# 🧠 AG-News Topic Classifier using BiLSTM and DistilBERT

Complete pipeline for text classification on AG News dataset using Deep Learning.

## 📋 Table of Contents



## 1. Setup & Installation {#setup}

```
# Install required packages for Google Colab
!pip install transformers datasets accelerate -q
!pip install wordcloud -q


# Upload your data files to Colab
from google.colab import files
import os

# Create data directory
os.makedirs('data', exist_ok=True)
os.makedirs('models', exist_ok=True)

print("📁 Directories created. Please upload train.csv and test.csv to the data/ folder.")
print("💡 Use the file browser on the left to upload your CSV files to data/ directory.")
print("🔧 Or use the code below to upload files directly:")
print("# uploaded = files.upload()  # Uncomment to upload files")
```

```
📁 Directories created. Please upload train.csv and test.csv to the data/ folder.
💡 Use the file browser on the left to upload your CSV files to data/ directory.
🔧 Or use the code below to upload files directly:
# uploaded = files.upload()  # Uncomment to upload files
```

```
Start coding or generate with AI.
```

```
ERROR: Could not find a version that satisfies the requirement AdamW (from versions: none)
ERROR: No matching distribution found for AdamW
```

```
import pandas as pd
import numpy as np
import torch
```

```
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.auto import tqdm  # Changed for Colab compatibility
import re
import warnings
warnings.filterwarnings('ignore')

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Device configuration - Colab GPU detection
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Check if GPU is available and show details
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
    print(f"Memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:.1f} GB")
else:
    print("⚠️  No GPU available, using CPU (training will be slower)")
```

```
Using device: cuda
GPU: Tesla T4
Memory: 15.8 GB
```

## 2. Data Loading & Preprocessing {#preprocessing}

```
class AGNewsDataset(Dataset):
    def __init__(self, texts, labels, tokenizer=None, max_length=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        if self.tokenizer:
            encoding = self.tokenizer(
                text,
                truncation=True,
                padding='max_length',
                max_length=self.max_length,
                return_tensors='pt'
```

```
            )

            return {
                'text': text,
                'input_ids': encoding['input_ids'].flatten(),
                'attention_mask': encoding['attention_mask'].flatten(),
                'label': torch.tensor(label, dtype=torch.long)
            }
        else:
            return {
                'text': text,
                'label': torch.tensor(label, dtype=torch.long)
            }
```

```
class DataProcessor:
    def __init__(self):
        self.label_mapping = {
            1: 0,  # World -> 0
            2: 1,  # Sports -> 1
            3: 2,  # Business -> 2
            4: 3   # Science/Technology -> 3
        }
        self.class_names = ['World', 'Sports', 'Business', 'Science/Technology']

    def clean_text(self, text):
        """Clean and preprocess text"""
        text = str(text).lower()
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
        text = re.sub(r'[^\w\s]', '', text)
        text = re.sub(r'\s+', ' ', text).strip()
        return text

    def load_data(self, train_path, test_path, sample_size=None):
        """Load and preprocess AG News data"""
        # Load data
        train_df = pd.read_csv(train_path)
        test_df = pd.read_csv(test_path)

        # Set column names
        train_df.columns = ['class', 'title', 'description']
        test_df.columns = ['class', 'title', 'description']

        # Sample data if specified (for faster training)
        if sample_size:
            train_df = train_df.sample(n=sample_size, random_state=42)
            test_df = test_df.sample(n=min(sample_size//5, len(test_df)), random_state=42)

        # Combine title and description
        train_df['text'] = train_df['title'] + ' ' + train_df['description']
        test_df['text'] = test_df['title'] + ' ' + test_df['description']

        # Clean text
        train_df['text'] = train_df['text'].apply(self.clean_text)
        test_df['text'] = test_df['text'].apply(self.clean_text)

        # Map labels
        train_df['label'] = train_df['class'].map(self.label_mapping)
```

```
        test_df['label'] = test_df['class'].map(self.label_mapping)

        return train_df, test_df

# Initialize data processor
processor = DataProcessor()
```

```
# Load data with sampling for faster training (remove sample_size for full dataset)
SAMPLE_SIZE = None # Reduced for Colab compatibility, set to None for full dataset

print("📊 Loading AG News dataset...")
try:
    train_df, test_df = processor.load_data('data/train.csv', 'data/test.csv', sample_size=
    print(f"✅ Data loaded successfully!")
    print(f"Training samples: {len(train_df)}")
    print(f"Test samples: {len(test_df)}")
    print(f"Classes: {processor.class_names}")
except FileNotFoundError:
    print("❌ Data files not found!")
    print("📁 Please upload train.csv and test.csv to the data/ directory")
    print("💡 You can:")
    print("   1. Use the file browser on the left")
    print("   2. Or uncomment the file upload code in the first cell")
    raise
```

```
📊 Loading AG News dataset...
✅ Data loaded successfully!
Training samples: 120000
Test samples: 7600
Classes: ['World', 'Sports', 'Business', 'Science/Technology']
```

