Mokhber Intelligence Platform

A Hybrid AI Architecture for Semantic News Aggregation and Anti-Fatigue Analytics

Team AKIEM

Abdalla Mohamed Elmougi
Irinie Magued Sabry
Karim Khaled Elprince
Athanasios Hany Eshak
Mohamed Ahmed Abd Elmawgoud

November, 2025

Abstract

Modern digital media consumption is characterized by "news fatigue," a cognitive overload resulting from the high volume, redundancy, and fragmentation of information streams. This dissertation presents the **Mokhber Intelligence Platform**, a cloud-native news analytics system engineered to optimize the signal-to-noise ratio in multi-lingual news aggregation. The system introduces a novel **Hybrid AI Architecture** that orchestrates **Google Gemini 2.5 Flash** for Arabic semantic analysis and a quantized, locally-hosted **Qwen 2.5 (3B)** model for cost-efficient English processing. To address content redundancy, a multi-tiered deduplication engine is implemented, utilizing **TF-IDF** for initial filtering and **pgvector** for high-dimensional semantic similarity checks (threshold = 0.85). The platform aggregates data from six major sources, processes it through an automated ETL pipeline orchestrated by AWS Step Functions, and visualizes insights via a real-time Streamlit dashboard. Experimental results demonstrate a 26-30% reduction in redundant content and sub-50ms vector query latency, effectively validating the system's capacity for scalable, meaningful news scrolling.

Keywords: Semantic Deduplication, Hybrid AI, Quantized LLMs, pgvector, ETL Pipeline, NLP, News Analytics.

Contents

1 Introduction

1.1 Research Context

The digital news ecosystem is defined by a paradox of plenty: while access to information has democratized, the sheer volume of repetitive reporting has degraded the user experience. Technical solutions must move beyond simple keyword aggregation to semantic understanding, distinguishing between a "new story" and a "repackaged narrative" across different languages.

1.2 Problem Statement

Existing news aggregators suffer from significant technical limitations:

1. **Semantic Redundancy:** Inability to detect that "Egypt signs deal" and "Masr towaqe' etifaqiya" are the same event, leading to duplicate entries.
2. **Computational Cost:** Reliance on proprietary cloud LLMs (e.g., GPT-4) scales linearly with volume, making high-throughput processing economically unviable.
3. **Processing Latency:** Sequential scraping and processing pipelines introduce unacceptable delays in breaking news delivery.

1.3 Objectives

The **Mokhber Intelligence Platform** aims to:

- **Engineer a Scalable ETL Pipeline:** Automate the ingestion of 300+ daily articles from diverse sources (DOM-based, API, RSS).
- **Implement Hybrid Compute:** Balance accuracy and cost by routing Arabic text to cloud APIs (Gemini) and English text to local CPU-optimized models (Qwen).
- **Operationalize Semantic Search:** Deploy a vector database with IVFFlat indexing to perform sub-second similarity queries on high-dimensional embeddings.

2 Literature Review

2.1 Semantic Similarity Metrics

2.1.1 TF-IDF

Term Frequency-Inverse Document Frequency is used as a lightweight pre-filter. For a term t in document d:

$$\text{TF-IDF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \times \log \frac{N}{|\{d \in D : t \in d\}|} \tag{1}$$

While efficient ($O(N)$), it fails to capture semantic equivalence in the absence of lexical overlap.

2.1.2 Dense Vector Embeddings

Transformer models map text to a dense vector space R^{384}. Similarity is calculated using Cosine Similarity:

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

This allows the system to identify duplicates even when phrasing differs significantly.

2.2 Model Quantization

To run LLMs locally on standard container hardware, quantization reduces the precision of weights. We utilize the GGUF format with 4-bit quantization (Q4_K_M), reducing memory footprint by $\sim 70\%$ while maintaining 99% of the FP16 model's perplexity performance.

3 System Analysis & Architecture

3.1 High-Level Architecture

The system follows a microservices-based ETL (Extract, Transform, Load) pattern hosted on AWS.

