User Guide: Multilingual PDF RAG System

Table of Contents

1. Installation
2. Quick Start
3. Basic Operations
4. Advanced Features
5. Troubleshooting
6. Best Practices

Installation

Prerequisites

- Python 3.8 or higher
- Google Colab account (for cloud execution)
- 16GB+ RAM recommended
- GPU access (optional but recommended)

Step 1: Install System Dependencies

```
# Install Tesseract OCR with language packs
!apt-get install -y tesseract-ocr tesseract-ocr-hin tesseract-ocr-ben
!apt-get install -y tesseract-ocr-chi-sim tesseract-ocr-chi-tra
!apt-get install -y tesseract-ocr-ara tesseract-ocr-urd
!apt-get install -y poppler-utils
```

Step 2: Install Python Packages

```
!pip install -q pytesseract pdf2image PyPDF2 pymupdf sentence-transformers
!pip install -q langchain chromadb faiss-cpu rank-bm25 transformers accelerate bitsandbytes
!pip install -q langdetect pandas matplotlib seaborn pillow
```

Step 3: Setup Authentication (if using HuggingFace token)

```
from huggingface_hub import login
login(token="your_hf_token_here")
```

Quick Start

1. Initialize the System

```
# Run the complete system initialization
rag, evaluator = main()
```

This will:

- Load the embedding model (paraphrase-multilingual-MiniLM-L12-v2)
- Load the LLM (google/gemma-2-2b-it)
- Initialize all components
- Display system configuration

2. Ingest Your PDFs

```
# Place your PDFs in a folder
PDF_FOLDER = "/path/to/your/pdfs"


# Ingest all PDFs from the folder
rag.ingest_pdfs(PDF_FOLDER)
```

Expected Output:

- Processing status for each PDF
- Language detection results
- Number of chunks created per document
- OCR confidence scores (for scanned PDFs)
- Final ingestion summary

3. Query the System

```
# Ask a question
result = rag.query("What are the main topics?")
print(result["answer"])
```

Basic Operations

Document Ingestion

Supported Languages

- English (eng)
- Hindi (hin)
- Bengali (ben)
- Chinese Simplified (chi_sim)
- Chinese Traditional (chi_tra)
- Arabic (ara)
- Urdu (urd)

Supported PDF Types

- **Digital PDFs**: Text-based PDFs with selectable text
- **Scanned PDFs**: Image-based PDFs requiring OCR
- **Mixed PDFs**: Combination of both

Example: Ingest Documents

```
# Ingest from Google Drive
rag.ingest_pdfs("/content/drive/MyDrive/PDF")


# Ingest from local directory
rag.ingest_pdfs("./documents")
```

Querying

Simple Query

```
result = rag.query("What is the main conclusion?")
print(result["answer"])
```

View Full Results

```
result = rag.query("Explain the methodology")
print(f"Answer: {result['answer']}")
print(f"Sources: {result['sources']}")
print(f"Languages: {result['source_languages']}")
print(f"Latency: {result['latency']:.2f}s")
```

Disable Chat Memory (for independent queries)

```
result = rag.query("Your question here", use_memory=False)
```

Document Summarization

```
# Get summary of all ingested documents
summary = rag.summarize_documents()
print(summary)
```

Clear Chat Memory

```
# Clear conversation history
rag.clear_memory()
```

Advanced Features

1. Benchmarking Queries

```
# Define test queries
test_queries = [
    {"question": "What are the main findings?"},
    {"question": "Describe the methodology"},
    {"question": "What are the conclusions?"}
]


# Run benchmark
df = evaluator.benchmark_queries(test_queries)
print(df)
```

2. Test Retrieval Accuracy

```
# Define retrieval tests
retrieval_tests = [
    {
        "question": "What is mentioned about data?",
        "expected_source": "document.pdf",
        "expected_keywords": ["data", "information"]
```

```
    }
]


# Test accuracy
accuracy = evaluator.test_retrieval_accuracy(retrieval_tests)
print(f"Source Accuracy: {accuracy['source_accuracy']*100:.1f}%")
```

