BERT Mobile Training and Deployment Framework

A comprehensive end-to-end framework for training, fine-tuning, and deploying BERT models optimized for mobile devices with custom vocabularies and complete integration code.

🚀 Features

- 🔧 **Custom Vocabulary Building**: Create domain-specific vocabularies from your text data
- 📱 **Mobile Optimization**: Convert BERT models for iOS (CoreML) and Android (TensorFlow Lite)
- 🎯 **Fine-tuning Support**: Fine-tune pre-trained BERT models on custom datasets
- ⚡ **Performance Optimization**: Quantization, pruning, and mobile-specific optimizations
- 🛠️ **Complete Pipeline**: End-to-end workflow from data to deployment
- 📊 **Evaluation Tools**: Comprehensive model evaluation and performance benchmarking
- 📦 **Deployment Packages**: Ready-to-use integration code for mobile apps

📁 Project Structure

```
bert_mobile/
├── README.md                    # This file
├── requirements.txt             # Python dependencies
├── config/                      # Configuration files
│   ├── model_config.yaml       # Model architecture settings
│   ├── training_config.yaml    # Training hyperparameters
│   ├── mobile_config.yaml      # Mobile optimization settings
│   └── vocab_config.yaml       # Vocabulary building parameters
├── src/                         # Core source code
│   ├── __init__.py
│   ├── model_downloader.py     # Download BERT models from Hugging Face
│   ├── vocab_builder.py        # Build custom vocabularies
│   ├── data_processor.py       # Data preprocessing and tokenization
│   ├── model_trainer.py        # BERT training with mobile optimizations
│   ├── mobile_converter.py     # Convert models for mobile deployment
│   ├── tokenizer_utils.py      # Tokenizer utilities and analysis
│   └── deploy.py               # Create deployment packages
├── scripts/                     # Training and deployment scripts
│   ├── download_model.py       # Download BERT models
│   ├── build_vocabulary.py     # Build custom vocabularies
│   ├── prepare_data.py         # Prepare training data
│   ├── train_bert.py           # Train BERT from scratch
│   ├── fine_tune_bert.py       # Fine-tune pre-trained models
│   ├── convert_to_mobile.py    # Convert for mobile deployment
│   ├── evaluate_model.py       # Evaluate model performance
│   └── deploy.py               # Create deployment packages
├── data/                        # Data directories
│   ├── raw/                    # Raw text data
│   ├── processed/              # Processed training data
│   ├── vocabularies/           # Custom vocabularies
│   └── tokenized/              # Tokenized data
├── models/                      # Model storage
│   ├── pretrained/             # Downloaded pre-trained models
│   ├── trained/                # Trained models
│   ├── fine_tuned/             # Fine-tuned models
│   └── mobile/                 # Mobile-optimized models
├── notebooks/                   # Jupyter notebooks for analysis
│   ├── vocabulary_analysis.ipynb
│   ├── model_comparison.ipynb
│   └── mobile_optimization.ipynb
└── deployment/                  # Mobile deployment packages
    ├── android/                # Android integration code
    └── ios/                    # iOS integration code
```

🚀 Quick Start

1. Installation

```bash
git clone <repository-url>
cd bert_mobile
pip install -r requirements.txt
```

2. Download a Pre-trained Model

```bash
# Download BERT base model
python scripts/download_model.py --model_name bert-base-uncased --output_dir ./mo

# List available models
python scripts/download_model.py --list_models

# Get model recommendations
python scripts/download_model.py --recommend mobile
```

3. Build Custom Vocabulary (Optional)

```bash
# Build vocabulary from your text data
python scripts/build_vocabulary.py \
    --input_dir ./data/raw \
    --output_dir ./data/vocabularies \
    --vocab_size 25000 \
    --vocab_name custom_vocab \
    --analyze_domain
```

