# NLP Pipeline - Complete Setup & Testing Guide

**Status:** Week 3-6 Implementation  
**Date:** October 2025  
**Version:** 1.0

## 📋 Table of Contents

1. [Overview](#overview)
2. [System Architecture](#system-architecture)
3. [Installation & Setup](#installation--setup)
4. [Testing the Pipeline](#testing-the-pipeline)
5. [Integration with Annotations](#integration-with-annotations)
6. [Performance Evaluation](#performance-evaluation)
7. [Troubleshooting](#troubleshooting)
8. [Next Steps](#next-steps)

## Overview

### What You’re Building

A complete NLP pipeline that: - **Processes** raw news articles - **Extracts** violent events with 5W1H attributes - **Formats** output matching your annotation template - **Enables** training of ML models on annotated data

### Components Delivered

✅ **Text Preprocessing** - HTML/boilerplate removal - Text normalization - Sentence splitting

✅ **Stanford CoreNLP Integration** - Tokenization, POS tagging - NER, dependency parsing - Coreference resolution

✅ **Feature Extraction** - Lexical features - Syntactic features - Domain-specific features

✅ **Event Detection** - Trigger identification - 5W1H extraction - Confidence scoring

✅ **Batch Processing** - Multiple article processing - Output formatting - Progress tracking

## 

## System Architecture

┌─────────────────────┐  
│ Raw News Articles │  
└──────────┬──────────┘  
 │  
 ▼  
┌─────────────────────┐  
│ Text Preprocessing │  
│ - Clean HTML │  
│ - Split sentences │  
└──────────┬──────────┘  
 │  
 ▼  
┌─────────────────────┐  
│ Stanford CoreNLP │  
│ - Tokenize │  
│ - POS tag │  
│ - Parse │  
│ - NER │  
└──────────┬──────────┘  
 │  
 ▼  
┌─────────────────────┐  
│ Event Detection │  
│ - Find triggers │  
│ - Extract 5W1H │  
└──────────┬──────────┘  
 │  
 ▼  
┌─────────────────────┐  
│ Output Formatting │  
│ - Match template │  
│ - Excel/CSV output │  
└──────────┬──────────┘  
 │  
 ▼  
┌─────────────────────┐  
│ Training Data │  
│ (for ML models) │  
└─────────────────────┘

## Installation & Setup

### Step 1: System Requirements

**Minimum:** - Python 3.8+ - 8GB RAM - 5GB disk space

**Recommended:** - Python 3.9+ - 16GB RAM - 10GB disk space - SSD for faster I/O

### Step 2: Install Dependencies

Create project structure:

mkdir nlp\_pipeline  
cd nlp\_pipeline  
  
# Create directory structure  
mkdir -p preprocessing  
mkdir -p stanford\_nlp  
mkdir -p features  
mkdir -p domain\_specific  
mkdir -p output  
mkdir -p logs  
mkdir -p resources  
  
# Create \_\_init\_\_.py files  
touch preprocessing/\_\_init\_\_.py  
touch stanford\_nlp/\_\_init\_\_.py  
touch features/\_\_init\_\_.py  
touch domain\_specific/\_\_init\_\_.py

Install Python packages:

# Core packages  
pip install pandas numpy scikit-learn  
  
# NLP packages  
pip install stanfordcorenlp  
  
# Visualization (optional)  
pip install matplotlib seaborn  
  
# Web/API (if needed later)  
pip install flask requests

Or use requirements.txt:

# Create requirements.txt  
cat > requirements.txt << EOF  
pandas>=1.3.0  
numpy>=1.21.0  
scikit-learn>=0.24.0  
stanfordcorenlp>=3.9.1  
matplotlib>=3.4.0  
seaborn>=0.11.0  
openpyxl>=3.0.0  
EOF  
  
# Install  
pip install -r requirements.txt

### Step 3: Download Stanford CoreNLP

# Download (4.5.5 is current as of 2024)  
wget http://nlp.stanford.edu/software/stanford-corenlp-4.5.5.zip  
  
# Unzip  
unzip stanford-corenlp-4.5.5.zip  
  
# Verify  
ls stanford-corenlp-4.5.5/  
# Should see: stanford-corenlp-4.5.5.jar and other files

### Step 4: Organize Code Files

Place the code from the artifacts:

nlp\_pipeline/  
├── config.yaml # Configuration  
├── preprocessing/  
│ ├── \_\_init\_\_.py  
│ ├── text\_cleaner.py # From draft.txt  
│ └── sentence\_splitter.py # From draft.txt  
├── stanford\_nlp/  
│ ├── \_\_init\_\_.py  
│ └── corenlp\_wrapper.py # From draft.txt  
├── features/  
│ ├── \_\_init\_\_.py  
│ ├── lexical\_features.py # From draft.txt  
│ └── syntactic\_features.py # From draft.txt  
├── domain\_specific/  
│ ├── \_\_init\_\_.py  
│ ├── violence\_lexicon.py # From draft.txt  
│ └── african\_ner.py # From draft.txt  
├── event\_extraction.py # New: Event detection module  
├── batch\_processing.py # New: Batch processor  
├── pipeline.py # From draft.txt (main orchestrator)  
├── test\_pipeline.py # Test script  
└── stanford-corenlp-4.5.5/ # CoreNLP directory

### Step 5: Create Configuration

Create config.yaml:

# Stanford CoreNLP Settings  
stanford\_corenlp:  
 path: "./stanford-corenlp-4.5.5"  
 memory: "4g"  
 timeout: 30000  
 annotators: "tokenize,ssplit,pos,lemma,ner,parse,depparse,coref"  
 language: "en"  
  
# Feature Extraction  
features:  
 use\_word\_embeddings: false # Set true if using embeddings  
 embedding\_dim: 300  
 context\_window: 5  
  
# Domain Resources  
domain:  
 violence\_lexicon\_path: "./resources/violence\_lexicon.txt"  
 actor\_database\_path: "./resources/actors.json"  
 location\_database\_path: "./resources/african\_locations.json"  
  
# Processing  
processing:  
 batch\_size: 10  
 max\_sentence\_length: 512  
  
# Output  
output:  
 directory: "./output"  
 format: "xlsx"  
  
# Logging  
logging:  
 level: "INFO"  
 file: "./logs/nlp\_pipeline.log"

## Testing the Pipeline

### Test 1: Basic Components (5 minutes)

Test individual components:

# test\_components.py  
  
def test\_text\_cleaner():  
 """Test text cleaning."""  
 from preprocessing.text\_cleaner import TextCleaner  
   
 cleaner = TextCleaner()  
   
 raw\_text = """  
 <html><body>  
 <h1>Breaking News</h1>  
 <p>Armed militants attacked a village in Mali.</p>  
 Advertisement  
 </body></html>  
 """  
   
 cleaned = cleaner.clean(raw\_text)  
 print("Cleaned text:", cleaned)  
 assert "Armed militants" in cleaned  
 assert "<html>" not in cleaned  
 print("✓ Text cleaner works!")  
  
def test\_violence\_lexicon():  
 """Test violence lexicon."""  
 from domain\_specific.violence\_lexicon import ViolenceLexicon  
   
 lexicon = ViolenceLexicon()  
   
 assert lexicon.is\_violence\_term('killed')  
 assert lexicon.is\_violence\_term('attack')  
 assert not lexicon.is\_violence\_term('happy')  
   
 print(f"✓ Violence lexicon loaded: {len(lexicon.all\_terms)} terms")  
  
def test\_african\_ner():  
 """Test African NER."""  
 from domain\_specific.african\_ner import AfricanNER  
   
 ner = AfricanNER()  
   
 text = "Boko Haram attacked a village in Maiduguri, Nigeria"  
   
 actors = ner.recognize\_actor(text)  
 locations = ner.recognize\_location(text)  
   
 print(f"Actors found: {actors}")  
 print(f"Locations found: {locations}")  
   
 assert len(actors) > 0  
 assert len(locations) > 0  
 print("✓ African NER works!")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_text\_cleaner()  
 test\_violence\_lexicon()  
 test\_african\_ner()  
 print("\n✅ All component tests passed!")

Run:

python test\_components.py

### Test 2: CoreNLP Integration (10 minutes)

Test Stanford CoreNLP:

# test\_corenlp.py  
  
from stanford\_nlp.corenlp\_wrapper import CoreNLPWrapper  
  
def test\_corenlp():  
 """Test CoreNLP wrapper."""  
   
 print("Initializing CoreNLP (may take 30 seconds)...")  
 nlp = CoreNLPWrapper('./stanford-corenlp-4.5.5')  
   
 text = "Armed militants killed 15 civilians in Maiduguri on Tuesday."  
   
 print("\nAnnotating text...")  
 annotation = nlp.annotate(text)  
   
 if 'sentences' in annotation:  
 sent = annotation['sentences'][0]  
   
 # Test token extraction  
 tokens = nlp.get\_tokens(sent)  
 print(f"✓ Tokens ({len(tokens)}): {[t['word'] for t in tokens[:5]]}...")  
   
