Scaffold AI System Handoff Document Technical Implementation and Operations Guide

Scaffold AI Development Team

September 10, 2025

Abstract

This handoff document provides comprehensive technical documentation for the Scaffold AI system, including deployment instructions, system architecture, operational procedures, and maintenance guidelines. The document covers the recent quality control bugfix implementation that achieved 100% success rates across all testing categories, ensuring reliable delivery of educational content to users.

Contents

1 System Overview

1.1 Purpose and Scope

The Scaffold AI system is a Flask-based web application designed to assist educators in integrating sustainability concepts into engineering curricula. The system leverages:

- **RAG Architecture**: Retrieval-Augmented Generation with 2,901 indexed documents
- **Vector Search**: FAISS-based semantic document retrieval
- **LLM Integration**: TinyLlama and Mistral-7B-Instruct model support
- **Quality Control**: Multi-stage response validation pipeline
- **Web Interface**: Enhanced UI with real-time query processing

1.2 Key Capabilities

1. Curriculum integration recommendations for sustainability concepts
2. Learning outcome development assistance
3. Framework integration guidance (LEED, UN SDGs, Engineering for One Planet)
4. Course-specific contextual recommendations
5. Resource and tool recommendations for educators

2 System Architecture

2.1 Component Overview

```
   Frontend                   Core Engine                 Data Layer



     Flask App                             Query
  P  r  o  c  e  s  s  o  r      FAISS Index
     Enhanced UI                 LLM Manager                      PDF
Documents
     Templates                   Quality  C o n t r o l
Metadata
                               Text Cleaning
```

Figure 1: High-level system architecture

2.2 Directory Structure

```
scaffold_ai/
          frontend/
                app_enhanced.py            # Main Flask application
                templates/
                    index_enhanced.html    # Enhanced UI template
          scaffold_core/
                config.py                  # System configuration
                llm.py                     # LLM integration layer
                text_clean.py              # Quality control algorithms
                pdf_processor.py           # Document processing
                vector/
                    enhanced_query_improved.py  # Core query engine
                    chunk.py               # Document chunking
          vector_outputs/
                scaffold_index_1.faiss     # Vector index
                scaffold_metadata_1.json   # Document metadata
                processed_1.json           # Processing logs
          data/                          # PDF document repository
          model_config.json              # Model configuration
          requirements.txt               # Python dependencies
```

3 Installation and Deployment

3.1 Prerequisites

- Python 3.12+
- CUDA-compatible GPU (recommended)
- 16GB+ RAM
- HuggingFace account and token

3.2 Environment Setup

```
# Create and activate virtual environment
python -m venv scaffold_env_312_py31210
scaffold_env_312_py31210/Scripts/activate  # Windows
# source scaffold_env_312_py31210/bin/activate  # Linux/Mac

# Install dependencies
pip install -r requirements.txt
```

Listing 1: Virtual environment creation

3.3 Configuration

3.3.1 Environment Variables

```
# Quality control configuration
export SC_ENABLE_PROOFREAD=1
export SC_GARBLED_STRICTNESS=medium
export SC_ENABLE_TRUNCATION_DETECTION=1

# HuggingFace integration
export HUGGINGFACE_TOKEN=hf_your_token_here
```

Listing 2: Required environment variables

3.3.2 Model Configuration

Edit `model_config.json`:

```
{
  "selected_models": {
    "llm": "tinyllama",          // or "mistral"
    "embedding": "miniLM"
  },
  "model_settings": {
    "llm": {
      "temperature": 0.2,
      "max_new_tokens": 1000,
      "top_p": 0.8
    },
    "embedding": {
      "chunk_size": 600,
      "chunk_overlap": 100
    }
  }
}
```

Listing 3: Model configuration file

3.4 Database Initialization

```
# Process PDF documents
python scaffold_core/scripts/chunk/ChunkTest.py

# Build vector index
python scaffold_core/vector/main.py

# Verify index creation
ls -la vector_outputs/
# Expected files:
# - scaffold_index_1.faiss (vector index)
# - scaffold_metadata_1.json (metadata)
# - processed_1.json (processing logs)
```

Listing 4: Vector database setup

4 Operations

4.1 Starting the System

```
# Activate environment
scaffold_env_312_py31210/Scripts/activate

# Start enhanced UI (recommended)
python frontend/app_enhanced.py --port 5004

# Alternative: Standard UI
python run_enhanced_ui.py
```

Listing 5: System startup

4.2 System Health Checks

```
# Test query processing
python -c "
import sys, os
sys.path.append('.')
os.environ['SC_ENABLE_PROOFREAD'] = '1'
os.environ['SC_GARBLED_STRICTNESS'] = 'medium'

from scaffold_core.vector.enhanced_query_improved import
    query_enhanced_improved
result = query_enhanced_improved('What is sustainability?')
print(f'Status: {"SUCCESS" if len(result.get("response", "")) > 100
    else "FAIL"}')
print(f'Response length: {len(result.get("response", ""))} characters')
"
```

Listing 6: Health check script

4.3 Performance Monitoring

Key metrics to monitor:

- **Response Time**: Target ¡30 seconds for complex queries
- **Success Rate**: Should maintain ¿95% non-fallback responses
- **Memory Usage**: Monitor GPU/CPU memory during operation
- **Error Rates**: Check logs for system exceptions

5 Quality Control System

5.1 Multi-Stage Pipeline

The quality control system implements a sophisticated validation pipeline:

1. **Initial Generation**: LLM produces candidate response

2. **Garbled Detection**: Check for obvious text corruption
3. **Surface Artifacts**: Scan for OCR-style artifacts
4. **Topic Validation**: Ensure response relevance
5. **Rewrite Attempt**: Attempt correction if issues found
6. **Final Decision**: Accept response or provide fallback

5.2 Configuration Parameters

Parameter	Values	Description
SC_ENABLE_PROOFREAD	0, 1	Enable/disable quality checking
SC_GARBLED_STRICTNESS	low, medium, high	Detection sensitivity
SC_ENABLE_TRUNCATION_DETECTION	0, 1	Detect incomplete responses

Table 1: Quality control configuration parameters

5.3 Pattern Detection Details

The refined surface artifact detection targets specific OCR corruption patterns:

```
# Patterns for detecting genuine OCR artifacts
inword_split_patterns = [
    r"\b(sustain|develop|architec|engin|build|energ)\s+(ed|ing|ment|t|
        ture|al|er|y)\b",
    r"\b(archite|buildi|sustaina|develo|enginee)\s+(cture|ng|bility|
        pment|ring)\b",
    r"\b(cur|fl|sys|eff)\s+(ricul|uid|tem|ici)\b",
]

# Examples of detected artifacts:
# "sustain ability" -> sustainability
# "archite cture" -> architecture
# "develop ment" -> development
```

Listing 7: Surface artifact patterns

6 Troubleshooting

6.1 Common Issues and Solutions

6.1.1 Model Loading Timeouts

Symptoms: Long delays during first query, timeout errors
Solution:

```
# Increase timeout and ensure sufficient memory
export PYTORCH_CUDA_ALLOC_CONF=max_split_size_mb:512
# Allow 2-3 minutes for initial model loading
```

6.1.2 High Fallback Rates

Symptoms: Frequent ”I apologize” responses
Solution:

```
# Adjust quality control strictness
export SC_GARBLED_STRICTNESS=low
# Or disable proofread temporarily for testing
export SC_ENABLE_PROOFREAD=0
```

6.1.3 Missing Vector Index

Symptoms: FileNotFoundError for FAISS files
Solution:

```
# Rebuild vector database
python scaffold_core/scripts/chunk/ChunkTest.py
python scaffold_core/vector/main.py
# Verify files exist in vector_outputs/
```

6.1.4 GPU Memory Issues

Symptoms: CUDA out of memory errors
Solution:

```
# Clear GPU cache
python -c "import torch; torch.cuda.empty_cache()"
# Reduce batch size in model configuration
# Use CPU fallback if necessary
```

6.2 Log Analysis

Important log locations and patterns:

```
# System logs show processing flow
grep "Enhanced query system initialized" logs/
grep "Generated response with" logs/
grep "WARNING.*Rejected response" logs/

# Error patterns to monitor
grep "ERROR" logs/
grep "FALLBACK" logs/
grep "FileNotFoundError" logs/
```

7 Maintenance

7.1 Regular Maintenance Tasks

7.1.1 Weekly

- Monitor system performance metrics
- Review error logs for patterns

- Verify disk space for model cache
- Test sample queries across categories

7.1.2 Monthly

- Update Python dependencies: `pip install -r requirements.txt --upgrade`
- Clear model cache: `rm -rf  /.cache/huggingface/`
- Backup vector database: `cp -r vector_outputs/ vector_outputs_backup/`
- Review and update test query suite

7.1.3 Quarterly

- Evaluate model performance against newer releases
- Review and update quality control patterns
- Conduct comprehensive testing across all categories
- Update documentation based on system changes

7.2 Performance Optimization

7.2.1 Model Selection

Model	**Size**	**Quality**	**Speed**
TinyLlama-1.1B	Small	Good	Fast
Mistral-7B-Instruct	Large	Excellent	Moderate

Table 2: Model performance characteristics

7.2.2 Hardware Optimization

- **GPU**: NVIDIA RTX 2080 Ti or better recommended
- **RAM**: 16GB minimum, 32GB recommended
- **Storage**: SSD for model cache and vector database
- **Network**: Stable internet for HuggingFace model downloads