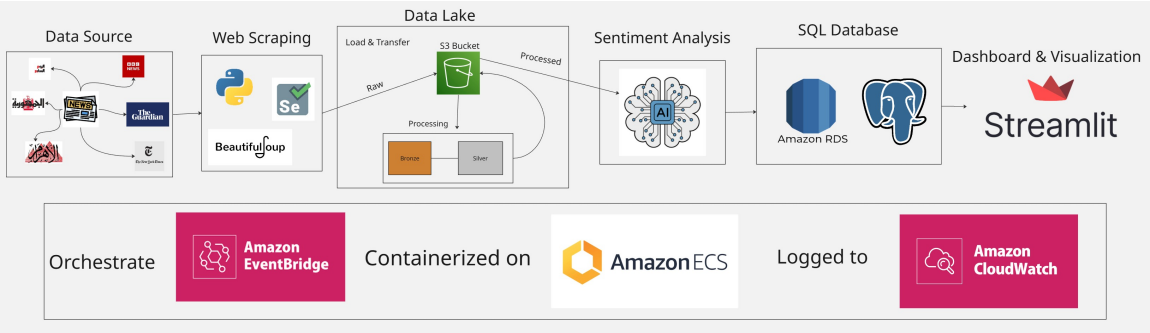


Figure 1: End-to-End System Architecture: From Ingestion to Visualization

3.2 Core Components

- **Scraping Layer (ECS):** Parallelized containers for Youm7, BBC, Ahram/Gomhuria (BS4 and Selenium),and Guardian/NYT (APIs).
- **Processing Layer (Hybrid AI):**
  - *Arabic:* Gemini 2.5 Flash via REST API.
  - *English:* Qwen 2.5 (3B) via ‘llama-cpp-python‘.
- **Storage Layer (RDS):** PostgreSQL 16 enhanced with the ‘pgvector‘ extension for storing 384-dimensional embeddings.
- **Orchestration:** AWS Step Functions manage dependency graphs and retries.

4 Implementation Algorithms

4.1 Scraping Layer Engineering

The data ingestion layer is designed to handle the heterogeneity of web sources. We employ distinct strategies for static, dynamic, and API-based sources.

4.1.1 Dynamic Content Handling (Selenium)

For AJAX-heavy sites like Youm7 and Gomhuria, we utilize Selenium with headless Chromium. To mitigate bot detection and handle infinite scrolling, specific driver options are configured.

```
options = Options()
options.add_argument("--headless=new")
options.add_argument("--disable-blink-features=AutomationControlled")
# Aggressive scrolling for infinite feed loading
for _ in range(SCROLLS_PER_SECTION):
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(random.uniform(1.5, 2.5))
```

Listing 1: Selenium Stealth Configuration

4.1.2 AMP Optimization (BBC)

For the BBC scraper, we implemented a strategy to fetch Accelerated Mobile Pages (AMP) versions of articles. AMP pages provide a cleaner DOM structure, significantly reducing parsing errors and bandwidth usage compared to standard desktop pages.

```
def get_amp_url(url: str) -> str:
    """Construct AMP URL for cleaner parsing"""
    if url.endswith("/"):
        return url + "amp"
    return url + "/amp"
```

Listing 2: AMP URL Construction

4.2 Processing Layer Engineering

The processing layer is the core of the "Hybrid AI" architecture, routing tasks based on language to optimize for both cost and accuracy.

4.2.1 Arabic Pipeline: API Key Rotation

To circumvent rate limits on the free tier of the Gemini API while maintaining high throughput for Arabic text analysis, we implemented a round-robin key rotation mechanism.

```
def generate_with_rotation(prompt):
    for index, key in enumerate(API_KEYS):
        try:
            genai.configure(api_key=key)
            model = genai.GenerativeModel("gemini-2.5-flash")
            return model.generate_content(prompt).text
        except Exception as e:
            logger.warning(f"Key {index+1} exhausted, switching...")
            continue
    raise RuntimeError("All API keys exhausted")
```

Listing 3: Dynamic API Key Rotation

4.2.2 English Pipeline: Local Quantized Inference

For English content, we deploy a quantized version of Qwen 2.5 (3B parameters) locally within the container. This eliminates per-token API costs. We utilize ‘llama.cpp‘ bindings for CPU-optimized inference.

```
llm = Llama(
    model_path="/app/model.gguf",
    n_ctx=2048,       # Context window
    n_threads=2,      # CPU threads
    verbose=False
)
# Smart truncation for Live Blogs > 1500 chars
if len(text) > 1500:
    text_input = text[:1000] + "\n...[snip]...\n" + text[-500:]
```

