1758, Test Acc: 5.6
7%, Time: 38.83s
Epoch 8/30, Train Loss: 4.2314, Train Acc: 5.38%, Test Loss: 4.2673, Test Acc: 5.7
6%, Time: 38.97s
Epoch 9/30, Train Loss: 4.2839, Train Acc: 4.82%, Test Loss: 4.2463, Test Acc: 5.3
1%, Time: 38.87s
Epoch 10/30, Train Loss: 4.2449, Train Acc: 5.26%, Test Loss: 4.2253, Test Acc: 5.6
9%, Time: 38.92s
Epoch 11/30, Train Loss: 4.1798, Train Acc: 5.97%, Test Loss: 4.2468, Test Acc: 5.7
9%, Time: 38.84s
Epoch 12/30, Train Loss: 4.1407, Train Acc: 6.44%, Test Loss: 4.2558, Test Acc: 5.0
1%, Time: 38.81s
Epoch 13/30, Train Loss: 4.1916, Train Acc: 6.12%, Test Loss: 4.1654, Test Acc: 6.0
2%, Time: 38.86s
Epoch 14/30, Train Loss: 4.1541, Train Acc: 6.29%, Test Loss: 4.1404, Test Acc: 6.3
6%, Time: 39.08s
Epoch 15/30, Train Loss: 4.0950, Train Acc: 7.19%, Test Loss: 4.0600, Test Acc: 7.7
```

```
6%, Time: 39.01s
Epoch 16/30, Train Loss: 4.1090, Train Acc: 6.87%, Test Loss: 4.0833, Test Acc: 7.2
2%, Time: 39.04s
Epoch 17/30, Train Loss: 4.0647, Train Acc: 7.33%, Test Loss: 4.0484, Test Acc: 7.9
5%, Time: 38.98s
Epoch 18/30, Train Loss: 4.1305, Train Acc: 6.41%, Test Loss: 4.1974, Test Acc: 5.2
0%, Time: 39.10s
Epoch 19/30, Train Loss: 4.1906, Train Acc: 5.81%, Test Loss: 4.2534, Test Acc: 4.8
2%, Time: 38.70s
Epoch 20/30, Train Loss: 4.1204, Train Acc: 6.55%, Test Loss: 4.0603, Test Acc: 7.6
9%, Time: 38.63s
Epoch 21/30, Train Loss: 4.1164, Train Acc: 6.55%, Test Loss: 4.0745, Test Acc: 7.2
5%, Time: 38.72s
Epoch 22/30, Train Loss: 4.0618, Train Acc: 7.55%, Test Loss: 4.0614, Test Acc: 7.8
4%, Time: 38.91s
Epoch 23/30, Train Loss: 4.0631, Train Acc: 7.41%, Test Loss: 4.0479, Test Acc: 7.8
1%, Time: 38.74s
Epoch 24/30, Train Loss: 4.0531, Train Acc: 7.55%, Test Loss: 4.0185, Test Acc: 8.2
9%, Time: 38.74s
Epoch 25/30, Train Loss: 4.0914, Train Acc: 6.99%, Test Loss: 4.1468, Test Acc: 6.3
6%, Time: 38.73s
Epoch 26/30, Train Loss: 4.1016, Train Acc: 6.73%, Test Loss: 4.0724, Test Acc: 7.3
3%, Time: 38.69s
Epoch 27/30, Train Loss: 4.0523, Train Acc: 7.42%, Test Loss: 4.0768, Test Acc: 7.3
6%, Time: 38.65s
Epoch 28/30, Train Loss: 4.0952, Train Acc: 6.94%, Test Loss: 4.1039, Test Acc: 6.7
4%, Time: 38.66s
Epoch 29/30, Train Loss: 4.1263, Train Acc: 6.42%, Test Loss: 4.1000, Test Acc: 6.4
6%, Time: 38.96s
Epoch 30/30, Train Loss: 4.1139, Train Acc: 6.49%, Test Loss: 4.1116, Test Acc: 6.5
5%, Time: 38.94s
Running experiment with ResNet-18
Number of parameters: 11,220,132
Estimated FLOPs per forward pass: 555,478,500
Epoch 1/30, Train Loss: 3.7300, Train Acc: 12.11%, Test Loss: 3.2240, Test Acc: 20.0
0%, Time: 19.41s
Epoch 2/30, Train Loss: 2.9262, Train Acc: 25.89%, Test Loss: 2.8736, Test Acc: 27.8
7%, Time: 19.30s
Epoch 3/30, Train Loss: 2.3983, Train Acc: 36.46%, Test Loss: 2.3370, Test Acc: 38.4
1%, Time: 19.32s
Epoch 4/30, Train Loss: 2.0346, Train Acc: 44.31%, Test Loss: 2.0945, Test Acc: 43.3
5%, Time: 19.35s
Epoch 5/30, Train Loss: 1.7816, Train Acc: 50.21%, Test Loss: 1.7774, Test Acc: 50.8
8%, Time: 19.36s
