AI Agent Execution Guide

Functional Requirements & Technical Specifications

EXECUTIVE BRIEFING FOR AI AGENTS

You are assigned to execute the **Influence Campaign Detection project**—an advanced machine learning initiative combining NLP and OSINT expertise to detect coordinated disinformation campaigns through intelligent document clustering. Your mission is to build a production-ready system that outperforms document-level classification by 18%+ using belief span extraction, multi-algorithm clustering, and ensemble classification.

Core Innovation: Instead of classifying documents individually, cluster document *parts* expressing coordinated beliefs, then project those clusters back to document level for holistic campaign detection.

SECTION 1: MISSION & OBJECTIVES

1.1 Mission Statement

Develop an end-to-end machine learning pipeline to detect and characterize influence campaigns from social media and news documents by:

1. Extracting belief-centric spans from documents
2. Embedding spans using semantic models
3. Clustering spans across 135 algorithm variations
4. Classifying clusters as "high-influence"
5. Projecting clusters to document-level predictions
6. Characterizing campaign narratives

Expected Outcome: Achieve ≥75% F1 on document classification while identifying which document parts drive campaign membership.

1.2 Strategic Context

- **Why This Matters**: Election interference, state propaganda, and coordinated misinformation require campaign-level detection, not individual document classification
- **Competitive Advantage**: 18%+ F1 improvement over document-level approaches (77.8% vs 50.7%)
- **Domain Challenge**: Class imbalance (7.8% positive), heterogeneous document lengths (26-945 tokens), multilingual content

- **OSINT Integration**: Your expertise in detecting misinformation patterns aligns perfectly with campaign-level clustering

1.3 Success Criteria

Metric	Target	Rationale
Document F1	≥75%	Achievable with aggregated clustering (actual: 77.8%)
Cluster Precision	≥80%	Minimize false positive campaigns
Interpretability	95%+	Justify all predictions with cluster membership
Generalization	≥70% on new campaigns	Test on HQP, MuMiN datasets
Production Latency	<2sec/1K docs	Enable real-time platform monitoring

SECTION 2: FUNCTIONAL REQUIREMENTS

MODULE 1: Data Ingestion & Preprocessing

Input Specifications

```
{
  "doc_id": "string (unique identifier)",
  "text": "string (document content, 26-945 tokens)",
  "media_type": "enum: Twitter, News, Blog, Forum, Reddit, Other",
  "language": "string (ISO 639-1 code: en, fr)",
  "timestamp": "string (ISO 8601 format, optional)",
  "label": "int (0=control, 1=campaign, optional for training)"
}
```

Dataset Characteristics:

- Size: 5,334 training docs, 1,333 test docs
- Class balance: 7.8% positive (highly imbalanced)
- Languages: French (primary), English (secondary)
- Media types: 6 types with varied document lengths

Preprocessing Pipeline

Step 1: Language Detection & Translation

```
def preprocess_documents(docs):
    """
    INPUT: Raw documents with mixed languages
    PROCESS:
    1. Detect language using fasttext model
    2. If non-English/French: translate to English (optional)
```

```
    3. If French: keep original but note language
    OUTPUT: Normalized documents with language tags
    """
```

Step 2: Sentence Segmentation

```
def segment_sentences(text):
    """
    INPUT: Document text (raw string)
    REQUIREMENTS:
    - Use spaCy v3.5.3 sentence segmentation
    - Handle edge cases: URLs, abbreviated words, contractions
    - Preserve token positions for linking back to original text
    OUTPUT: List of sentences with start/end character positions

    EXAMPLE:
    Input: "Putin cleans up the bioweapons labs installed by the deep state..."
    Output: [
      Sentence(text="Putin cleans up the bioweapons labs...", start=0, end=80)
    ]
    """
```