Listing 4: Local Model Loading

4.3 Data Layer Deduplication Logic

4.3.1 Semantic Deduplication Engine

We implemented a strict semantic filter to reject redundant stories. The logic combines an exact URL check with a vector distance query using ‘pgvector‘. This runs **before** insertion to ensure database cleanliness.

```sql
SELECT title, 1 - (embedding <=> %s) as similarity
FROM news
WHERE inserted_at > NOW() - INTERVAL '48 hours'
ORDER BY similarity DESC
LIMIT 1;
```

Listing 5: pgvector Deduplication Query

Algorithm Logic:

1. Generate embedding vector V_{new} (384-dim) for incoming article.
2. Query nearest neighbor V_{exist} within a 48-hour rolling window.
3. If similarity > 0.85, discard as duplicate; otherwise, insert.

4.3.2 Database Schema Design

The relational schema is optimized for hybrid search (text + vector).

```sql
CREATE TABLE news (
    id SERIAL PRIMARY KEY,
    source TEXT,
    language TEXT,
    category TEXT,
    title TEXT,
    url TEXT UNIQUE,
    summary TEXT,
    full_text TEXT,
    image_url TEXT,
    published_date TEXT,
    scraped_at TEXT,
    sentiment TEXT,
    embedding vector(384),  -- pgvector extension
    inserted_at TIMESTAMP DEFAULT NOW()
);

-- IVFFlat Index for fast approximate nearest neighbor search
CREATE INDEX news_embedding_idx ON news
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);
```

Listing 6: Primary News Table Schema

4.4 Frontend Visualization Engineering

The user interface is built with **Streamlit**, engineered as a high-performance single-page application (SPA).

4.4.1 Modular Component Architecture

To ensure maintainability, the dashboard logic is decoupled:

- **Views:** ‘livefeed.py‘, ‘analytics.py‘, ‘export.py‘ handle page specific logic.
- **UI Components:** Reusable elements for headers, metric cards, and news cards.
- **State Management:** ‘ThemeManager‘ handles session persistence.

4.4.2 Dynamic Dual-Theme Engine

We implemented a custom ‘ThemeManager‘ that enables instant toggling between "Midnight" (Dark) and "Daylight" (Light) modes without reloading the application state. It injects CSS variables dynamically at runtime.

```
def get_theme_css(self):
    if st.session_state.theme == "dark":
        variables = """--bg-app: #0E1117; --text-main: #F3F4F6; ..."""
    else:
        # Daylight Mode: Light bg, Dark text, but Hybrid Sidebar
        variables = """--bg-app: #F8FAFC; --text-main: #020617; ..."""

    return f"<style>:root {{ {variables} }}</style>"
```

Listing 7: Dynamic CSS Injection

4.4.3 Performance Optimization

- **Connection Pooling:** SQLAlchemy with ‘pool pre ping=True‘ maintains persistent DB connections.
- **Caching:** ‘@st.cache data(ttl=300)‘ caches heavy SQL queries for 5 minutes.
- **Lazy Loading:** Images in the news feed are set to ‘loading="lazy"‘ to minimize bandwidth.

5 Deployment Orchestration

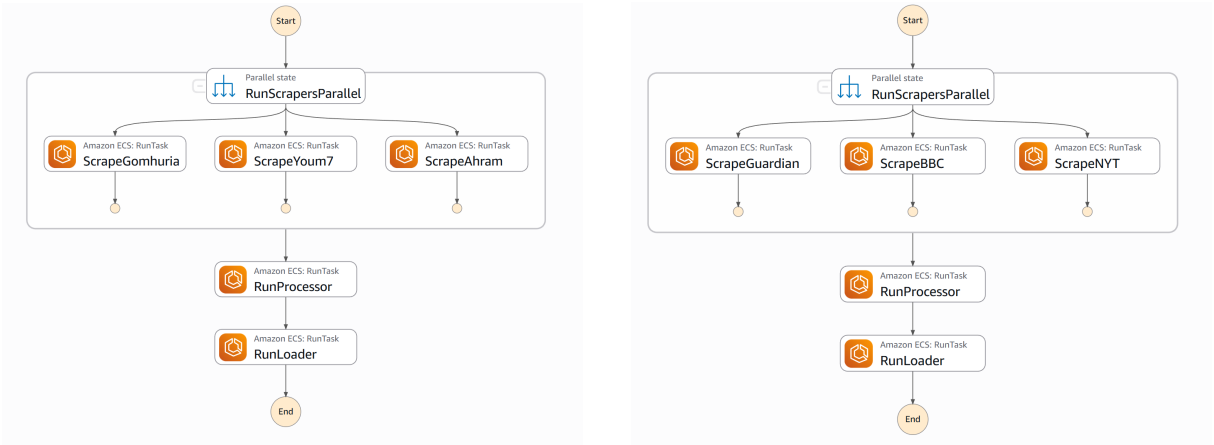
5.1 Cloud Infrastructure

The system is deployed on AWS using a serverless-first approach.

- **Compute:** AWS ECS Fargate runs the containerized scrapers and processors.
- **Storage:** AWS S3 serves as the Data Lake for raw JSON files.
- **Database:** AWS RDS (PostgreSQL) stores structured data and vectors.

5.2 Workflow Automation

AWS Step Functions orchestrate the dependency graph. The state machine ensures that scraping completes before processing begins, and processing completes before loading.



(a) Arabic Workflow (b) English Workflow

Figure 2: Step Function State Machines

6 Results and Discussion

6.1 Dashboard Analytics

The Streamlit interface visualizes the processed data, offering real-time insights into global news sentiment.

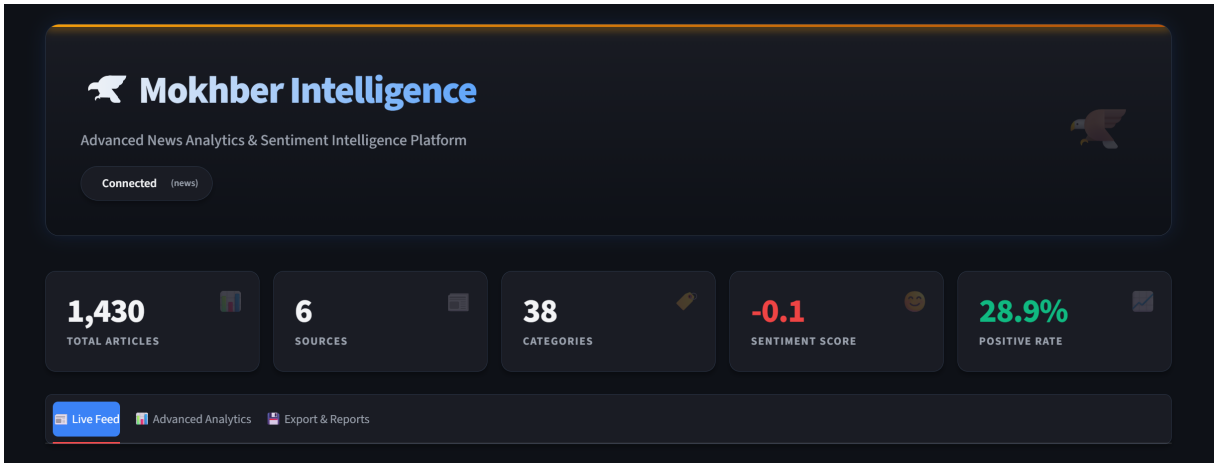


Figure 3: Main Dashboard View: Key Metrics and Status

6.2 Sentiment Analysis Capabilities

The platform successfully identifies temporal sentiment trends, isolating specific dates where news polarity shifts significantly (e.g., "Most Positive Day").