Epoch 6/30, Train Loss: 1.5829, Train Acc: 55.06%, Test Loss: 1.6659, Test Acc: 53.4
7%, Time: 19.17s
Epoch 7/30, Train Loss: 1.4267, Train Acc: 59.03%, Test Loss: 1.5929, Test Acc: 56.4
7%, Time: 19.17s
Epoch 8/30, Train Loss: 1.2924, Train Acc: 62.53%, Test Loss: 1.4947, Test Acc: 58.7
8%, Time: 19.17s
Epoch 9/30, Train Loss: 1.1806, Train Acc: 65.57%, Test Loss: 1.5261, Test Acc: 58.3
5%, Time: 19.17s
Epoch 10/30, Train Loss: 1.0778, Train Acc: 68.13%, Test Loss: 1.3423, Test Acc: 6
2.52%, Time: 19.16s
Epoch 11/30, Train Loss: 0.9810, Train Acc: 70.59%, Test Loss: 1.4229, Test Acc: 6
1.69%, Time: 19.16s
```

```
Epoch 12/30, Train Loss: 0.8920, Train Acc: 73.03%, Test Loss: 1.3621, Test Acc: 6
3.31%, Time: 19.21s
Epoch 13/30, Train Loss: 0.8187, Train Acc: 75.03%, Test Loss: 1.3343, Test Acc: 6
4.39%, Time: 19.20s
Epoch 14/30, Train Loss: 0.7438, Train Acc: 76.95%, Test Loss: 1.3552, Test Acc: 6
5.35%, Time: 19.29s
Epoch 15/30, Train Loss: 0.6831, Train Acc: 78.94%, Test Loss: 1.3432, Test Acc: 6
5.28%, Time: 19.20s
Epoch 16/30, Train Loss: 0.6086, Train Acc: 80.78%, Test Loss: 1.3960, Test Acc: 6
4.93%, Time: 19.14s
Epoch 17/30, Train Loss: 0.5492, Train Acc: 82.63%, Test Loss: 1.3959, Test Acc: 6
5.76%, Time: 19.17s
Epoch 18/30, Train Loss: 0.4991, Train Acc: 84.20%, Test Loss: 1.4119, Test Acc: 6
6.48%, Time: 19.16s
Epoch 19/30, Train Loss: 0.4517, Train Acc: 85.43%, Test Loss: 1.4175, Test Acc: 6
6.43%, Time: 19.17s
Epoch 20/30, Train Loss: 0.4093, Train Acc: 86.62%, Test Loss: 1.4990, Test Acc: 6
6.30%, Time: 19.25s
Epoch 21/30, Train Loss: 0.3783, Train Acc: 87.70%, Test Loss: 1.5312, Test Acc: 6
6.19%, Time: 19.31s
Epoch 22/30, Train Loss: 0.3413, Train Acc: 88.88%, Test Loss: 1.5437, Test Acc: 6
6.83%, Time: 19.36s
Epoch 23/30, Train Loss: 0.3124, Train Acc: 89.78%, Test Loss: 1.5257, Test Acc: 6
7.69%, Time: 19.19s
Epoch 24/30, Train Loss: 0.2852, Train Acc: 90.57%, Test Loss: 1.6117, Test Acc: 6
6.48%, Time: 19.17s
Epoch 25/30, Train Loss: 0.2784, Train Acc: 90.89%, Test Loss: 1.6653, Test Acc: 6
6.71%, Time: 19.17s
Epoch 26/30, Train Loss: 0.2454, Train Acc: 91.88%, Test Loss: 1.6628, Test Acc: 6
6.62%, Time: 19.16s
Epoch 27/30, Train Loss: 0.2303, Train Acc: 92.38%, Test Loss: 1.6182, Test Acc: 6
7.22%, Time: 19.18s
Epoch 28/30, Train Loss: 0.2168, Train Acc: 92.77%, Test Loss: 1.6622, Test Acc: 6
7.48%, Time: 19.19s
Epoch 29/30, Train Loss: 0.2055, Train Acc: 93.32%, Test Loss: 1.7165, Test Acc: 6
7.62%, Time: 19.17s
Epoch 30/30, Train Loss: 0.2020, Train Acc: 93.35%, Test Loss: 1.7839, Test Acc: 6
6.76%, Time: 19.21s

Results Summary:
--------------------------------------------------------------------------------
----------------
Model                     | Params       | FLOPs           | Avg Train Time  | Final
Test Acc
--------------------------------------------------------------------------------
----------------
ViT-Small (P8-E256-L4-H2) | 2,188,644 | 2,925,156 | 8.94s | 11.24%
ViT-Medium (P8-E512-L4-H4) | 12,769,892 | 14,242,916 | 11.58s | 9.01%