 # Test entity extraction  
 entities = nlp.get\_entities(sent)  
 print(f"✓ Entities: {entities}")  
   
 # Test dependencies  
 deps = nlp.get\_dependencies(sent)  
 print(f"✓ Dependencies ({len(deps)}): {deps[:3]}...")  
   
 else:  
 print("❌ CoreNLP annotation failed")  
 return False  
   
 nlp.close()  
 print("\n✅ CoreNLP test passed!")  
 return True  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_corenlp()

Run:

python test\_corenlp.py

**Expected output:**

Initializing CoreNLP (may take 30 seconds)...  
Annotating text...  
✓ Tokens (7): ['Armed', 'militants', 'killed', '15', 'civilians']...  
✓ Entities: [{'type': 'NUMBER', 'text': '15'}, {'type': 'LOCATION', 'text': 'Maiduguri'}]  
✓ Dependencies (15): [{'dep': 'amod', 'governor': 2, 'dependent': 1}, ...]  
✅ CoreNLP test passed!

### Test 3: Event Extraction (15 minutes)

Test complete extraction:

# test\_extraction.py  
  
from pipeline import ViolentEventNLPPipeline  
from event\_extraction import EventExtractor  
from domain\_specific.violence\_lexicon import ViolenceLexicon  
from domain\_specific.african\_ner import AfricanNER  
  
def test\_event\_extraction():  
 """Test full event extraction pipeline."""  
   
 # Sample article  
 article\_text = """  
 Maiduguri, Nigeria - At least 15 people were killed when suspected   
 Boko Haram militants attacked a village in northeastern Nigeria on   
 Tuesday. The attackers arrived at dawn and opened fire on residents.   
 Witnesses said the gunmen burned several houses before fleeing.  
 """  
   
 # Initialize pipeline  
 config = {  
 'stanford\_corenlp': {  
 'path': './stanford-corenlp-4.5.5',  
 'memory': '4g'  
 }  
 }  
   
 print("Initializing pipeline...")  
 pipeline = ViolentEventNLPPipeline(config)  
   
 # Initialize event extractor  
 print("Initializing event extractor...")  
 lexicon = ViolenceLexicon()  
 ner = AfricanNER()  
 extractor = EventExtractor(lexicon, ner)  
   
 # Process article  
 print("\nProcessing article...")  
 nlp\_result = pipeline.process\_article(article\_text, 'TEST\_001')  
   
 print(f"✓ Processed {nlp\_result['num\_sentences']} sentences")  
   
 # Extract events  
 print("\nExtracting events...")  
 events = extractor.extract\_events(nlp\_result)  
   
 print(f"✓ Extracted {len(events)} events")  
   
 # Display first event  
 if events:  
 event = events[0]  
 print("\nFirst Event:")  
 print(f" Trigger: {event['trigger']['word']}")  
 print(f" Who: {event['who']}")  
 print(f" Whom: {event['whom']}")  
 print(f" Where: {event['where']}")  
 print(f" When: {event['when']}")  
 print(f" How: {event['how']}")  
 print(f" Confidence: {event['confidence']}")  
   
 pipeline.close()  
 print("\n✅ Event extraction test passed!")  
 return len(events) > 0  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 success = test\_event\_extraction()  
 if not success:  
 print("❌ No events extracted - check configuration")

Run:

python test\_extraction.py

### Test 4: Batch Processing (20 minutes)

Create test articles:

// test\_articles.json  
[  
 {  
 "id": "ART\_001",  
 "text": "Armed militants killed 15 civilians in Maiduguri, Nigeria on Tuesday.",  
 "url": "https://example.com/1",  
 "source": "Reuters",  
 "publication\_date": "2024-03-20"  
 },  
 {  
 "id": "ART\_002",   
 "text": "Al-Shabaab attacked a military base in Mogadishu, killing 8 soldiers.",  
 "url": "https://example.com/2",  
 "source": "AFP",  
 "publication\_date": "2024-03-21"  
 },  
 {  
 "id": "ART\_003",  
 "text": "Police shot protesters in Harare, Zimbabwe, killing at least 6 people.",  
 "url": "https://example.com/3",  
 "source": "BBC",  
 "publication\_date": "2024-03-22"  
 }  
]

Test batch processing:

# test\_batch.py  
  
from batch\_processing import IntegratedPipeline  
  
def test\_batch\_processing():  
 """Test batch article processing."""  
   