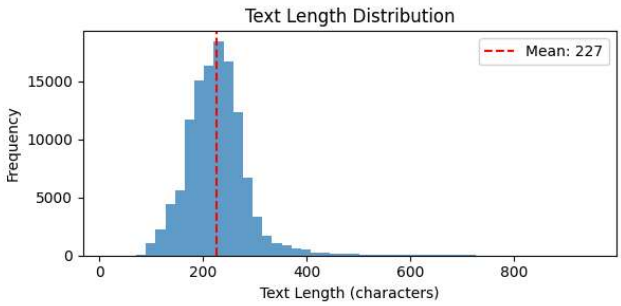
## 3. Exploratory Data Analysis {#eda}

```
# Display sample data
print("📝 Sample data:")
display(train_df.head())

# Class distribution
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
class_counts = train_df['label'].value_counts().sort_index()
plt.bar(range(len(processor.class_names)), class_counts.values)
plt.title('Training Data - Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(range(len(processor.class_names)), processor.class_names, rotation=45)

plt.subplot(1, 2, 2)
text_lengths = train_df['text'].str.len()
plt.hist(text_lengths, bins=50, alpha=0.7)
plt.title('Text Length Distribution')
plt.xlabel('Text Length (characters)')
plt.ylabel('Frequency')
plt.axvline(text_lengths.mean(), color='red', linestyle='--', label=f'Mean: {text_lengths.m
```

```
plt.legend()

plt.tight_layout()
plt.show()

print(f"Average text length: {text_lengths.mean():.0f} characters")
print(f"Max text length: {text_lengths.max():.0f} characters")
```

Sample data:

| | class | title | description | text | label |
|---|---|---|---|---|---|
| **0** | 3 | Wall St. Bears Claw Back Into the Black (Reuters) | Reuters - Short-sellers, Wall Street's dwindli... | wall st bears claw back into the black reuters... | 2 |
| **1** | 3 | Carlyle Looks Toward Commercial Aerospace (Reu... | Reuters - Private investment firm Carlyle Grou... | carlyle looks toward commercial aerospace reut... | 2 |
| **2** | 3 | Oil and Economy Cloud Stocks' Outlook (Reuters) | Reuters - Soaring crude prices plus worries\ab... | oil and economy cloud stocks outlook reuters r... | 2 |
| **3** | 3 | Iraq Halts Oil Exports from Main Southern Pipe... | Reuters - Authorities have halted oil export\f... | iraq halts oil exports from main southern pipe... | 2 |
| **4** | 3 | Oil prices soar to all-time record, posing new... | AFP - Tearaway world oil prices, toppling reco... | oil prices soar to alltime record posing new m... | 2 |


```
Average text length: 227 characters
Max text length: 951 characters
```

```
# Create train/validation split
train_texts = train_df['text'].tolist()
train_labels = train_df['label'].tolist()
test_texts = test_df['text'].tolist()
test_labels = test_df['label'].tolist()

# Split training data into train/validation
train_texts, val_texts, train_labels, val_labels = train_test_split(
    train_texts, train_labels, test_size=0.1, random_state=42, stratify=train_labels
)

print(f"Final splits:")
print(f"Train: {len(train_texts)} samples")
print(f"Validation: {len(val_texts)} samples")
print(f"Test: {len(test_texts)} samples")
```

```
Final splits:
Train: 108000 samples
Validation: 12000 samples
Test: 7600 samples
```