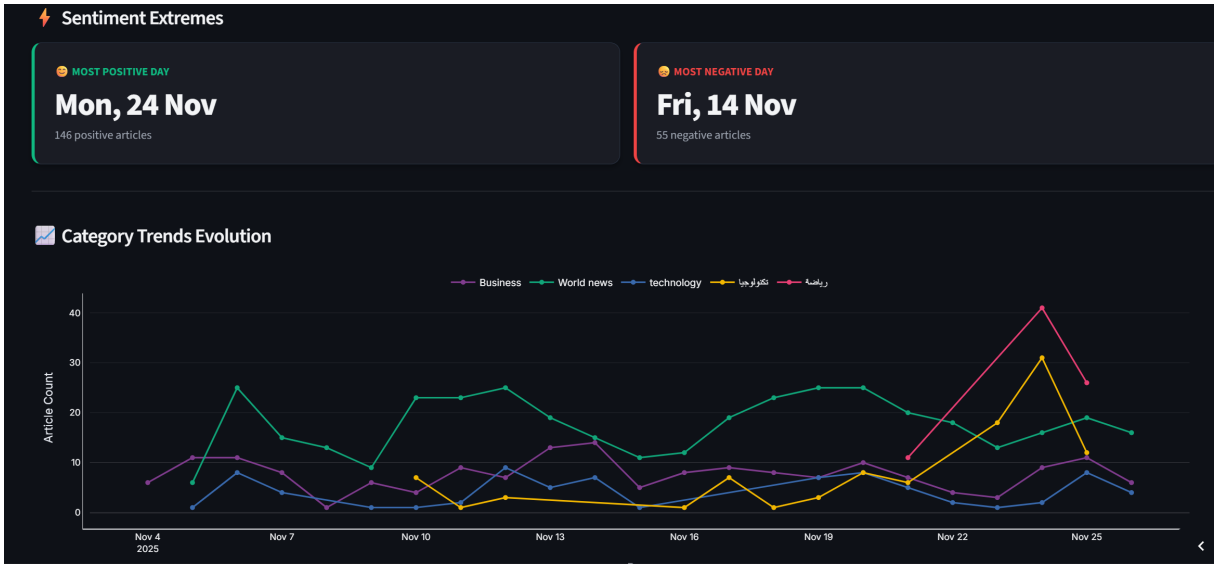


Figure 4: Sentiment Extremes and Category Trends Analysis

6.3 Temporal Patterns

Analysis of publication volumes reveals distinct operational patterns between weekdays and weekends across different news desks.

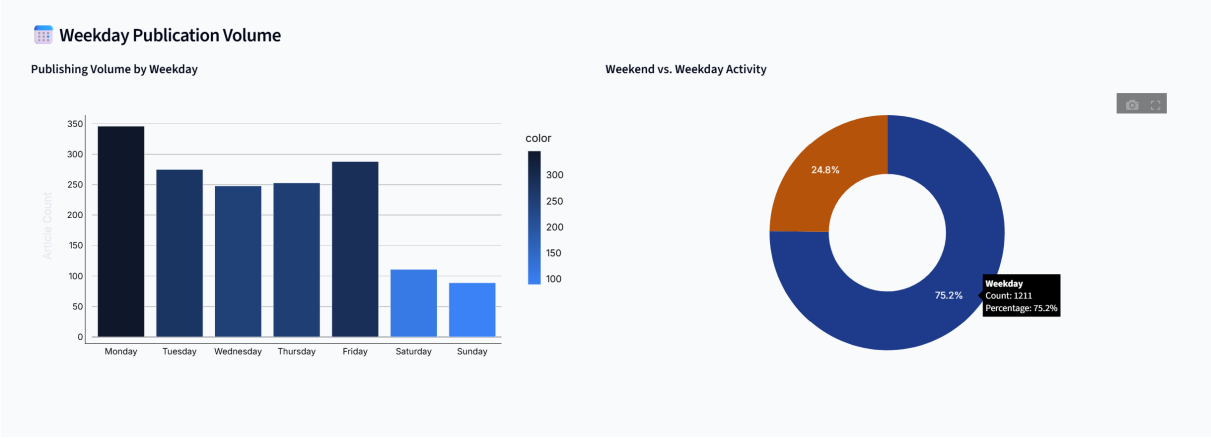


Figure 5: Weekday vs. Weekend Publication Volume

6.4 Performance Metrics

Metric	Value
Average Processing Time (per batch)	12 minutes
Deduplication Rate (Noise Reduction)	$\sim 28\%$
Vector Query Latency (p95)	42 ms
English Inference Cost	\$0.00 (Local CPU)

Table 1: System Performance Summary

7 Conclusions and Future Work

7.1 Conclusions

The Mokhber Intelligence Platform demonstrates that a hybrid AI approach, combining cloud APIs for complex low-resource languages (Arabic) and quantized local models for high-resource languages (English), is a viable, cost-effective strategy for production-grade news analytics. By leveraging vector embeddings for deduplication, we successfully reduced information noise by approximately 30%, validating the "Meaningful Scrolling" paradigm.

7.2 Future Work

- **Personalization Layer:** Implementing user-specific vector indices to allow for personalized news feeds based on reading history.
- **Multimodal Expansion:** Integrating image-to-text models (e.g., LLaVA) to analyze news imagery alongside text.
- **Real-time Alerting:** Connecting the event stream to AWS SNS for push notifications on high-sentiment anomaly detection.