 config = {  
 'stanford\_corenlp': {  
 'path': './stanford-corenlp-4.5.5',  
 'memory': '4g'  
 },  
 'output': {  
 'directory': './output'  
 }  
 }  
   
 print("Initializing integrated pipeline...")  
 pipeline = IntegratedPipeline(config)  
   
 print("\nProcessing batch of articles...")  
 results = pipeline.process\_from\_file('test\_articles.json', file\_type='json')  
   
 print("\n" + "="\*70)  
 print("BATCH PROCESSING RESULTS")  
 print("="\*70)  
 print(f"Total articles: {results['total\_articles']}")  
 print(f"Processed: {results['processed']}")  
 print(f"Failed: {results['failed']}")  
 print(f"Total events: {results['total\_events']}")  
 print(f"Output: {results.get('output\_file', 'N/A')}")  
 print("="\*70)  
   
 pipeline.close()  
   
 # Check output file exists  
 if results.get('output\_file'):  
 import os  
 assert os.path.exists(results['output\_file'])  
 print(f"\n✓ Output file created: {results['output\_file']}")  
   
 print("\n✅ Batch processing test passed!")  
 return True  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_batch\_processing()

Run:

python test\_batch.py

## Integration with Annotations

### Goal

Compare NLP pipeline output with human annotations to: 1. Evaluate extraction accuracy 2. Identify where pipeline fails 3. Create features for ML training

### Step 1: Prepare Comparison Dataset

Take 50 articles that annotators have already completed:

# compare\_with\_annotations.py  
  
import pandas as pd  
  
def compare\_pipeline\_with\_annotations():  
 """  
 Compare automated extraction with human annotations.  
 """  
   
 # Load human annotations  
 human\_annotations = pd.read\_excel('human\_annotations\_sample.xlsx',   
 sheet\_name='Event Records')  
   
 # Load NLP pipeline output  
 pipeline\_output = pd.read\_excel('output/batch\_test\_events.xlsx',  
 sheet\_name='Event Records')  
   
 # Compare for same articles  
 article\_ids = set(human\_annotations['Article\_ID'])  
   
 comparison = []  
   
 for article\_id in article\_ids:  
 human\_events = human\_annotations[  
 human\_annotations['Article\_ID'] == article\_id  
 ]  
 pipeline\_events = pipeline\_output[  
 pipeline\_output['Article\_ID'] == article\_id  
 ]  
   
 comparison.append({  
 'article\_id': article\_id,  
 'human\_count': len(human\_events),  
 'pipeline\_count': len(pipeline\_events),  
 'match': len(human\_events) == len(pipeline\_events)  
 })  
   
 comp\_df = pd.DataFrame(comparison)  
   
 print("\nComparison Results:")  
 print(f"Total articles: {len(comp\_df)}")  
 print(f"Exact matches: {comp\_df['match'].sum()}")  
 print(f"Average human events: {comp\_df['human\_count'].mean():.2f}")  
 print(f"Average pipeline events: {comp\_df['pipeline\_count'].mean():.2f}")  
   
 return comp\_df  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 results = compare\_pipeline\_with\_annotations()  
 results.to\_excel('comparison\_results.xlsx', index=False)

### Step 2: Detailed Attribute Comparison

def compare\_attributes(human\_row, pipeline\_row):  
 """Compare specific attributes."""  
   
 comparison = {}  
   
 # Actor match  
 human\_actor = str(human\_row['Actor\_Normalized']).lower()  
 pipeline\_actor = str(pipeline\_row['Actor\_Normalized']).lower()  
 comparison['actor\_match'] = human\_actor == pipeline\_actor  
   
 # Victim match  
 human\_victim = str(human\_row['Victim\_Normalized']).lower()  
 pipeline\_victim = str(pipeline\_row['Victim\_Normalized']).lower()  
 comparison['victim\_match'] = human\_victim == pipeline\_victim  
   
 # Location match  
 human\_loc = str(human\_row['Location\_Country']).lower()  
 pipeline\_loc = str(pipeline\_row['Location\_Country']).lower()  
 comparison['location\_match'] = human\_loc == pipeline\_loc  
   
 # Casualty match  
 comparison['deaths\_match'] = (  
 human\_row['Deaths'] == pipeline\_row['Deaths']  
 )  
   
 return comparison

## Performance Evaluation

### Metrics to Track

1. **Event Detection**
   * Precision: Of events found, how many are real?
   * Recall: Of real events, how many were found?
   * F1-score: Harmonic mean
2. **5W1H Extraction** (for each attribute)
   * Exact match %
   * Partial match %
   * Missing %
3. **Processing Speed**
   * Articles per hour
   * Seconds per article
   * Memory usage