## 4. Model 1: DistilBERT Fine-tuning {#distilbert}

```
# Initialize DistilBERT tokenizer and model
print("🤖 Loading DistilBERT model...")
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=4
).to(device)

print(f"✅ Model loaded and moved to {device}")
```

```
🤖 Loading DistilBERT model...
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 5.17kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 9.07MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 5.72MB/s]
config.json: 100%  483/483 [00:00<00:00, 38.3kB/s]
model.safetensors: 100%  268M/268M [00:01<00:00, 205MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model chec
You should probably TRAIN this model on a down-stream task to be able to use it for predicti
✅ Model loaded and moved to cuda
```

```
# Create datasets and dataloaders
# Adjust batch size based on available memory
if torch.cuda.is_available():
    BATCH_SIZE = 16  # GPU can handle larger batches
else:
    BATCH_SIZE = 8   # Smaller for CPU

MAX_LENGTH = 128

train_dataset = AGNewsDataset(train_texts, train_labels, tokenizer, MAX_LENGTH)
val_dataset = AGNewsDataset(val_texts, val_labels, tokenizer, MAX_LENGTH)
test_dataset = AGNewsDataset(test_texts, test_labels, tokenizer, MAX_LENGTH)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"📦 Created dataloaders with batch size: {BATCH_SIZE}")
print(f"Total training batches: {len(train_loader)}")
print(f"⏱️  Estimated training time per epoch: {len(train_loader) * BATCH_SIZE * 0.1 / 60:
```

```
📦 Created dataloaders with batch size: 16
Total training batches: 6750
⏱️  Estimated training time per epoch: 180.0 minutes
```

```
# Training function for DistilBERT
def train_distilbert(model, train_loader, val_loader, epochs=3):
    optimizer = optim.AdamW(model.parameters(), lr=2e-5)
    criterion = nn.CrossEntropyLoss()

    train_losses = []
    val_accuracies = []

    for epoch in range(epochs):
        print(f"\n🔄 Epoch {epoch+1}/{epochs}")

        # Training
        model.train()
        total_loss = 0

        progress_bar = tqdm(train_loader, desc="Training")
        for batch in progress_bar:
            optimizer.zero_grad()

            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labe
            loss = outputs.loss

            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            progress_bar.set_postfix({'loss': f'{loss.item():.4f}'})

        avg_loss = total_loss / len(train_loader)
        train_losses.append(avg_loss)

        # Validation
        model.eval()
        correct = 0
        total = 0

        with torch.no_grad():
            for batch in tqdm(val_loader, desc="Validation"):
                input_ids = batch['input_ids'].to(device)
                attention_mask = batch['attention_mask'].to(device)
                labels = batch['label'].to(device)

                outputs = model(input_ids=input_ids, attention_mask=attention_mask)
                predictions = torch.argmax(outputs.logits, dim=-1)

                correct += (predictions == labels).sum().item()
                total += labels.size(0)

        val_accuracy = correct / total
        val_accuracies.append(val_accuracy)

        print(f"📊 Epoch {epoch+1} - Loss: {avg_loss:.4f}, Val Accuracy: {val_accuracy:.4f
```

```
    return train_losses, val_accuracies
```

```
# Train DistilBERT model
EPOCHS = 5  # Start with 3 epochs for quick training

print(f"🚀 Starting DistilBERT training for {EPOCHS} epochs...")
train_losses, val_accuracies = train_distilbert(model, train_loader, val_loader, epochs=EPO

# Save the model
torch.save(model.state_dict(), 'models/distilbert_model.pth')
print("💾 Model saved to models/distilbert_model.pth")
```

```
🚀 Starting DistilBERT training for 5 epochs...

🔄 Epoch 1/5
Training: 100%                                    6750/6750 [20:40<00:00,  5.50it/s, loss=0.0591]
Validation: 100%                                   750/750 [00:47<00:00, 16.45it/s]
📊 Epoch 1 - Loss: 0.2378, Val Accuracy: 0.9374

🔄 Epoch 2/5
Training: 100%                                    6750/6750 [20:42<00:00,  5.48it/s, loss=0.0412]
Validation: 100%                                   750/750 [00:47<00:00, 16.11it/s]
📊 Epoch 2 - Loss: 0.1449, Val Accuracy: 0.9424

🔄 Epoch 3/5
Training: 100%                                    6750/6750 [20:43<00:00,  5.45it/s, loss=0.1901]
Validation: 100%                                   750/750 [00:47<00:00, 16.08it/s]
📊 Epoch 3 - Loss: 0.0968, Val Accuracy: 0.9434

🔄 Epoch 4/5
Training: 100%                                    6750/6750 [20:42<00:00,  5.37it/s, loss=0.0107]
Validation: 100%                                   750/750 [00:47<00:00, 14.86it/s]
📊 Epoch 4 - Loss: 0.0637, Val Accuracy: 0.9414

🔄 Epoch 5/5
Training: 100%                                    6750/6750 [20:42<00:00,  5.50it/s, loss=0.0007]
Validation: 100%                                   750/750 [00:47<00:00, 16.03it/s]
📊 Epoch 5 - Loss: 0.0432, Val Accuracy: 0.9410
💾 Model saved to models/distilbert_model.pth
```

## 5. Model 2: BiLSTM with Attention {#bilstm}

```
# BiLSTM Model with Attention
class BiLSTMAttention(nn.Module):
    def __init__(self, vocab_size, embedding_dim=100, hidden_dim=256, num_layers=2, num_cla
        super(BiLSTMAttention, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
```

```
            self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers,
                                batch_first=True, bidirectional=True, dropout=dropout)

            # Attention mechanism
            self.attention = nn.Linear(hidden_dim * 2, 1)

            # Classification head
            self.classifier = nn.Sequential(
                nn.Dropout(dropout),
                nn.Linear(hidden_dim * 2, hidden_dim),
                nn.ReLU(),
                nn.Dropout(dropout),
                nn.Linear(hidden_dim, num_classes)
            )

        def forward(self, x, lengths=None):
            # Embedding
            embedded = self.embedding(x)  # (batch_size, seq_len, embedding_dim)

            # LSTM
            lstm_out, (hidden, cell) = self.lstm(embedded)  # (batch_size, seq_len, hidden_dim*

            # Attention mechanism
            attention_weights = torch.softmax(self.attention(lstm_out), dim=1)  # (batch_size,
            context_vector = torch.sum(attention_weights * lstm_out, dim=1)  # (batch_size, hid

            # Classification
            output = self.classifier(context_vector)
            return output, attention_weights
```

```
    # Simple tokenizer for BiLSTM (word-level)
    from collections import Counter

    class SimpleTokenizer:
        def __init__(self, max_vocab_size=10000):
            self.max_vocab_size = max_vocab_size
            self.word2idx = {'<PAD>': 0, '<UNK>': 1}
            self.idx2word = {0: '<PAD>', 1: '<UNK>'}

        def build_vocab(self, texts):
            # Count word frequencies
            word_counts = Counter()
            for text in texts:
                words = text.split()
                word_counts.update(words)

            # Add most frequent words to vocabulary
            most_common = word_counts.most_common(self.max_vocab_size - 2)

            for i, (word, count) in enumerate(most_common):
                idx = i + 2  # Start from 2 (after PAD and UNK)
                self.word2idx[word] = idx
                self.idx2word[idx] = word

            print(f"Built vocabulary with {len(self.word2idx)} words")

        def encode(self, texts, max_length=128):
```

```
        encoded = []
        for text in texts:
            words = text.split()[:max_length]
            indices = [self.word2idx.get(word, 1) for word in words]  # 1 is UNK

            # Pad to max_length
            if len(indices) < max_length:
                indices.extend([0] * (max_length - len(indices)))  # 0 is PAD

            encoded.append(indices)

        return torch.tensor(encoded, dtype=torch.long)

# Build vocabulary and encode texts for BiLSTM
bilstm_tokenizer = SimpleTokenizer(max_vocab_size=10000)
bilstm_tokenizer.build_vocab(train_texts)

# Encode texts
train_encoded = bilstm_tokenizer.encode(train_texts, max_length=128)
val_encoded = bilstm_tokenizer.encode(val_texts, max_length=128)
test_encoded = bilstm_tokenizer.encode(test_texts, max_length=128)

print(f"Encoded shapes - Train: {train_encoded.shape}, Val: {val_encoded.shape}, Test: {tes
```

```
Built vocabulary with 10000 words
Encoded shapes - Train: torch.Size([108000, 128]), Val: torch.Size([12000, 128]), Test: torc
```

```
# Create BiLSTM datasets and dataloaders
class BiLSTMDataset(Dataset):
    def __init__(self, encoded_texts, labels):
        self.encoded_texts = encoded_texts
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.encoded_texts)

    def __getitem__(self, idx):
        return {
            'input_ids': self.encoded_texts[idx],
            'label': self.labels[idx]
        }

bilstm_train_dataset = BiLSTMDataset(train_encoded, train_labels)
bilstm_val_dataset = BiLSTMDataset(val_encoded, val_labels)
bilstm_test_dataset = BiLSTMDataset(test_encoded, test_labels)

bilstm_train_loader = DataLoader(bilstm_train_dataset, batch_size=BATCH_SIZE, shuffle=True)
bilstm_val_loader = DataLoader(bilstm_val_dataset, batch_size=BATCH_SIZE, shuffle=False)
bilstm_test_loader = DataLoader(bilstm_test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

```
# Initialize BiLSTM model
vocab_size = len(bilstm_tokenizer.word2idx)
bilstm_model = BiLSTMAttention(
    vocab_size=vocab_size,
    embedding_dim=100,
    hidden_dim=128,
```

```
    num_layers=2,
    num_classes=4,
    dropout=0.3
).to(device)

print(f"BiLSTM model initialized with vocab size: {vocab_size}")
print(f"Model parameters: {sum(p.numel() for p in bilstm_model.parameters())}")
```

```
BiLSTM model initialized with vocab size: 10000
Model parameters: 1664453
```

```
# Training function for BiLSTM
def train_bilstm(model, train_loader, val_loader, epochs=5):
    optimizer = optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.CrossEntropyLoss()

    train_losses = []
    val_accuracies = []

    for epoch in range(epochs):
        print(f"\n🔄 Epoch {epoch+1}/{epochs}")

        # Training
        model.train()
        total_loss = 0

        progress_bar = tqdm(train_loader, desc="Training")
        for batch in progress_bar:
            optimizer.zero_grad()

            input_ids = batch['input_ids'].to(device)
            labels = batch['label'].to(device)

            outputs, attention_weights = model(input_ids)
            loss = criterion(outputs, labels)

            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            progress_bar.set_postfix({'loss': f'{loss.item():.4f}'})

        avg_loss = total_loss / len(train_loader)
        train_losses.append(avg_loss)

        # Validation
        model.eval()
        correct = 0
        total = 0

        with torch.no_grad():
            for batch in tqdm(val_loader, desc="Validation"):
                input_ids = batch['input_ids'].to(device)
                labels = batch['label'].to(device)

                outputs, _ = model(input_ids)
                predictions = torch.argmax(outputs, dim=-1)
```

```
                correct += (predictions == labels).sum().item()
                total += labels.size(0)

        val_accuracy = correct / total
        val_accuracies.append(val_accuracy)

        print(f"📊 Epoch {epoch+1} - Loss: {avg_loss:.4f}, Val Accuracy: {val_accuracy:.4f

    return train_losses, val_accuracies
```

```
# Train BiLSTM model
BILSTM_EPOCHS = 5

print(f"🚀 Starting BiLSTM training for {BILSTM_EPOCHS} epochs...")
bilstm_train_losses, bilstm_val_accuracies = train_bilstm(bilstm_model, bilstm_train_loader

# Save the model
torch.save(bilstm_model.state_dict(), 'models/bilstm_model.pth')
print("💾 BiLSTM model saved to models/bilstm_model.pth")
```

```
🚀 Starting BiLSTM training for 5 epochs...

🔄 Epoch 1/5
Training: 100%                                    6750/6750 [00:51<00:00, 136.65it/s, loss=0.2929]

Validation: 100%                                   750/750 [00:01<00:00, 365.67it/s]
📊 Epoch 1 - Loss: 0.4155, Val Accuracy: 0.9079

🔄 Epoch 2/5
Training: 100%                                    6750/6750 [00:50<00:00, 122.71it/s, loss=0.6642]

Validation: 100%                                   750/750 [00:02<00:00, 384.53it/s]
📊 Epoch 2 - Loss: 0.2416, Val Accuracy: 0.9193

🔄 Epoch 3/5
Training: 100%                                    6750/6750 [00:51<00:00, 135.24it/s, loss=0.3018]

Validation: 100%                                   750/750 [00:01<00:00, 386.96it/s]
📊 Epoch 3 - Loss: 0.1839, Val Accuracy: 0.9190

🔄 Epoch 4/5
Training: 100%                                    6750/6750 [00:50<00:00, 134.77it/s, loss=0.0472]

Validation: 100%                                   750/750 [00:01<00:00, 390.70it/s]
📊 Epoch 4 - Loss: 0.1368, Val Accuracy: 0.9175

🔄 Epoch 5/5
Training: 100%                                    6750/6750 [00:51<00:00, 133.73it/s, loss=0.1063]

Validation: 100%                                   750/750 [00:01<00:00, 389.92it/s]
📊 Epoch 5 - Loss: 0.1001, Val Accuracy: 0.9124
💾 BiLSTM model saved to models/bilstm_model.pth
```

## 6. Model Evaluation & Comparison {#evaluation}

```
Start coding or generate with AI.
```

```
# Evaluation function
def evaluate_model(model, test_loader, model_type="distilbert"):
    model.eval()
    all_predictions = []
    all_labels = []

    with torch.no_grad():
        for batch in tqdm(test_loader, desc=f"Evaluating {model_type}"):
            if model_type == "distilbert":
                input_ids = batch['input_ids'].to(device)
                attention_mask = batch['attention_mask'].to(device)
                labels = batch['label'].to(device)

                outputs = model(input_ids=input_ids, attention_mask=attention_mask)
                predictions = torch.argmax(outputs.logits, dim=-1)
            else:  # bilstm
                input_ids = batch['input_ids'].to(device)
                labels = batch['label'].to(device)

                outputs, _ = model(input_ids)
                predictions = torch.argmax(outputs, dim=-1)

            all_predictions.extend(predictions.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_predictions), np.array(all_labels)

# Evaluate both models
print("📊 Evaluating models on test set...")

# DistilBERT evaluation
distilbert_preds, distilbert_labels = evaluate_model(model, test_loader, "distilbert")
distilbert_accuracy = accuracy_score(distilbert_labels, distilbert_preds)

# BiLSTM evaluation
bilstm_preds, bilstm_labels = evaluate_model(bilstm_model, bilstm_test_loader, "bilstm")
bilstm_accuracy = accuracy_score(bilstm_labels, bilstm_preds)

print(f"\n🎯 Test Results:")
print(f"DistilBERT Accuracy: {distilbert_accuracy:.4f}")
print(f"BiLSTM Accuracy: {bilstm_accuracy:.4f}")
```

```
📊 Evaluating models on test set...
Evaluating distilbert: 100%  475/475 [00:29<00:00, 16.41it/s]
Evaluating bilstm: 100%  475/475 [00:01<00:00, 387.85it/s]

🎯 Test Results:
DistilBERT Accuracy: 0.9351
BiLSTM Accuracy: 0.9133
```

```
# Detailed classification reports
print("📊 DistilBERT Classification Report:")
print(classification_report(distilbert_labels, distilbert_preds, target_names=processor.cla

print("\n📊 BiLSTM Classification Report:")
print(classification_report(bilstm_labels, bilstm_preds, target_names=processor.class_names
```

```
📊 DistilBERT Classification Report:
                    precision    recall  f1-score   support

             World       0.96      0.93      0.94      1900
            Sports       0.98      0.99      0.98      1900
          Business       0.91      0.89      0.90      1900
Science/Technology       0.89      0.93      0.91      1900

          accuracy                           0.94      7600
         macro avg       0.94      0.94      0.94      7600
      weighted avg       0.94      0.94      0.94      7600


📊 BiLSTM Classification Report:
                    precision    recall  f1-score   support

             World       0.93      0.92      0.92      1900
            Sports       0.96      0.98      0.97      1900
          Business       0.86      0.89      0.88      1900
Science/Technology       0.90      0.86      0.88      1900

          accuracy                           0.91      7600
         macro avg       0.91      0.91      0.91      7600
      weighted avg       0.91      0.91      0.91      7600
```

## 7. Interactive Prediction {#prediction}

```
# Interactive prediction function
def predict_text(text, use_distilbert=True):
    """Predict the class of a given text"""
    # Clean text
    cleaned_text = processor.clean_text(text)

    if use_distilbert:
        # DistilBERT prediction
        model.eval()
        encoding = tokenizer(
            cleaned_text,
            truncation=True,
            padding='max_length',
            max_length=128,
            return_tensors='pt'
        )

        input_ids = encoding['input_ids'].to(device)
        attention_mask = encoding['attention_mask'].to(device)

        with torch.no_grad():
            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
```

```
            probabilities = torch.softmax(outputs.logits, dim=-1)
            predicted_class = torch.argmax(probabilities, dim=-1).item()
            confidence = probabilities[0][predicted_class].item()

    else:
        # BiLSTM prediction
        bilstm_model.eval()
        encoded = bilstm_tokenizer.encode([cleaned_text], max_length=128)
        input_ids = encoded.to(device)

        with torch.no_grad():
            outputs, attention_weights = bilstm_model(input_ids)
            probabilities = torch.softmax(outputs, dim=-1)
            predicted_class = torch.argmax(probabilities, dim=-1).item()
            confidence = probabilities[0][predicted_class].item()

    return {
        'text': text,
        'cleaned_text': cleaned_text,
        'predicted_class': processor.class_names[predicted_class],
        'confidence': confidence,
        'all_probabilities': {processor.class_names[i]: probabilities[0][i].item() for i in
    }

# Test with sample texts
sample_texts = [
    "Apple Inc. reports record quarterly earnings with strong iPhone sales",
    "Scientists discover new exoplanet using advanced telescope technology",
    "Lakers defeat Warriors in overtime thriller at Staples Center",
    "Breaking: Political tensions rise as world leaders meet for summit"
]

print("🔮 Sample Predictions:")
for text in sample_texts:
    print(f"\n📝 Text: {text}")

    # DistilBERT prediction
    distilbert_result = predict_text(text, use_distilbert=True)
    print(f"🤖 DistilBERT: {distilbert_result['predicted_class']} (confidence: {distilbert_

    # BiLSTM prediction
    bilstm_result = predict_text(text, use_distilbert=False)
    print(f"🧠 BiLSTM: {bilstm_result['predicted_class']} (confidence: {bilstm_result['con
```

```
🔮 Sample Predictions:

📝 Text: Apple Inc. reports record quarterly earnings with strong iPhone sales
🤖 DistilBERT: Science/Technology (confidence: 0.737)
🧠 BiLSTM: Business (confidence: 0.567)

📝 Text: Scientists discover new exoplanet using advanced telescope technology
🤖 DistilBERT: World (confidence: 0.508)
🧠 BiLSTM: Science/Technology (confidence: 1.000)

📝 Text: Lakers defeat Warriors in overtime thriller at Staples Center
🤖 DistilBERT: Sports (confidence: 1.000)
🧠 BiLSTM: Sports (confidence: 0.997)

📝 Text: Breaking: Political tensions rise as world leaders meet for summit
```

```
🤖 DistilBERT: World (confidence: 0.986)
🧠 BiLSTM: World (confidence: 0.998)
```

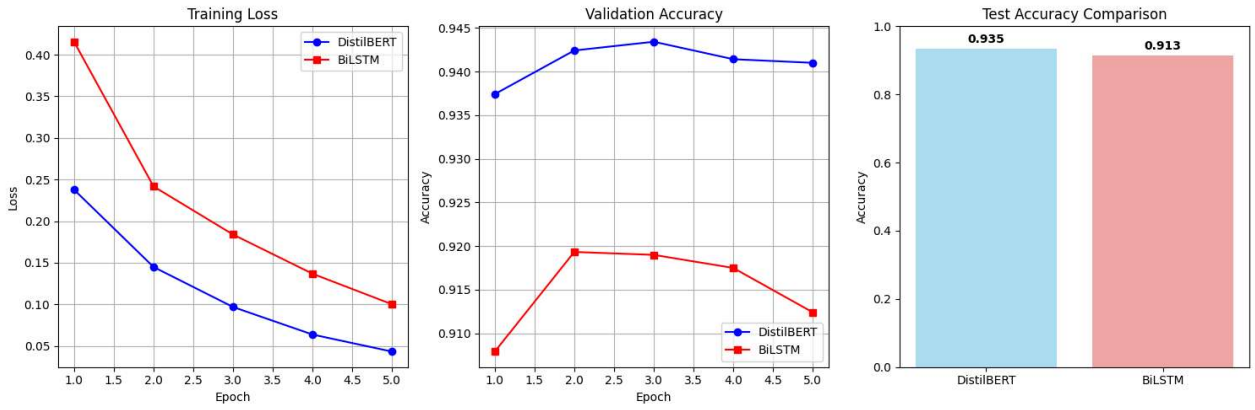
```
# Interactive prediction cell - modify the text below to test your own examples
YOUR_TEXT = "Enter your text here for classification"  # <-- Modify this line

print(f"🔮 Classifying: '{YOUR_TEXT}'")
print("\n" + "="*50)

# Get predictions from both models
distilbert_result = predict_text(YOUR_TEXT, use_distilbert=True)
bilstm_result = predict_text(YOUR_TEXT, use_distilbert=False)

print(f"🤖 DistilBERT Prediction:")
print(f"   Class: {distilbert_result['predicted_class']}")
print(f"   Confidence: {distilbert_result['confidence']:.3f}")
print(f"   All probabilities: {distilbert_result['all_probabilities']}")

print(f"\n🧠 BiLSTM Prediction:")
print(f"   Class: {bilstm_result['predicted_class']}")
print(f"   Confidence: {bilstm_result['confidence']:.3f}")
print(f"   All probabilities: {bilstm_result['all_probabilities']}")

print("\n" + "="*50)
agreement = "✅ AGREE" if distilbert_result['predicted_class'] == bilstm_result['predicted
print(f"Models {agreement} on classification")
```

```
🔮 Classifying: 'Enter your text here for classification'

==================================================
🤖 DistilBERT Prediction:
   Class: Science/Technology
   Confidence: 0.989
   All probabilities: {'World': 0.010264690965414047, 'Sports': 0.00010852619743673131, 'Bus

🧠 BiLSTM Prediction:
   Class: Science/Technology
   Confidence: 0.638
   All probabilities: {'World': 0.008440869860351086, 'Sports': 7.14335692464374e-05, 'Busin

==================================================
Models ✅ AGREE on classification
```

## 8. Results Visualization {#visualization}

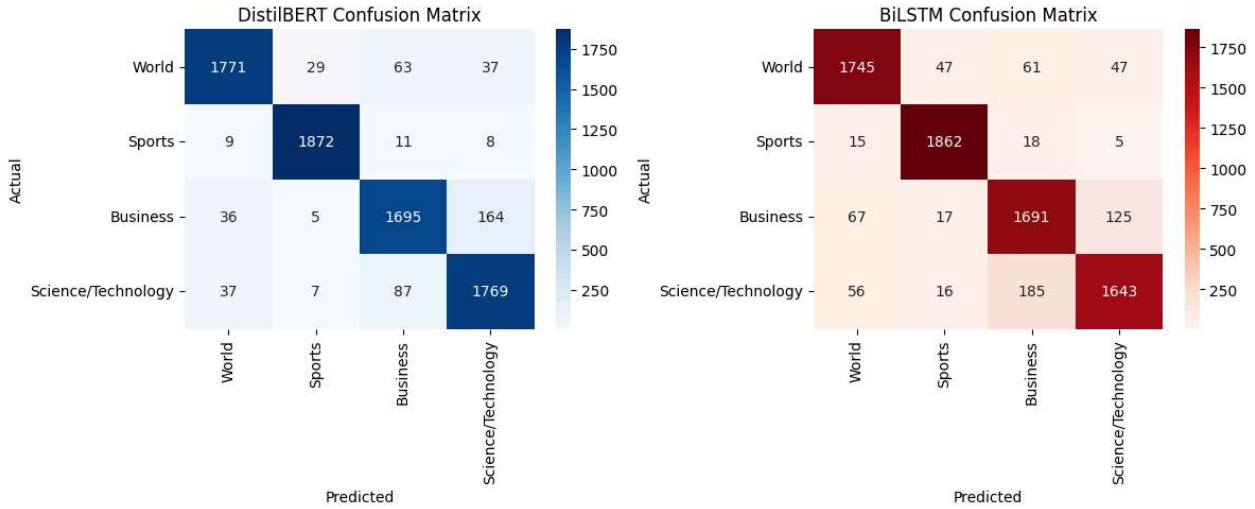
```
# Training curves comparison
plt.figure(figsize=(15, 5))