8 Testing Framework

8.1 Automated Testing

```python
#!/usr/bin/env python3
"""Automated testing script for Scaffold AI system."""

import os
import sys
sys.path.append('.')

# Set environment
os.environ.update({
    'SC_ENABLE_PROOFREAD': '1',
    'SC_GARBLED_STRICTNESS': 'medium',
    'SC_ENABLE_TRUNCATION_DETECTION': '1'
})

from scaffold_core.vector.enhanced_query_improved import
   query_enhanced_improved

test_queries = [
    "How can I incorporate sustainability into my Fluid Mechanics
       course?",
    "What are some learning outcomes for environmental justice?",
    "Sustainability in fluids?",  # Brief query test
    "Integration of LEED principles into curriculum?"
]

def run_tests():
    results = []
    for i, query in enumerate(test_queries, 1):
        try:
            result = query_enhanced_improved(query)
            response = result.get('response', '')
            is_fallback = 'apologize' in response.lower() and 'having
               trouble' in response.lower()
            status = 'FALLBACK' if is_fallback else 'SUCCESS'
            results.append({
                'query': query,
                'status': status,
                'length': len(response)
            })
            print(f"Test {i}: {status} ({len(response)} chars)")
        except Exception as e:
            results.append({'query': query, 'status': 'ERROR', 'length'
               : 0})
            print(f"Test {i}: ERROR - {str(e)}")

    # Summary
    success_count = sum(1 for r in results if r['status'] == 'SUCCESS')
    print(f"\nResults: {success_count}/{len(results)} successful")
    return success_count == len(results)

if __name__ == "__main__":
    success = run_tests()
    sys.exit(0 if success else 1)
```

Listing 8: Automated test script

8.2 Test Categories

Comprehensive testing should cover:

1. **Core Integration**: Subject-specific sustainability integration
2. **Learning Outcomes**: Pedagogical goal development
3. **Framework Integration**: Standards and frameworks
4. **Course Context**: Detailed scenario handling
5. **Brief Queries**: Minimal input processing
6. **Advanced Integration**: Complex conceptual queries

9 Security Considerations

9.1 Data Protection

- **API Keys**: Store HuggingFace tokens securely, rotate regularly
- **User Data**: No persistent storage of user queries
- **Document Access**: Ensure PDF sources are appropriately licensed
- **Network Security**: Use HTTPS in production deployments

9.2 Model Security

- **Model Provenance**: Use verified models from HuggingFace
- **Input Validation**: Sanitize user inputs before processing
- **Output Filtering**: Quality control prevents harmful outputs
- **Resource Limits**: Implement timeouts and memory limits

10 Support and Escalation

10.1 Issue Classification

Severity	**Response Time**	**Examples**
Critical	Immediate	System completely down, data loss
High	4 hours	High fallback rates, performance degradation
Medium	24 hours	Feature issues, minor bugs
Low	1 week	Documentation updates, enhancement requests

Table 3: Issue severity classification

10.2 Contact Information

- **Repository**: `https://github.com/kevinmastascusa/scaffold_ai`
- **Issues**: GitHub Issues for bug reports and feature requests
- **Documentation**: This handoff document and README files
- **Technical Specs**: Research report and test results

11 Appendices

11.1 Appendix A: Configuration Files

11.1.1 requirements.txt

```
flask==2.3.3
torch>=2.0.0
transformers>=4.30.0
sentence-transformers>=2.2.0
faiss-cpu>=1.7.0
numpy>=1.24.0
pandas>=2.0.0
PyPDF2>=3.0.0
python-dotenv>=1.0.0
```

11.1.2 Sample model_config.json

```
{
  "selected_models": {
    "llm": "tinyllama",
    "embedding": "miniLM"
  },
  "model_settings": {
    "llm": {
      "temperature": 0.2,
      "max_new_tokens": 1000,
      "top_p": 0.8
    },
    "embedding": {
      "chunk_size": 600,
      "chunk_overlap": 100
    }
  }
}
```

11.2 Appendix B: Performance Benchmarks

Query Type	Response Time	Success Rate	Avg Length	Quality
Brief Queries	15-20s	100%	2,000+ chars	High
Standard Queries	20-30s	100%	1,500+ chars	High
Complex Queries	30-45s	100%	2,500+ chars	High
Framework Queries	25-35s	100%	2,000+ chars	High

Table 4: Performance benchmark data

11.3 Appendix C: Error Codes

Code	Description	Action
FAISS_001	Vector index not found	Rebuild index
LLM_001	Model loading timeout	Increase timeout, check GPU
QC_001	High fallback rate	Adjust quality parameters
MEM_001	Out of memory	Reduce batch size, clear cache

Table 5: Common error codes and resolutions