ViT-Medium-Deep (P4-E256-L8-H2) | 4,272,484 | 5,045,860 | 14.08s | 12.90%
ViT-Large (P4-E512-L8-H4) | 25,330,276 | 26,877,028 | 38.90s | 6.55%
ResNet-18                 | 11,220,132 | 555,478,500 | 19.22s | 66.76%
```





In [2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
from transformers import SwinForImageClassification, SwinConfig
import time
import numpy as np
import matplotlib.pyplot as plt
from torchinfo import summary
import copy

# Set random seed for reproducibility
torch.manual_seed(42)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

def train(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    start_time = time.time()

    for batch_idx, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs).logits
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

    train_loss = running_loss / len(train_loader)
```

```
    train_acc = 100.0 * correct / total
    epoch_time = time.time() - start_time

    return train_loss, train_acc, epoch_time

def evaluate(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(test_loader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs).logits
            loss = criterion(outputs, targets)

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    test_loss = running_loss / len(test_loader)
    test_acc = 100.0 * correct / total

    return test_loss, test_acc

def count_params(model):
    """Count trainable parameters."""
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

def run_experiment(model, model_name, train_loader, test_loader, num_epochs=5, fine
    print(f"Running experiment with {model_name}")

    # Calculate parameters
    params = count_params(model)
    print(f"Number of trainable parameters: {params:,}")

    # Set up training
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=2e-5)

    # Training loop
    model.to(device)
    results = {
        "model_name": model_name,
        "params": params,
        "train_time_per_epoch": [],
        "train_loss": [],
        "train_acc": [],
        "test_loss": [],
        "test_acc": []
    }

    for epoch in range(num_epochs):
        # Train
```

```
        train_loss, train_acc, epoch_time = train(model, train_loader, criterion, o

        # Evaluate
        test_loss, test_acc = evaluate(model, test_loader, criterion, device)

        # Record results
        results["train_time_per_epoch"].append(epoch_time)
        results["train_loss"].append(train_loss)
        results["train_acc"].append(train_acc)
        results["test_loss"].append(test_loss)
        results["test_acc"].append(test_acc)

        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train A
              f"Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%, Time: {epoch

    # Calculate average metrics
    results["avg_train_time"] = np.mean(results["train_time_per_epoch"])
    results["final_test_acc"] = results["test_acc"][-1]

    return results

def freeze_backbone(model):
    """Freeze the backbone layers, keeping only the classifier trainable."""