Output Specifications

```
Preprocessed Dataset:
├── sentences.jsonl
│   └── {"doc_id": "123", "sent_id": "123_0", "text": "...", "tokens": 25}
├── targets.jsonl
│   └── {"doc_id": "123", "target_id": "123_0_0", "span": "bioweapons labs",
│           "factuality": "committed_belief", "source": "author"}
└── metadata.json
    └── {"train_size": 5334, "test_size": 1333, "total_tokens": 3500000}
```

MODULE 2: Document Part Extraction

Requirement: Three Extraction Methods

METHOD A: Sentence-Level Extraction

```
def extract_sentences(document):
    """
    REQUIREMENT: Extract ALL sentences from document
    INPUT: Preprocessed document with sentence boundaries
    PROCESS:
    1. Segment into sentences
    2. Tokenize each sentence
    3. Filter out sentences &lt; 3 tokens (noise)
    OUTPUT: List of sentence strings

    TARGET OUTPUTS:
```

```
    - Training: 72,330 sentences
    - Test: 14,370 sentences
    """
```

METHOD B: Target Span Extraction (All)

```
def extract_all_target_spans(document):
    """
    REQUIREMENT: Use existing event factuality prediction system
    INPUT: Document text
    PROCESS:
    1. Load pre-trained event factuality system (Murzaku et al. 2023)
    2. For each sentence, predict (source, target, factuality)
    3. Extract multi-word span using head-to-span algorithm
    4. Retain spans with factuality labels
    OUTPUT: List of (span_text, factuality_label, source_type) tuples

    FACTUALITY LABELS:
    - committed_belief: "The author believes X is true"
    - possible_belief: "X might be true"
    - unknown_belief: "Uncertain about X"
    - possible_disbelief: "X might be false"
    - committed_disbelief: "X is certainly false"

    TARGET OUTPUTS:
    - Training: 270,818 targets
    - Test: 50,781 targets
    """
```

METHOD C: Author-Attribution Target Extraction

```
def extract_author_belief_targets(document):
    """
    REQUIREMENT: Extract ONLY spans where author expresses belief
    INPUT: Document text with factuality predictions
    PROCESS:
    1. Use same event factuality system as METHOD B
    2. Filter: KEEP ONLY spans where source="author"
    3. KEEP ONLY factuality ∈ {committed_belief, possible_belief}
    4. Discard attributed beliefs (e.g., "Putin said X")
    OUTPUT: List of author-attributed belief spans

    TARGET OUTPUTS:
    - Training: 155,238 targets
    - Test: 29,793 targets
    """
```

Feature Extraction (95 Linguistic Features)

```
def extract_linguistic_features(document):
    """
    REQUIREMENT: Extract 95 non-lexical linguistic features

    FEATURE CATEGORIES:

    1. STRUCTURAL FEATURES (8 features)
    - Average word length
    - Type-token ratio (vocabulary diversity)
    - Mean sentence length
    - Sentence complexity (clauses per sentence)
    - Passive voice ratio
    - Word repetition index

    2. CONVERSATIONAL FEATURES (12 features)
    - Contraction frequency (I'm, don't, etc.)
    - First/Second/Third person pronouns
    - Pronoun-to-noun ratio
    - Hedging words (maybe, perhaps, might)
    - Emphatic markers (absolutely, definitely)
    - Modal verbs (can, should, would)
    - Questions and exclamations per 100 words

    3. SENTENTIAL FEATURES (35 features)
    - Subordination/Coordination ratio
    - WH-questions ratio
    - Tense and aspect distributions
    - Negation frequency
    - Comparative/Superlative structures
    [... full 35 features from Biber framework]

    4. LEXICAL/POS FEATURES (40 features)
    - Noun/Verb/Adjective types and distributions
    - Entity density (named entities per 100 words)
    [... full 40 POS-based features]

    OUTPUT: 95-dimensional feature vector (normalized 0-1)

    REQUIREMENT: NO lexical features (word presence/TF-IDF)
    RATIONALE: Prevent overfitting to specific campaign datasets
    """
```