### Evaluation Script

# evaluate\_pipeline.py  
  
from sklearn.metrics import precision\_recall\_fscore\_support  
import pandas as pd  
import time  
  
def evaluate\_extraction\_accuracy(human\_file, pipeline\_file):  
 """  
 Calculate extraction metrics.  
 """  
 human\_df = pd.read\_excel(human\_file, sheet\_name='Event Records')  
 pipeline\_df = pd.read\_excel(pipeline\_file, sheet\_name='Event Records')  
   
 results = {  
 'event\_detection': {},  
 'attribute\_extraction': {},  
 'processing\_speed': {}  
 }  
   
 # Event detection metrics  
 # (This is simplified - real implementation needs event matching)  
 human\_counts = human\_df.groupby('Article\_ID').size()  
 pipeline\_counts = pipeline\_df.groupby('Article\_ID').size()  
   
 results['event\_detection'] = {  
 'human\_total': len(human\_df),  
 'pipeline\_total': len(pipeline\_df),  
 'avg\_per\_article\_human': human\_counts.mean(),  
 'avg\_per\_article\_pipeline': pipeline\_counts.mean()  
 }  
   
 return results  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 results = evaluate\_extraction\_accuracy(  
 'human\_annotations\_sample.xlsx',  
 'output/batch\_test\_events.xlsx'  
 )  
   
 print("\nEvaluation Results:")  
 print(json.dumps(results, indent=2))

## Troubleshooting

### Common Issues

#### 1. CoreNLP Won’t Start

**Error:** Connection refused or timeout

**Solutions:**

# Check Java version (need 8+)  
java -version  
  
# Try different port  
# In code: StanfordCoreNLP(..., port=9001)  
  
# Increase memory  
# In config: memory: "6g"  
  
# Check if port is in use  
netstat -an | grep 9000

#### 2. Out of Memory

**Error:** Java heap space

**Solutions:** - Increase CoreNLP memory: memory: "6g" or "8g" - Process smaller batches - Close and restart pipeline between batches

#### 3. Slow Processing

**Issue:** Taking >2 minutes per article

**Solutions:** - Use fewer annotators in config - Disable coreference if not needed - Process in parallel (if multiple CPUs) - Consider GPU acceleration for future ML

#### 4. Extraction Misses Events

**Issue:** Not finding obvious events

**Solutions:** - Check violence lexicon is loaded - Verify trigger detection thresholds - Add domain-specific patterns - Review sentence splitting (may split events)

## Next Steps

### Week 4: Refinement (Current Week)

✅ **This Week:** 1. Run full test suite 2. Process 100 test articles 3. Compare with human annotations 4. Identify common extraction errors 5. Refine extraction rules

### Week 5-6: Feature Engineering

**Prepare for ML Phase:** 1. Create feature vectors from NLP output 2. Design features for classification: - Lexical features - Syntactic features - Entity features - Context features

1. Save features alongside annotations
2. Validate feature quality

### Week 7-8: Database Design (Parallel)

While NLP pipeline runs, design: - PostgreSQL schema for events - Neo4j graph schema for relationships - API specifications - Sample queries

### Week 9-12: Machine Learning

**With complete training data:** 1. Train hierarchical classifiers 2. Train entity extraction models 3. Optimize hyperparameters 4. Evaluate on test set

## Quick Reference Commands

### Daily Testing

# Test components  
python test\_components.py  
  
# Test extraction on one article  
python test\_extraction.py  
  
# Process small batch  
python batch\_processing.py --input test\_articles.json --output ./output

### Production Processing

# Process full article set  
python batch\_processing.py \  
 --input articles\_batch1.json \  
 --type json \  
 --output ./output \  
 --config config.yaml

### Evaluation

# Compare with annotations  
python compare\_with\_annotations.py  
  
# Generate evaluation report  
python evaluate\_pipeline.py

## Success Criteria

By end of Week 6, you should have:

✅ NLP pipeline processing articles reliably  
✅ Event extraction working (even if not perfect)  
✅ Output format matching annotation template  
✅ Comparison with human annotations complete  
✅ Documentation of strengths/weaknesses  
✅ Ready to start ML model training (Week 9)

**Current Priority:** Get the pipeline working end-to-end, even if extraction isn’t perfect. ML models will improve accuracy!

## Support

**Issues?** Document them: 1. Error message 2. Input that caused it 3. Expected vs actual output 4. System info (Python version, RAM, etc.)

**Questions about ML phase?** We’ll cover that in Weeks 9-12!

**Ready to start?** Begin with Test 1 (Basic Components) 🚀

**Document Status:** Ready for Implementation  
**Last Updated:** October 2025  
**Version:** 1.0