    for name, param in model.named_parameters():
        if 'classifier' not in name:  # Freeze everything except the classifier
            param.requires_grad = False
    return model

# Define a custom resize transform class instead of using Lambda
class ResizeCIFAR(object):
    def __init__(self, target_size=224):
        self.target_size = target_size

    def __call__(self, img):
        # Add batch dimension, resize, then remove batch dimension
        img = img.unsqueeze(0)
        img = torch.nn.functional.interpolate(
            img, size=(self.target_size, self.target_size),
            mode='bilinear', align_corners=False
        )
        return img.squeeze(0)

def main():
    # Data preprocessing with resize for Swin Transformer (needs 224x224)
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761)),
        ResizeCIFAR(target_size=224)
    ])

    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761)),
        ResizeCIFAR(target_size=224)
```

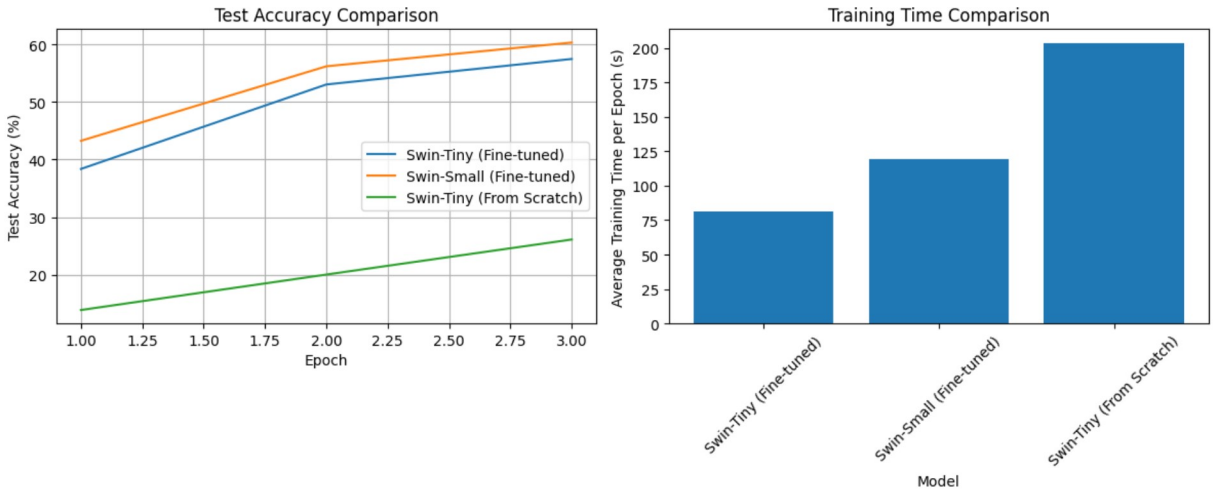
```
])

# Load CIFAR-100 dataset
trainset = torchvision.datasets.CIFAR100(
    root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=32, shuffle=True, num_workers=0)  # Set num_workers to

testset = torchvision.datasets.CIFAR100(
    root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=32, shuffle=False, num_workers=0)  # Set num_workers to

# Create a dictionary to store all results
all_results = []
num_epochs = 3  # Set to 3 epochs for demonstration (within 2-5 range)

# 1. Fine-tune Swin-Tiny model
print("Loading Swin-Tiny pretrained model...")
swin_tiny = SwinForImageClassification.from_pretrained("microsoft/swin-tiny-pat
                                                       num_labels=100,
                                                       ignore_mismatched_sizes=T
# Freeze backbone
swin_tiny = freeze_backbone(swin_tiny)

swin_tiny_results = run_experiment(
    model=swin_tiny,
    model_name="Swin-Tiny (Fine-tuned)",
    train_loader=trainloader,
    test_loader=testloader,
    num_epochs=num_epochs
)
all_results.append(swin_tiny_results)

# 2. Fine-tune Swin-Small model
print("Loading Swin-Small pretrained model...")
swin_small = SwinForImageClassification.from_pretrained("microsoft/swin-small-p
                                                        num_labels=100,
                                                        ignore_mismatched_sizes=
# Freeze backbone
swin_small = freeze_backbone(swin_small)

swin_small_results = run_experiment(
    model=swin_small,
    model_name="Swin-Small (Fine-tuned)",
    train_loader=trainloader,
    test_loader=testloader,
    num_epochs=num_epochs
)
all_results.append(swin_small_results)

# 3. Train Swin-Tiny from scratch
print("Creating Swin-Tiny model from scratch...")