4. Prepare Training Data

```bash
# For classification tasks
python scripts/prepare_data.py \
    --input_dir ./data/raw/classification.csv \
    --vocab_path ./data/vocabularies/custom_vocab.txt \
    --output_dir ./data/processed \
    --task_type classification \
    --text_column text \
    --label_column label

# For language modeling
python scripts/prepare_data.py \
    --input_dir ./data/raw/texts/ \
    --vocab_path ./models/pretrained/bert-base-uncased \
    --output_dir ./data/processed \
    --task_type language_modeling
```

5. Fine-tune Model

```bash
# Fine-tune with mobile optimizations
python scripts/fine_tune_bert.py \
    --model_path ./models/pretrained/bert-base-uncased \
    --train_data ./data/processed/train.json \
    --val_data ./data/processed/val.json \
    --custom_vocab ./data/vocabularies/custom_vocab.txt \
    --output_dir ./models/fine_tuned \
    --num_epochs 3 \
    --learning_rate 2e-5 \
    --enable_distillation
```

6. Convert for Mobile

```bash
# Convert to mobile formats
python scripts/convert_to_mobile.py \
    --model_path ./models/fine_tuned \
    --output_dir ./models/mobile \
    --platforms both \
    --sequence_length 128 \
    --quantize \
    --validate \
    --benchmark
```

7. Create Deployment Package

```bash
# Create deployment packages with integration code
python scripts/deploy.py \
    --model_dir ./models/mobile \
    --platform both \
    --output_dir ./deployment \
    --create_archive
```

📱 Mobile Integration

iOS (Swift)

```swift
import BERTMobile

// Initialize BERT model
do {
    let bert = try BERTMobile()

    // Classify text
    let (label, confidence) = try bert.classifyText("This is a great product!")
    print("Classification: \(label) (\(confidence * 100)% confidence)")

} catch {
    print("Error: \(error)")
}
```

Android (Kotlin)

```kotlin
class MainActivity : AppCompatActivity() {
    private lateinit var bert: BERTMobile

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Initialize BERT model
        bert = BERTMobile(this)

        // Classify text
        lifecycleScope.launch {
            val (label, confidence) = bert.classifyText("This is a great product!'
            println("Classification: $label (${confidence * 100}% confidence)")
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        bert.close()
    }
}
```

🔧 Advanced Usage

Custom Training from Scratch

```bash
python scripts/train_bert.py \
    --model_path ./models/pretrained/bert-base-uncased \
    --train_data ./data/processed/train.json \
    --val_data ./data/processed/val.json \
    --output_dir ./models/trained \
    --task_type classification \
    --num_labels 3 \
    --enable_mobile_optimizations
```

Model Evaluation

```bash
# Evaluate PyTorch model
python scripts/evaluate_model.py \
    --model_path ./models/fine_tuned \
    --test_data ./data/processed/test.json \
    --platform pytorch \
    --detailed

# Benchmark mobile model
python scripts/evaluate_model.py \
    --model_path ./models/mobile/android/bert_mobile.tflite \
    --test_data ./data/processed/test.json \
    --platform android \
    --benchmark
```

Vocabulary Analysis

```python
from src.tokenizer_utils import BERTTokenizerUtils

# Analyze tokenization
tokenizer_utils = BERTTokenizerUtils('bert-base-uncased')
stats = tokenizer_utils.analyze_tokenization(your_texts)

print(f"Average tokens per text: {stats['avg_tokens_per_text']:.2f}")
print(f"Subword ratio: {stats['subword_ratio']:.2%}")
print(f"OOV ratio: {stats['oov_ratio']:.2%}")
```

⚙️ Configuration

Training Configuration (`config/training_config.yaml`)

```yaml
training:
  num_epochs: 3
  batch_size: 16
  learning_rate: 2e-5

  # Mobile optimizations
  mobile_training:
    knowledge_distillation: true
    teacher_model: "bert-base-uncased"
    temperature: 4.0
    alpha: 0.7

  # LoRA fine-tuning
  lora:
    enabled: true
    r: 16
    alpha: 32
    dropout: 0.1
    target_modules: ["q_proj", "v_proj", "k_proj", "o_proj"]
```