# Training losses
plt.subplot(1, 3, 1)
plt.plot(range(1, len(train_losses)+1), train_losses, 'b-', label='DistilBERT', marker='o')
plt.plot(range(1, len(bilstm_train_losses)+1), bilstm_train_losses, 'r-', label='BiLSTM', m
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.legend()
plt.grid(True)

# Validation accuracies
plt.subplot(1, 3, 2)
plt.plot(range(1, len(val_accuracies)+1), val_accuracies, 'b-', label='DistilBERT', marker=
plt.plot(range(1, len(bilstm_val_accuracies)+1), bilstm_val_accuracies, 'r-', label='BiLSTM
plt.title('Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Final comparison
plt.subplot(1, 3, 3)
models = ['DistilBERT', 'BiLSTM']
accuracies = [distilbert_accuracy, bilstm_accuracy]
colors = ['skyblue', 'lightcoral']

bars = plt.bar(models, accuracies, color=colors, alpha=0.7)
plt.title('Test Accuracy Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)

# Add value labels on bars
for bar, acc in zip(bars, accuracies):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{acc:.3f}', ha='center', va='bottom', fontweight='bold')

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




```
# Confusion matrices
plt.figure(figsize=(12, 5))

# DistilBERT confusion matrix
plt.subplot(1, 2, 1)
cm_distilbert = confusion_matrix(distilbert_labels, distilbert_preds)
sns.heatmap(cm_distilbert, annot=True, fmt='d', cmap='Blues',
            xticklabels=processor.class_names, yticklabels=processor.class_names)
plt.title('DistilBERT Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
# BiLSTM confusion matrix
plt.subplot(1, 2, 2)
cm_bilstm = confusion_matrix(bilstm_labels, bilstm_preds)
sns.heatmap(cm_bilstm, annot=True, fmt='d', cmap='Reds',
            xticklabels=processor.class_names, yticklabels=processor.class_names)
plt.title('BiLSTM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

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




```
# Model summary and final results
print("🎯 FINAL RESULTS SUMMARY")
print("="*50)
print(f"📊 Dataset: AG News Classification")
print(f"🔢 Classes: {len(processor.class_names)} ({', '.join(processor.class_names)})")
print(f"📈 Training samples: {len(train_texts):,}")
print(f"🔍 Test samples: {len(test_texts):,}")
print(f"💻 Device: {device}")

print(f"\n🤖 DistilBERT Model:")
print(f"   • Architecture: DistilBERT-base-uncased fine-tuned")
print(f"   • Parameters: ~66M")
print(f"   • Test Accuracy: {distilbert_accuracy:.4f} ({distilbert_accuracy*100:.2f}%)")
print(f"   • Training Epochs: {EPOCHS}")

print(f"\n🧠 BiLSTM Model:")
print(f"   • Architecture: BiLSTM + Attention")
print(f"   • Vocabulary Size: {vocab_size:,}")
print(f"   • Parameters: {sum(p.numel() for p in bilstm_model.parameters()):,}")
print(f"   • Test Accuracy: {bilstm_accuracy:.4f} ({bilstm_accuracy*100:.2f}%)")
print(f"   • Training Epochs: {BILSTM_EPOCHS}")

print(f"\n🏆 Best Model: {'DistilBERT' if distilbert_accuracy > bilstm_accuracy else 'BiLSTM
print(f"🎯 Performance Difference: {abs(distilbert_accuracy - bilstm_accuracy)*100:.2f}%")

print("\n✅ All models trained and evaluated successfully!")
print("💾 Models saved in 'models/' directory")
```

```
print("📊 Ready for deployment or further analysis")
```

```
🎯 FINAL RESULTS SUMMARY
==================================================
📊 Dataset: AG News Classification
🔢 Classes: 4 (World, Sports, Business, Science/Technology)
📈 Training samples: 108,000
🔍 Test samples: 7,600
💻 Device: cuda

🤖 DistilBERT Model:
   • Architecture: DistilBERT-base-uncased fine-tuned
   • Parameters: ~66M
   • Test Accuracy: 0.9351 (93.51%)
   • Training Epochs: 5

🧠 BiLSTM Model:
   • Architecture: BiLSTM + Attention
   • Vocabulary Size: 10,000
   • Parameters: 1,664,453
   • Test Accuracy: 0.9133 (91.33%)
   • Training Epochs: 5

🏆 Best Model: DistilBERT
🎯 Performance Difference: 2.18%

✅ All models trained and evaluated successfully!
💾 Models saved in 'models/' directory
📊 Ready for deployment or further analysis
```