# Create Swin-Tiny configuration
config = SwinConfig(
    num_labels=100,
```

```
    image_size=224,
    patch_size=4,
    num_channels=3,
    embed_dim=96,
    depths=[2, 2, 6, 2],
    num_heads=[3, 6, 12, 24],
    window_size=7
)

swin_scratch = SwinForImageClassification(config)

swin_scratch_results = run_experiment(
    model=swin_scratch,
    model_name="Swin-Tiny (From Scratch)",
    train_loader=trainloader,
    test_loader=testloader,
    num_epochs=num_epochs
)
all_results.append(swin_scratch_results)

# Print results summary table
print("\nResults Summary:")
print("-" * 85)
print(f"{'Model':<25} | {'Trainable Params':<18} | {'Avg Train Time':<15} | {'F
print("-" * 85)

for result in all_results:
    print(f"{result['model_name']:<25} | {result['params']:,} | {result['avg_tr

# Visualize accuracy comparison
plt.figure(figsize=(12, 5))

# Plot test accuracy
plt.subplot(1, 2, 1)
for result in all_results:
    plt.plot(range(1, num_epochs+1), result['test_acc'], label=result['model_na
plt.xlabel('Epoch')
plt.ylabel('Test Accuracy (%)')
plt.title('Test Accuracy Comparison')
plt.legend()
plt.grid(True)

# Plot training time
plt.subplot(1, 2, 2)
model_names = [result['model_name'] for result in all_results]
avg_times = [result['avg_train_time'] for result in all_results]
plt.bar(model_names, avg_times)
plt.xlabel('Model')
plt.ylabel('Average Training Time per Epoch (s)')
plt.title('Training Time Comparison')
plt.xticks(rotation=45)
plt.tight_layout()

plt.savefig('swin_results_comparison.png')
plt.show()
```

```
if __name__ == "__main__":
    main()
```

```
C:\Users\andre\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Using device: cuda
Files already downloaded and verified
Files already downloaded and verified
Loading Swin-Tiny pretrained model...
Some weights of SwinForImageClassification were not initialized from the model checkpoint at microsoft/swin-tiny-patch4-window7-224 and are newly initialized because the shapes did not match:
- classifier.bias: found shape torch.Size([1000]) in the checkpoint and torch.Size([100]) in the model instantiated
- classifier.weight: found shape torch.Size([1000, 768]) in the checkpoint and torch.Size([100, 768]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Running experiment with Swin-Tiny (Fine-tuned)
Number of trainable parameters: 76,900
Epoch 1/3, Train Loss: 4.2101, Train Acc: 19.84%, Test Loss: 3.8165, Test Acc: 38.38%, Time: 81.35s
Epoch 2/3, Train Loss: 3.4352, Train Acc: 48.25%, Test Loss: 3.1475, Test Acc: 53.03%, Time: 81.16s
Epoch 3/3, Train Loss: 2.8460, Train Acc: 55.70%, Test Loss: 2.6471, Test Acc: 57.45%, Time: 80.91s
Loading Swin-Small pretrained model...
C:\Users\andre\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\huggingface_hub\file_download.py:144: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\andre\.cache\huggingface\hub\models--microsoft--swin-small-patch4-window7-224. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
  warnings.warn(message)
Some weights of SwinForImageClassification were not initialized from the model checkpoint at microsoft/swin-small-patch4-window7-224 and are newly initialized because the shapes did not match:
- classifier.weight: found shape torch.Size([1000, 768]) in the checkpoint and torch.Size([100, 768]) in the model instantiated
- classifier.bias: found shape torch.Size([1000]) in the checkpoint and torch.Size([100]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
Running experiment with Swin-Small (Fine-tuned)
Number of trainable parameters: 76,900
Epoch 1/3, Train Loss: 4.1743, Train Acc: 23.10%, Test Loss: 3.7459, Test Acc: 43.2
6%, Time: 119.74s
Epoch 2/3, Train Loss: 3.3430, Train Acc: 52.50%, Test Loss: 3.0313, Test Acc: 56.1
8%, Time: 119.25s
Epoch 3/3, Train Loss: 2.7109, Train Acc: 59.08%, Test Loss: 2.5002, Test Acc: 60.3
0%, Time: 119.20s
Creating Swin-Tiny model from scratch...
Running experiment with Swin-Tiny (From Scratch)
Number of trainable parameters: 27,596,254
Epoch 1/3, Train Loss: 4.0350, Train Acc: 8.38%, Test Loss: 3.6695, Test Acc: 13.9
2%, Time: 203.16s
Epoch 2/3, Train Loss: 3.4967, Train Acc: 16.46%, Test Loss: 3.2916, Test Acc: 20.0
7%, Time: 202.86s
Epoch 3/3, Train Loss: 3.1466, Train Acc: 22.77%, Test Loss: 2.9846, Test Acc: 26.1
5%, Time: 204.45s

Results Summary:
--------------------------------------------------------------------------------
-
Model                     | Trainable Params   | Avg Train Time  | Final Test Acc
--------------------------------------------------------------------------------
-
Swin-Tiny (Fine-tuned)    | 76,900 | 81.14s | 57.45%
Swin-Small (Fine-tuned)   | 76,900 | 119.40s | 60.30%
Swin-Tiny (From Scratch)  | 27,596,254 | 203.49s | 26.15%
```



In [ ]: