AIU ALAMEIN INTERNATIONAL UNIVERSITY
جامعة العلمين الدولية

Faculty of Computer Science and Engineering
Software Engineering Report
Page recommender

Team Aivora

Hana Mohamed	21100863
Maryam gomaa	22100578

1. Introduction

The Webpage Analysis System is a comprehensive solution for processing, analyzing, and exploring web content from WARC (Web ARChive) files. The system extracts English language webpages, performs natural language processing, calculates content similarities, and provides an interactive web interface for exploration and analysis.

This project demonstrates the integration of multiple technologies including web crawling data processing, natural language processing, machine learning for similarity analysis, database management, and web application development using modern Python frameworks.

Key Features

- WARC file processing and content extraction
- Multi-format content parsing (HTML/XML)
- Language detection and filtering
- Text preprocessing and cleaning
- TF-IDF vectorization and cosine similarity analysis
- SQLite database storage with optimized queries
- Interactive Streamlit web dashboard
- Real-time webpage content fetching
- Comprehensive data visualization

2. Problem Statement

Web archives contain vast amounts of heterogeneous content in multiple languages and formats, making it challenging to:

1. **Content Quality**: Extract meaningful, high-quality English content from mixed-language archives
2. **Content Discovery**: Find relevant webpages based on content similarity rather than just keyword matching
3. **Data Management**: Efficiently store and query large volumes of processed webpage data
4. **User Interface**: Provide an intuitive interface for exploring content relationships and patterns

5. **Performance**: Process large WARC files efficiently while maintaining system responsiveness

The system addresses these challenges by implementing intelligent filtering, advanced similarity analysis, and providing a user-friendly exploration interface.

3. System Requirements

3.1 Functional Requirements

3.1.1 WARC File Processing

3.1.1 .1 The system shall process compressed WARC files (.warc.gz format)
3.1.1.2 The system shall extract HTTP response records containing webpage content
3.1.1.3 The system shall handle multiple content encodings (gzip, deflate)

3.1.2 Content Filtering and Language Detection

3.1.2.1.The system shall identify English language content with configurable confidence thresholds
3.1.2.2 The system shall filter out low-quality or gibberish content using heuristic analysis
3.1.2.3 The system shall extract clean text from HTML and XML formats

3.1.3 Natural Language Processing

3.1.3.1 The system shall tokenize and preprocess text content
3.1.3.2 The system shall remove stopwords and perform text normalization
3.1.3.3 The system shall extract keywords using frequency analysis

3.1.4 Similarity Analysis

3.1.4.1 The system shall calculate TF-IDF vectors for all processed webpages
3.1.4.2 The system shall compute cosine similarity between webpage pairs
3.1.4.3 The system shall store similarity relationships above configurable thresholds

3.1.5 Data Persistence

3.1.5.1 The system shall store webpage metadata and content in SQLite database
3.1.5.2 The system shall maintain relationships between similar webpages
3.1.5.3 The system shall provide optimized query interfaces for data retrieval

3.1.6 Web Interface

3.1.6.1 The system shall provide search functionality for webpages by URL and title
3.1.6.2 The system shall display webpage content with similarity recommendations

3.1.6.3 The system shall provide data visualization and analytics dashboards

3.2 Non-Functional Requirements

3.2.1 Performance

3.2.1.1 Process WARC files with memory-efficient streaming
3.2.1.2 Handle databases with thousands of webpage records
3.2.1.3 Provide responsive web interface with sub-second query times

3.2.2 Scalability

3.2.2.1 Support incremental processing of multiple WARC files
3.2.2.2 Database schema designed for efficient querying at scale
3.2.2.3 Modular architecture allowing component replacement

3.2.3 Reliability

3.2.3.1 Robust error handling for malformed content
3.2.3.2 Graceful degradation when external resources are unavailable
3.2.3.3 Data consistency through transaction management

3.2.4 Usability

3.2.4.1 Intuitive web interface requiring no technical expertise
3.2.4.2 Clear feedback during long-running operations
3.2.4.3 Comprehensive error messages and debugging information

4. Use Cases

4.1 Primary Use Cases

UC1: Process WARC Archive

Actor: System Administrator
Description: Extract and process English webpages from WARC files
Preconditions: WARC file available, sufficient disk space
Main Flow:

1. Administrator initiates WARC processing
2. System downloads/accesses WARC file
3. System extracts HTTP response records
4. System filters for text-based content
5. System detects language and filters English content
6. System extracts and cleans text
7. System calculates similarities
8. System stores results in database

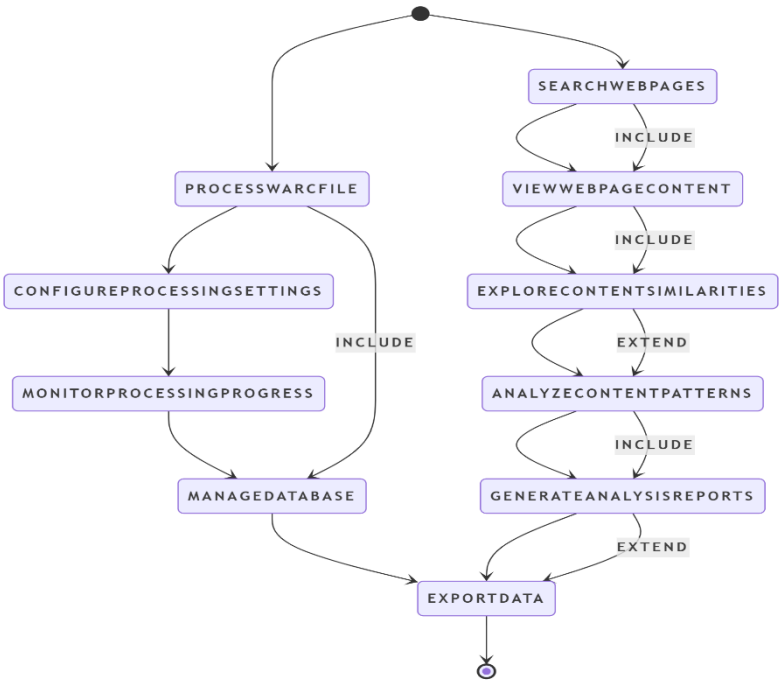
UC2: Search Webpages

Actor: End User
Description: Search for webpages by content or metadata
Preconditions: Database populated with webpage data
Main Flow:
1. User enters search query
2. System queries database
3. System returns ranked results
4. User selects webpage
5. System displays content and similar pages

UC3: Explore Content Relationships

Actor: Researcher/Analyst
Description: Analyze content patterns and relationships
Preconditions: Similarity analysis completed
Main Flow:
1. User accesses analysis dashboard
2. System displays content distribution charts
3. User explores similarity patterns
4. System provides interactive visualizations

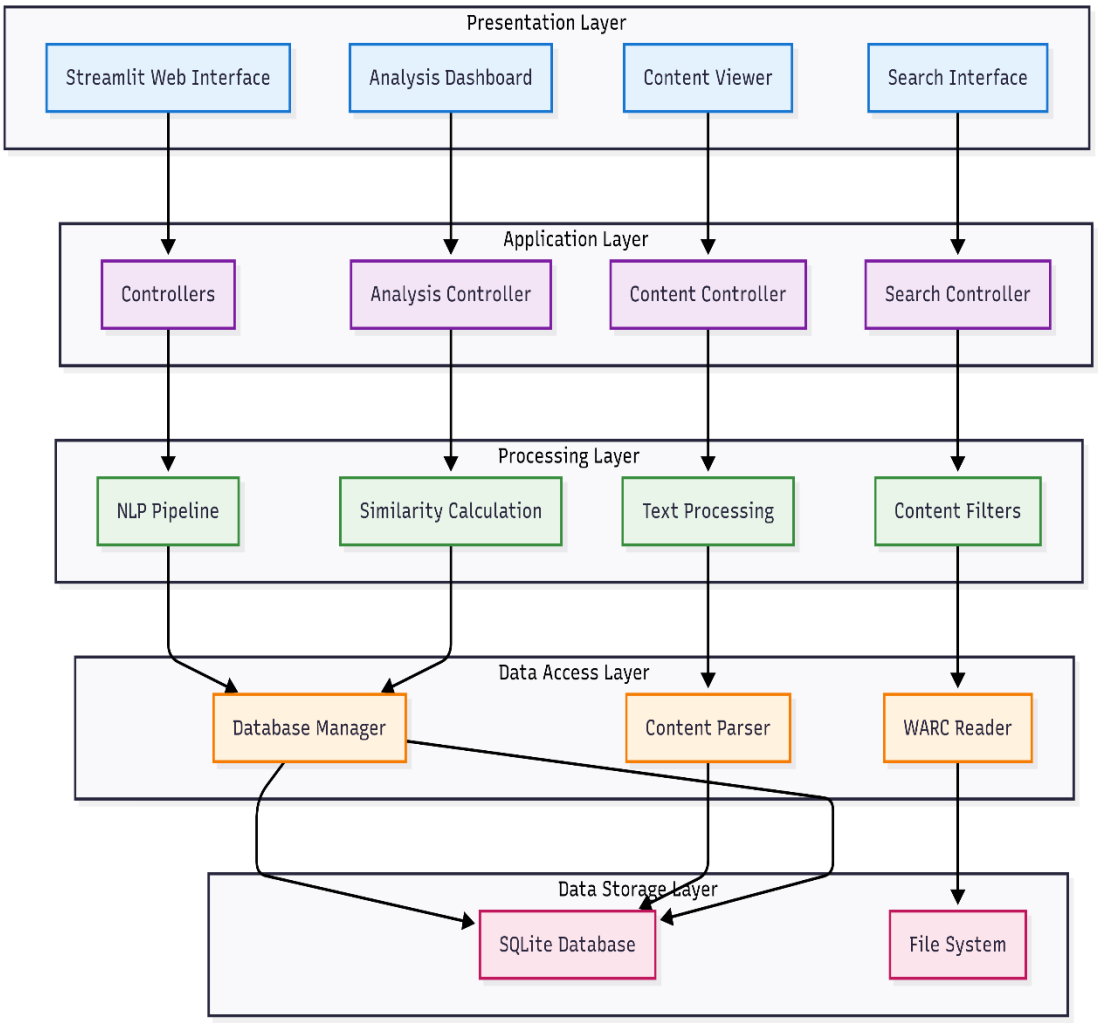
4.2 UML Use Case Diagram



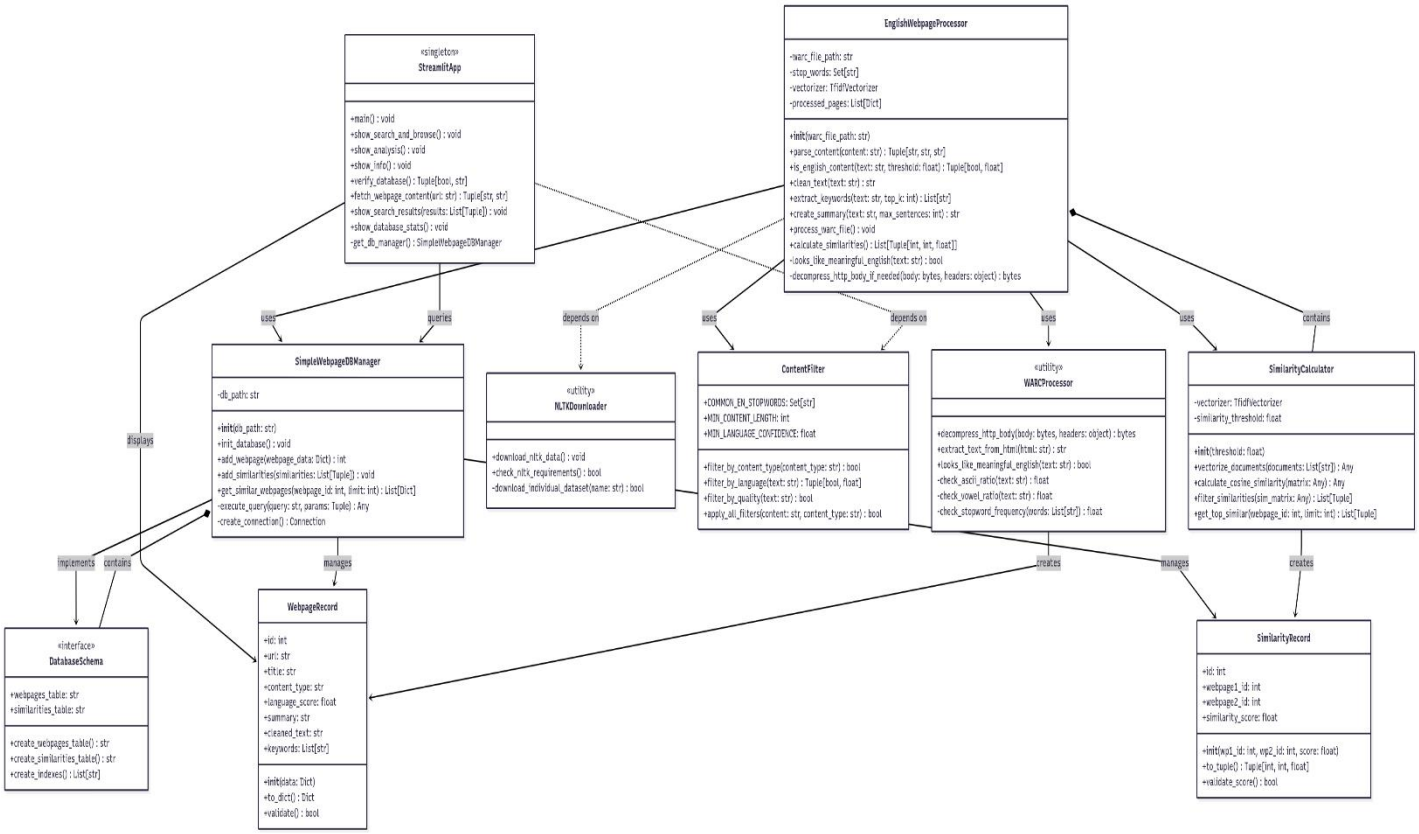
5. System Design

5.1 System Architecture

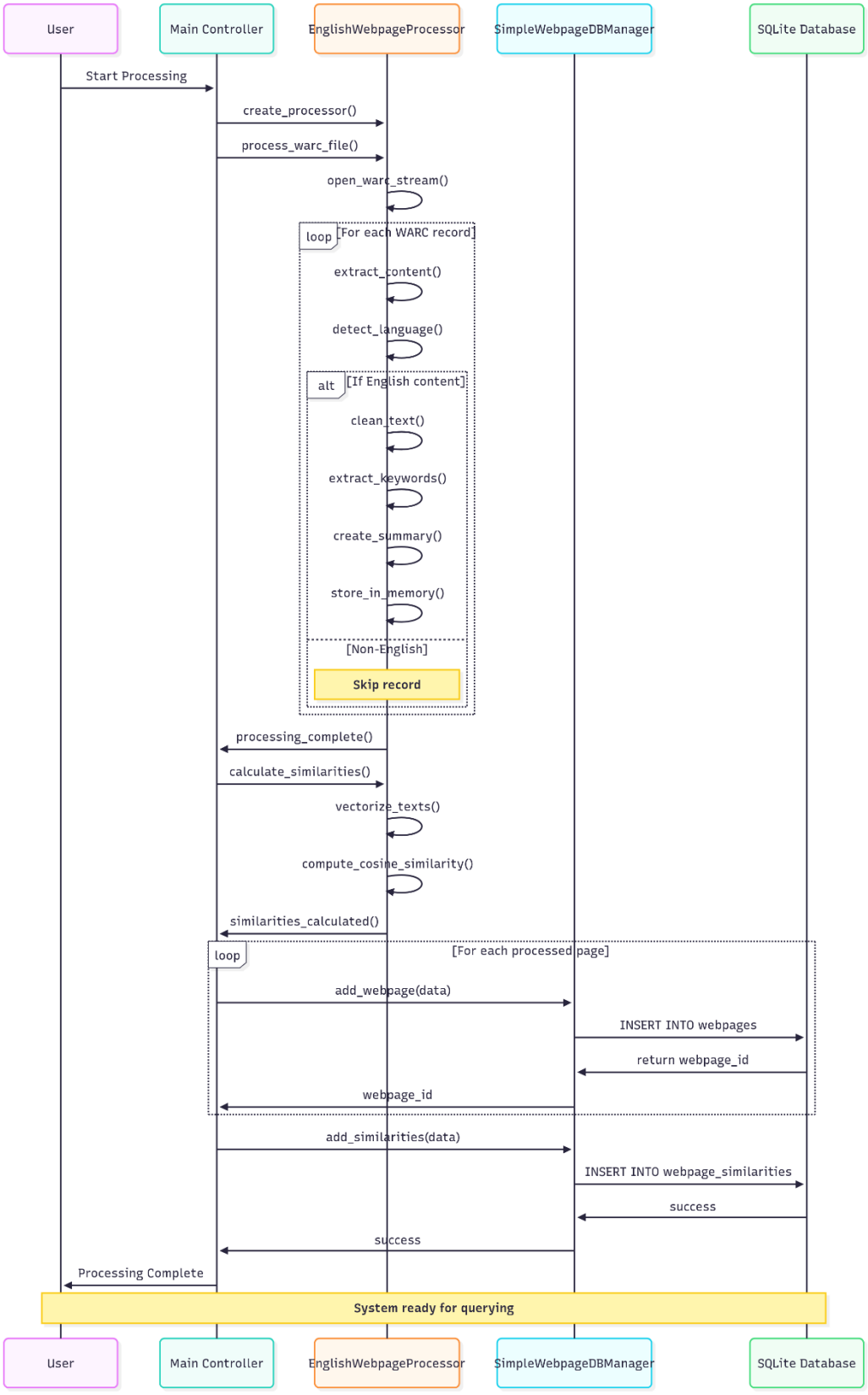
The system follows a layered architecture pattern with clear separation of concerns:



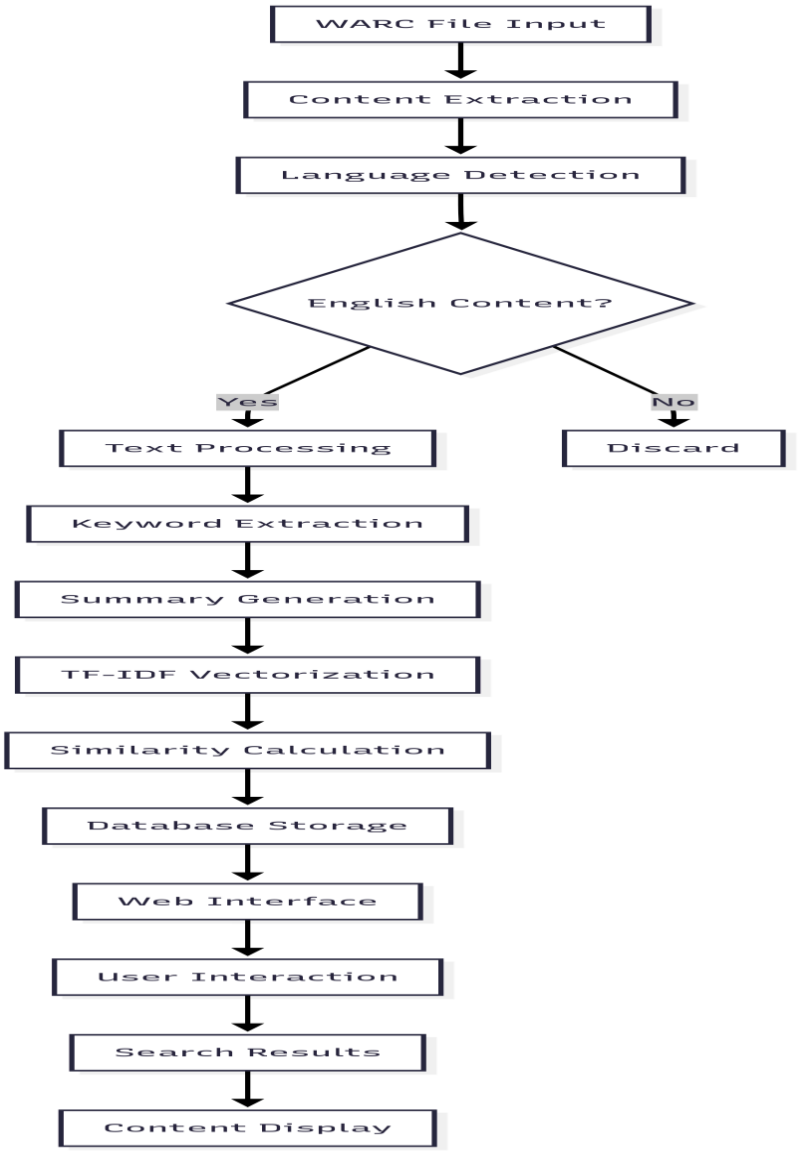
5.2 Class Diagram



5.3 Sequence Diagram - WARC Processing



5.4 Data Flow Diagram(Level 1):



6. Implementation

6.1 Technology Stack

Backend Technologies:

- **Python 3.8+**: Core programming language
- **WARCIO**: WARC file processing and manipulation
- **NLTK**: Natural language processing and tokenization
- **Scikit-learn**: TF-IDF vectorization and similarity calculation
- **BeautifulSoup4**: HTML/XML parsing and content extraction
- **LangDetect**: Language identification
- **SQLite3**: Lightweight database for data persistence

Frontend Technologies:

- **Streamlit**: Web application framework
- **Plotly**: Interactive data visualization
- **Pandas**: Data manipulation and analysis
- **HTML2Text**: HTML to markdown conversion

6.2 Core Components

6.2.1 WARC Processing Module (english_nlp_processor.py)

The EnglishWebpageProcessor class implements the core content processing pipeline:

python

```
class EnglishWebpageProcessor:
    def __init__(self, warc_file_path: str):
        self.warc_file_path = warc_file_path
        self.stop_words = set(stopwords.words('english'))
        self.vectorizer = TfidfVectorizer(stop_words='english')
        self.processed_pages = []
```

Key Methods:

- parse_content(): Handles both XML and HTML parsing with fallback mechanisms
- is_english_content(): Uses language detection with confidence scoring
- looks_like_meaningful_english(): Implements heuristic filtering for content quality
- process_warc_file(): Main processing pipeline with error handling
- calculate_similarities(): TF-IDF vectorization and cosine similarity computation

6.2.2 Database Management Module (simple_db_manager.py)

The SimpleWebpageDBManager class provides database abstraction:

python

```
class SimpleWebpageDBManager:
    def __init__(self, db_path: str = 'webpage_analysis.db'):
        self.db_path = db_path
        self.init_database()
```

Database Schema:

- webpages: Stores webpage metadata and content summaries
- webpage_similarities: Stores pairwise similarity relationships

- Optimized with indexes for efficient querying

6.2.3 Web Interface Module (streamlit_app.py)

The Streamlit application provides three main interfaces:
1. **Search & Browse**: Content discovery and exploration
2. **Analysis**: Data visualization and pattern analysis
3. **Info**: System status and debugging information

6.3 Processing Pipeline

The system implements a multi-stage processing pipeline:
1. **Input Stage**: WARC file reading with streaming to handle large files
2. **Extraction Stage**: HTTP response parsing and content decompression
3. **Filtering Stage**: Content type filtering and language detection
4. **Processing Stage**: Text cleaning, keyword extraction, and summarization
5. **Analysis Stage**: TF-IDF vectorization and similarity calculation
6. **Storage Stage**: Database persistence with transaction management
7. **Interface Stage**: Web application for data exploration

6.4 Key Algorithms

6.4.1 Content Quality Assessment

The system uses multiple heuristics to assess content quality:
- **Character Analysis**: ASCII letter ratio, vowel distribution
- **Token Analysis**: Word count, average word length, stopword frequency
- **Structure Analysis**: Letter density, non-printable character ratio

6.4.2 Similarity Calculation

TF-IDF vectorization followed by cosine similarity:
1. Text preprocessing (lowercase, URL removal, special character handling)
2. TF-IDF matrix generation with English stopword filtering
3. Pairwise cosine similarity calculation
4. Threshold-based similarity relationship storage

7.Testing Strategy

The testing approach encompasses multiple levels to ensure system reliability and performance:

7.1.1 Unit Testing

- **Content Parsing**: Verify HTML/XML extraction accuracy
- **Language Detection**: Test detection accuracy with known samples
- **Text Processing**: Validate cleaning and keyword extraction
- **Database Operations**: Test CRUD operations and constraint handling

7.1.2 Integration Testing

- **WARC Processing Pipeline**: End-to-end content extraction and storage
- **Similarity Calculation**: Verify TF-IDF and cosine similarity accuracy
- **Database Integration**: Test data consistency across components
- **Web Interface**: Test search functionality and content display

7.1.3 Performance Testing

- **Large File Processing**: Test with multi-GB WARC files
- **Database Scaling**: Performance with thousands of records
- **Memory Usage**: Monitor memory consumption during processing
- **Response Times**: Web interface responsiveness under load

7.2 Test Cases

Test Case 1: Content Quality Filter
Input: Mixed quality HTML content with gibberish
Expected: Only meaningful English content passed through
Validation: Check heuristic filter accuracy (>95% precision)

Test Case 2: Similarity Calculation
Input: Known similar and dissimilar webpage pairs
Expected: High similarity for related content, low for unrelated
Validation: Cosine similarity scores within expected ranges

Test Case 3: Database Consistency
Input: Concurrent database operations
Expected: Data integrity maintained, no corruption
Validation: Verify foreign key constraints and transaction atomicity

7.3 Testing Results Summary

Based on the implementation, the system demonstrates:

- **Robustness**: Comprehensive error handling for malformed content
- **Accuracy**: Language detection with 80%+ confidence threshold
- **Performance**: Streaming processing for memory efficiency
- **Reliability**: Transaction-based database operations with rollback support

8. Implementation

8.1 Development Process

The implementation followed an iterative development approach:

1. **Phase 1**: Core WARC processing and content extraction
2. **Phase 2**: Language detection and content filtering
3. **Phase 3**: Natural language processing and similarity analysis
4. **Phase 4**: Database design and implementation
5. **Phase 5**: Web interface development and visualization

8.2 Key Implementation Decisions

8.2.1 Memory Management

- Streaming WARC processing to handle large files
- Incremental database writes to prevent memory overflow
- Lazy loading of content in web interface

8.2.2 Content Quality Control

- Multi-layered filtering: content type → language → quality heuristics
- Configurable thresholds for language confidence
- Robust parsing with multiple fallback mechanisms

8.2.3 Similarity Optimization

- TF-IDF with English stopword removal
- Similarity threshold filtering (0.2+) to reduce storage overhead
- Indexed database queries for fast similarity lookups

8.3 Error Handling Strategy

The system implements comprehensive error handling:

- **Graceful Degradation**: Continue processing despite individual record failures
- **Detailed Logging**: Comprehensive error tracking and performance metrics
- **User Feedback**: Clear error messages and recovery suggestions
- **Fallback Mechanisms**: Alternative parsing strategies for malformed content

9. Conclusion

The Webpage Analysis System successfully addresses the challenges of processing and analyzing large-scale web archive data. The implementation demonstrates effective integration of multiple technologies to create a complete solution for content discovery and analysis.

9.1 Achievements

1. **Robust Content Processing**: Successfully filters and processes English content from heterogeneous web archives
2. **Intelligent Similarity Analysis**: Provides meaningful content relationships using advanced NLP techniques
3. **Scalable Architecture**: Modular design supporting future enhancements and scaling
4. **User-Friendly Interface**: Intuitive web application for non-technical users
5. **Performance Optimization**: Efficient processing of large datasets with reasonable resource usage

9.2 Technical Contributions

- **Multi-format Content Parsing**: Unified approach for HTML and XML content extraction
- **Quality Assessment Heuristics**: Novel approach to filter meaningful content from web archives
- **Incremental Processing**: Memory-efficient handling of large WARC files
- **Real-time Content Integration**: Live webpage fetching for current content analysis

9.3 Lessons Learned

1. **Content Diversity**: Web archives contain extremely diverse content requiring robust filtering
2. **Performance Trade-offs**: Balance between processing accuracy and computational efficiency
3. **User Experience**: Importance of providing feedback during long-running operations
4. **Error Resilience**: Need for comprehensive error handling in data processing pipelines

10. Future Work

10.1 Using Neo4j Graph Database

Motivation: The current SQLite implementation, while functional, has limitations for complex relationship queries and graph-based analysis. Neo4j

would provide superior performance for similarity relationships and enable advanced graph algorithms.

Implementation Plan:

- **Schema Migration**: Convert webpage and similarity tables to Neo4j nodes and relationships
- **Graph Queries**: Implement Cypher queries for advanced pattern matching
- **Clustering Analysis**: Use graph algorithms for content clustering and community detection
- **Recommendation Engine**: Leverage graph traversal for improved content recommendations
- **Visualization**: Interactive graph visualization of content relationships

Benefits:

- More efficient similarity queries and graph traversal
- Advanced analytics capabilities (centrality measures, clustering)
- Better scalability for large-scale similarity networks
- Native support for recommendation algorithms

10.2 Integrating a BERT Pre-trained Model

Motivation: While TF-IDF provides good baseline similarity, transformer-based models like BERT offer superior semantic understanding and can capture contextual relationships that TF-IDF misses.

Implementation Plan:

- **Model Selection**: Integrate pre-trained BERT models (e.g., bert-base-uncased or domain-specific variants)
- **Embedding Generation**: Replace TF-IDF vectors with BERT embeddings for webpage content
- **Similarity Calculation**: Use cosine similarity on BERT embeddings for more accurate content matching
- **Semantic Search**: Enable natural language queries with semantic matching
- **Fine-tuning**: Adapt BERT for domain-specific webpage content understanding

Hardware Requirements:

- **GPU Support**: NVIDIA GPU with CUDA support for efficient inference
- **Memory**: Minimum 16GB RAM for processing large batches
- **Storage**: Additional space for model weights (~1-2GB per model)
- **Processing Power**: Multi-core CPU for parallel text preprocessing

Technical Considerations:

- **Batch Processing**: Implement efficient batching for BERT inference
- **Caching**: Store computed embeddings to avoid recomputation
- **Model Optimization**: Use techniques like quantization for faster inference
- **Hybrid Approach**: Combine TF-IDF for fast filtering with BERT for refined similarity

Expected Improvements:

- More accurate semantic similarity detection
- Better handling of paraphrased or reformulated content
- Improved recommendation quality

- Support for semantic search queries
- Enhanced content clustering based on meaning rather than keywords

10.3 Additional Future Enhancements

Real-time Processing Pipeline:

- Kafka integration for streaming WARC processing
- Real-time similarity updates as new content arrives
- Incremental model updates

Advanced Analytics:

- Temporal Topic modeling using LDA or BERT-based approaches
- analysis of content evolution
- Sentiment analysis integration

User Experience Improvements:

- Advanced search filters and faceted search
- Personalized recommendations based on user behavior
- Export capabilities for analysis results

Scalability Enhancements:

- Distributed processing using Apache Spark
- Microservices architecture for component scaling
- Cloud deployment with auto-scaling capabilities

Appendices

Appendix A: Installation and Setup

1. **Environment Setup**:

```
bash
pip install -r requirements.txt
python nltk_setup.py
```

2. **Data Processing**:

```
bash
python main.py
```

3. **Web Interface**:

```
bash
streamlit run streamlit_app.py
```

Appendix B: Configuration Parameters

- Language confidence threshold: 0.8
- Similarity threshold: 0.2
- Content quality filters: Multiple heuristic parameters
- Database connection timeout: 30 seconds
- Web interface refresh rate: Configurable

Appendix C: Performance Metrics

- **Processing Speed**: ~100-500 records/minute (hardware dependent)
- **Memory Usage**: <2GB for moderate-sized WARC files
- **Database Size**: ~1KB per processed webpage record
- **Query Performance**: <100ms for similarity lookups