3. Test Multilingual Capability

```
# Test multilingual document handling
ml_results = evaluator.test_multilingual_capability()
print(f"Languages Supported: {ml_results['languages_supported']}")
print(f"Capability Score: {ml_results['capability_score']:.2f}")
```

4. Generate Performance Report

```
# Generate detailed performance report
evaluator.generate_performance_report("performance_report.txt")
```

5. Visualize Performance Metrics

```
# Generate performance plots
evaluator.plot_performance_metrics("performance_plots.png")
```

Troubleshooting

Common Issues

Issue 1: PDF Extraction Fails

Symptoms: "Failed to extract" message **Solutions**:

- Check if PDF is corrupted
- Ensure Tesseract is installed correctly
- Verify language packs are installed
- Try increasing OCR confidence threshold

Issue 2: Low OCR Confidence

Symptoms: OCR confidence < 60% **Solutions**:

- Check PDF image quality
- Ensure correct language pack is installed
- Consider preprocessing images (contrast enhancement)
- Manual review may be needed

Issue 3: Out of Memory

Symptoms: CUDA out of memory errors **Solutions**:

- Reduce batch size in configuration
- Clear GPU memory: torch.cuda.empty_cache()
- Use CPU instead of GPU
- Process documents in smaller batches

Issue 4: Slow Query Response

Symptoms: Latency > 30s **Solutions**:

- Check number of retrieved chunks (reduce top_k)
- Clear chat memory if very long
- Ensure GPU is being used
- Consider using smaller LLM

Debugging Tips

Check Ingestion Statistics

stats = rag.get_ingestion_stats()

print(f"Total Documents: {stats['total_files']}")

print(f"Successful: {stats['successful']}")

print(f"Failed: {stats['failed']}")

Inspect Retrieved Documents

result = rag.query("Your question")

```
for doc in result['retrieved_docs']:
    print(f"Source: {doc.metadata['source']}")
    print(f"Content: {doc.page_content[:200]}...")
    print("---")
```

Best Practices

Document Preparation

1. **Scan Quality**: Use 300 DPI or higher for scanned PDFs
2. **File Organization**: Group documents by language/topic
3. **Naming Convention**: Use descriptive filenames
4. **File Size**: Keep individual PDFs under 50MB for optimal processing

Query Formulation

1. **Be Specific**: Clear, focused questions get better answers
2. **Use Context**: Reference document names when relevant
3. **Follow-ups**: Leverage chat memory for related questions
4. **Complex Queries**: Break down into simpler sub-questions

System Maintenance

1. **Regular Cleanup**: Clear memory between unrelated query sessions
2. **Monitor Performance**: Use evaluation tools regularly
3. **Update Documents**: Re-ingest when documents are modified
4. **Backup**: Save ingestion statistics and configurations

Performance Optimization

1. **Batch Ingestion**: Process multiple documents at once
2. **GPU Utilization**: Ensure GPU is available and used
3. **Memory Management**: Clear unnecessary variables
4. **Caching**: Consider implementing query result caching for repeated questions

Configuration Options

Modify RAG Configuration

```
from dataclasses import replace
```

```
# Create custom configuration
custom_config = replace(
    config,
    chunk_size=256,  # Smaller chunks
    top_k_retrieval=10,  # Fewer results
    top_k_rerank=3  # Fewer reranked results
)

# Initialize with custom config
rag = RAGPipeline(custom_config)
```

Adjust Model Parameters

```
# Modify generation parameters in llm.generate()
# See LLMGenerator class for available options:
# - max_length: Maximum response length
# - temperature: Randomness (0.1-1.0)
# - top_p: Nucleus sampling threshold
```

Support and Feedback

For issues, questions, or feedback:

1. Check the troubleshooting section
2. Review technical documentation
3. Examine system logs for error messages
4. Contact development team with:
    - Error messages
    - System configuration
    - Sample queries that failed
    - Expected vs actual behavior