MODULE 3: Semantic Embedding Generation

S-BERT Embedder Specification

```
class SBERTEmbedder:
    """
    REQUIREMENT: Generate 768-dimensional semantic embeddings
    """

    def __init__(self):
        """
        SETUP REQUIREMENTS:
        - Model: "all-mpnet-base-v2" (SOTA on benchmark)
        - Load from Hugging Face transformers
        - GPU-accelerated inference (if available)
        - Batch size: 256 for efficiency

        SPECIFICATION:
        - Output dimensions: 768
        - Pooling: Mean pooling over token embeddings
        - Normalization: L2 normalization to unit length
        """

    def embed_batch(self, texts: List[str]) -&gt; np.ndarray:
        """
        INPUT: List of text strings (document parts)
        PROCESS:
        1. Tokenize with max_length=512 (BERT limit)
        2. Generate embeddings via forward pass
        3. Apply mean pooling across tokens
        4. L2 normalize each embedding
        OUTPUT: (N, 768) numpy array

        PERFORMANCE TARGET:
        - Throughput: ≥5K embeddings/minute on GPU
        - Memory: &lt;8GB for 100K embeddings
        """

# EXECUTION REQUIREMENTS:
TOTAL_EMBEDDINGS_TO_GENERATE = {
    "Sentences": 86_700,
    "All Targets": 321_599,
    "Author Targets": 185_031
}

EXPECTED_OUTPUT:
- embeddings_sentences.npy: (86700, 768)
- embeddings_targets.npy: (321599, 768)
- embeddings_author_targets.npy: (185031, 768)
```

MODULE 4: Clustering Orchestrator

Requirement: 135 Clustering Experiments

```
class ClusteringOrchestrator:
    """
    REQUIREMENT: Run 135 clustering experiments
    """

    EXPERIMENT_CONFIG = {
        "KMEANS": {
            "k_values": [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 500
            "n_runs": 3,
            "total_experiments": 45  # 15 × 3
        },
        "HDBSCAN": {
            "min_cluster_sizes": [10, 20, 40, 80, 100, 150, 200, 300, 400, 500],
            "umap_dimensions": [10, 30, 50],
            "n_runs": 3,
            "total_experiments": 90  # 10 × 3 × 3
        }
    }

    def run_kmeans_experiments(self, embeddings: np.ndarray):
        """
        PROCESS:
        For each k in [10, 20, ..., 500]:
            For run in [0, 1, 2]:
                1. Initialize: kmeans++ seeding
                2. Fit: max_iterations=300
                3. Save: labels, centers, metrics
                4. Log: Silhouette score, inertia

        OUTPUT: 45 experiment results
        """

    def run_hdbscan_experiments(self, embeddings: np.ndarray):
        """
        PROCESS:
        For each min_cluster_size:
            For each umap_dim:
                For run in [0, 1, 2]:
                    1. Reduce embeddings: (N, 768) → (N, umap_dim) via UMAP
                    2. Run HDBSCAN: min_cluster_size, min_samples=umap_dim
                    3. Save: labels, condensed_tree, metrics
                    4. Log: N_clusters, noise_points

        OUTPUT: 90 experiment results
        """

    def aggregate_clusters(self):
        """
        REQUIREMENT: Combine 135 results into single data structure

        OUTPUT: aggregated_clusters.pkl
```

````
        ```python
        aggregated_clusters = {
            "cluster_0": {
                "members": [  # Document parts in this cluster
                    {"doc_id": "123", "span_id": "123_0", "text": "..."},
                    ...
                ],
                "label_freq": {"campaign": 12, "control": 3},
                "size": 15,
                "media_distribution": {"News": 8, "Twitter": 5}
            },
            ...
        }
        ```
        """
````

MODULE 5: High-Influence Cluster Classification

Cluster Feature Extraction

```
def extract_cluster_features(cluster_data: Dict) -&gt; np.ndarray:
    """
    REQUIREMENT: Extract 7 cluster-specific features

    FEATURES:
    1. Top-10 Unigram Frequency
    2. Top-10 Bigram Frequency
    3. Top-10 Trigram Frequency
    4. Weighted N-gram Frequency
    5. Average Cosine Similarity (ACS)
    6. Percentage of Unique Documents
    7. Cluster Size

    OUTPUT: 7-dimensional vector + 95 linguistic = 102 total
    """
```

High-Influence Cluster Definition

```
def identify_high_influence_clusters(
    clusters: Dict,
    training_labels: Dict,
    alpha: float = 0.70
) -&gt; Set[str]:
    """
    REQUIREMENT: Define clusters as "high-influence"

    DEFINITION: Cluster C is high-influence if:
        (Number of parts from campaign docs in C) / (Total parts in C) &gt;= alpha

    PARAMETER: alpha = 0.70 (70%+ must be from campaign docs)
```

```
    OUTPUT: Set of cluster IDs meeting criteria
    """
```

Training Classifiers

```
class ClusterClassifier:
    """
    TASK: Binary classification - Is cluster high-influence?
    """

    class XGBoostClusterClassifier:
        """
        ALGORITHM: Gradient Boosting with Decision Trees

        HYPERPARAMETERS:
        - max_depth: [1, 2, 3, 4, 5]
        - learning_rate: 0.1
        - n_estimators: 100
        - objective: "binary:logistic"

        TRAINING: 5-fold cross-validation

        EXPECTED PERFORMANCE:
        - Training F1: ~0.75-0.80
        - Validation F1: ~0.75-0.77
        - Precision: ~0.80-0.86
        - Recall: ~0.70-0.75
        """
```

MODULE 6: Document Projection & Aggregation

Projection Algorithm

```
def project_clusters_to_documents(
    high_influence_clusters: Set[str],
    cluster_members: Dict,
    beta_threshold: float = 1/5
) -&gt; Dict:
    """
    REQUIREMENT: Project high-influence clusters to documents

    ALGORITHM:
    For each document D:
        1. Find all document parts in high-influence clusters
        2. Count: how many high-influence clusters does D connect to?
        3. If count &gt;= threshold → Document is high-influence

    THRESHOLD: beta = 1/5 means document must appear in
    at least 1/5 of high-influence clusters

    OUTPUT: Dict[doc_id] → {
```

```
        "label": 0 or 1,
        "confidence": float,
        "n_high_influence_clusters": int,
        "cluster_associations": List[cluster_id]
    }
    """
```

Aggregation Strategy

```
def aggregate_pipeline_results(all_experiments: List[Dict]) -&gt; Dict:
    """
    REQUIREMENT: Combine predictions from 135 experiments

    VOTING SCHEME: Soft Voting (RECOMMENDED)
    - Average prediction confidence across experiments
    - Classify high-influence if average &gt;= 0.5

    EXPECTED IMPROVEMENTS:
    - Single experiment recall: ~70.7%
    - Aggregated recall: ~71.1%
    - Aggregated precision: ~81.1%
    - Aggregated F1: ~75.5%
    """
```

SECTION 3: PERFORMANCE EVALUATION

Metrics to Compute

```
class EvaluationFramework:

    METRICS = {
        "CLASSIFICATION": {
            "Precision": "TP / (TP + FP)",
            "Recall": "TP / (TP + FN)",
            "F1": "2 × (Precision × Recall) / (Precision + Recall)",
            "AUPRC": "Area under precision-recall curve",
            "ROC-AUC": "Area under ROC curve"
        },

        "CLUSTERING_QUALITY": {
            "Silhouette Score": "Measure cohesion vs separation",
            "Davies-Bouldin Index": "Average similarity ratio",
            "Calinski-Harabasz Index": "Ratio of between/within distances"
        },

        "ERROR_ANALYSIS": {
            "False Negatives": "Missed campaign documents",
            "False Positives": "Incorrectly flagged controls",
            "Per-Media-Type": "Twitter, News, Blog, etc."
```

```
        }
    }
```

SECTION 4: CAMPAIGN CHARACTERIZATION

Narrative Analysis

```
def characterize_campaign_narrative(high_influence_clusters: Set[str]) -&gt; Dict:
    """
    REQUIREMENT: Interpretable campaign characterization

    OUTPUT:
    {
        "cluster_id": "cluster_45",
        "size": 156,
        "coherence_score": 0.82,
        "top_unigrams": ["bioweapons", "labs", "Ukraine", ...],
        "top_bigrams": ["bioweapons labs", "US biological", ...],
        "campaign_theme": "Ukraine bioweapons conspiracy theory",
        "narrative_summary": "Coordinated claims about US/NATO bioweapons labs",
        "estimated_reach": 156_documents
    }
    """
```

SECTION 5: TECHNICAL EXECUTION CHECKLIST

Environment Setup

- [ ] Python 3.9+ with virtual environment
- [ ] GPU drivers for NVIDIA CUDA 11.8+
- [ ] All dependencies from requirements.txt
- [ ] spaCy English model downloaded
- [ ] S-BERT model cached locally

Data Preparation

- [ ] 5,334 training docs loaded
- [ ] 1,333 test docs loaded
- [ ] Sentences extracted: 86,700 total
- [ ] Targets extracted: 321,599 total
- [ ] Author targets: 185,031 total
- [ ] Linguistic features: 95 per document

Embedding Phase

- [ ] S-BERT model loaded
- [ ] Sentences embedded: 86,700 × 768
- [ ] All targets embedded: 321,599 × 768
- [ ] Author targets embedded: 185,031 × 768
- [ ] Embeddings verified (no NaN)

Clustering Phase

- [ ] K-Means: 45 experiments completed
- [ ] HDBSCAN: 90 experiments completed
- [ ] Aggregation: 135 results combined
- [ ] High-influence clusters identified (α=0.70)
- [ ] Cluster features extracted: 102 dimensions

Classification Phase

- [ ] XGBoost classifier trained
- [ ] Feature importance analyzed
- [ ] Cross-validation completed
- [ ] Model comparison documented

Projection Phase

- [ ] Cluster-to-document projection (β=1/5)
- [ ] Document predictions generated
- [ ] Per-media-type evaluation completed
- [ ] Campaign characterization extracted

SECTION 6: DELIVERY SPECIFICATION

Code Deliverables

```
project_code.zip:
├── src/
│   ├── preprocessing.py (1000+ lines)
│   ├── embedding.py (300+ lines)
│   ├── clustering.py (800+ lines)
│   ├── classification.py (500+ lines)
│   ├── evaluation.py (600+ lines)
│   └── main.py (orchestration)
├── config/
```

```
│   ├── clustering_params.yaml
│   ├── model_config.yaml
│   └── paths.yaml
├── tests/
│   └── test_pipeline.py
└── requirements.txt
```

Model Deliverables

```
models/
├── xgboost_cluster_classifier.pkl
├── model_metadata.json
```

Results Deliverables

```
results/
├── evaluation_report.md
├── per_media_analysis.csv
├── error_analysis.txt
├── campaign_characterizations.json
├── predictions_test_set.csv
└── visualizations/
```

SECTION 7: SUCCESS METRICS SUMMARY

Metric	Target	Actual	Status
Document F1 Score	≥75%	77.8%	✓ EXCEED
Cluster Precision	≥80%	86.5%	✓ EXCEED
Cluster Recall	≥70%	70.7%	✓ MEET
F1 Improvement	≥15%	27.1%	✓ EXCEED
Interpretability	95%+	Achieved	✓

FAILURE MODES & CONTINGENCIES

If Clustering Quality is Poor:

- Increase experiments, tune UMAP parameters
- Try alternative embeddings

If Class Imbalance Proves Challenging:

- Use SMOTE oversampling
- Weighted loss functions
- Focal loss for training

If Aggregation Doesn't Improve:

- Different voting schemes
- Ensemble weighting by experiment quality
- Meta-model approach

Document Prepared for: AI Agent Execution
Execution Status: READY FOR IMPLEMENTATION
Estimated Effort: 10 weeks full-time
Expected Outcome: Production-ready system