Mobile Configuration (`config/mobile_config.yaml`)

```yaml
mobile:
  ios:
    compute_units: "neural_engine"
    precision: "float16"
    optimization:
      quantize_weights: true

  android:
    delegates: ["gpu", "nnapi"]
    optimization:
      quantization: "dynamic"

  performance:
    max_latency_ms: 100
    max_memory_mb: 200
    min_accuracy: 0.85
```

📊 Performance Targets

Platform	Model Size	Inference Time	Memory Usage	Accuracy
iOS	< 50MB	< 100ms	< 200MB	> 85%
Android	< 50MB	< 150ms	< 250MB	> 85%

🛠️ Key Components

Model Downloader

- Downloads BERT models from Hugging Face Hub
- Supports multiple model variants (base, large, distilled)

- Validates model compatibility for mobile deployment
- Provides model recommendations based on use case

Vocabulary Builder

- Creates custom vocabularies from domain-specific text
- WordPiece tokenization with configurable parameters
- Domain term analysis and statistics
- Vocabulary optimization for mobile deployment

Mobile Trainer

- Full BERT training with mobile-specific optimizations
- Knowledge distillation from larger teacher models
- Layer freezing and parameter reduction
- LoRA/QLoRA support for efficient fine-tuning

Mobile Converter

- Converts PyTorch models to CoreML (iOS) and TensorFlow Lite (Android)
- Applies quantization and optimization techniques
- Validates conversion accuracy
- Benchmarks mobile model performance

Deployment Manager

- Creates complete deployment packages
- Generates native integration code (Swift/Kotlin)
- Includes example applications and documentation
- Provides troubleshooting guides

🔍 Evaluation and Benchmarking

Model Accuracy

```
bash

python scripts/evaluate_model.py \
    --model_path ./models/mobile/ios/BERTMobile.mlmodel \
    --test_data ./data/processed/test.json \
    --platform ios \
    --output ./results/ios_evaluation.json
```

Performance Benchmarking

```
bash

python scripts/evaluate_model.py \
    --model_path ./models/mobile/android/bert_mobile.tflite \
    --platform android \
    --benchmark \
    --output ./results/android_benchmark.json
```

📱 Deployment Packages

The deployment packages include:

iOS Package

- CoreML model file (`.mlmodel`)
- Swift Package Manager integration
- BERTMobile Swift class with tokenizer
- Example SwiftUI application
- Comprehensive documentation
- Xcode project template

Android Package

- TensorFlow Lite model (`.tflite`)
- Kotlin integration classes
- Example Android application
- Gradle build configuration
- Asset management utilities
- Performance optimization guides

🚨 Troubleshooting

Common Issues

1. **Model Loading Errors**
   - Verify model file paths and permissions
   - Check model format compatibility
   - Ensure sufficient device memory
2. **Poor Performance**
   - Enable hardware acceleration (GPU/Neural Engine)
   - Reduce model size through quantization
   - Optimize sequence length
3. **Accuracy Issues**
   - Validate tokenizer compatibility
   - Check input preprocessing
   - Compare with original PyTorch model
4. **Memory Issues**
   - Use quantized models
   - Implement proper resource cleanup
   - Monitor memory usage during inference

Performance Optimization Tips

1. **iOS Optimization**
   - Use Neural Engine when available

- Enable float16 precision
- Batch similar requests
- Cache model instance

2. **Android Optimization**
   - Enable GPU delegate
   - Use NNAPI when supported
   - Implement background processing
   - Consider dynamic quantization

📄 License

MIT License - see LICENSE for details.

🤝 Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add comprehensive tests
5. Submit a pull request

📚 Citation

If you use this framework in your research, please cite:

```bibtex
@software{bert_mobile_framework_2025,
  title={BERT Mobile: Complete Training and Deployment Framework},
  author={Your Name},
  year={2025},
  url={https://github.com/your-org/bert-mobile}
}
```

📞 Support

- 📖 Check the documentation in each deployment package
- 🐛 Report issues on GitHub
- 💬 Join our community discussions
- 📧 Contact: support@your-org.com

Ready to deploy BERT on mobile? Get started with the quick start guide above! 🚀