Chronicle: Temporal Exploration of Nigerian Archives

Background

Archivi.ng is a digitised archive of Nigerian newspapers, magazines, transcripts, and other primary sources. At present, Archivi.ng's search interface (https://archivi.ng/search) accepts natural language queries and returns relevant documents from the archive, which users can browse page by page.

The feature this document proposes is called Chronicle. It extends Archivi.ng's search by providing a visual timeline of the original search results. The search results are grouped into periods, and each period cluster is summarized, titled and enriched with context.

This project is proposed to be developed as part of the 2026 Archivi.ng Fellowship (Cohort #002). The fellowship's theme— "The More Things Change"—inquires about temporal patterns in Nigerian history. By automating the revelation of patterns, Chronicle directly interacts with this idea.

Architecture

Overview

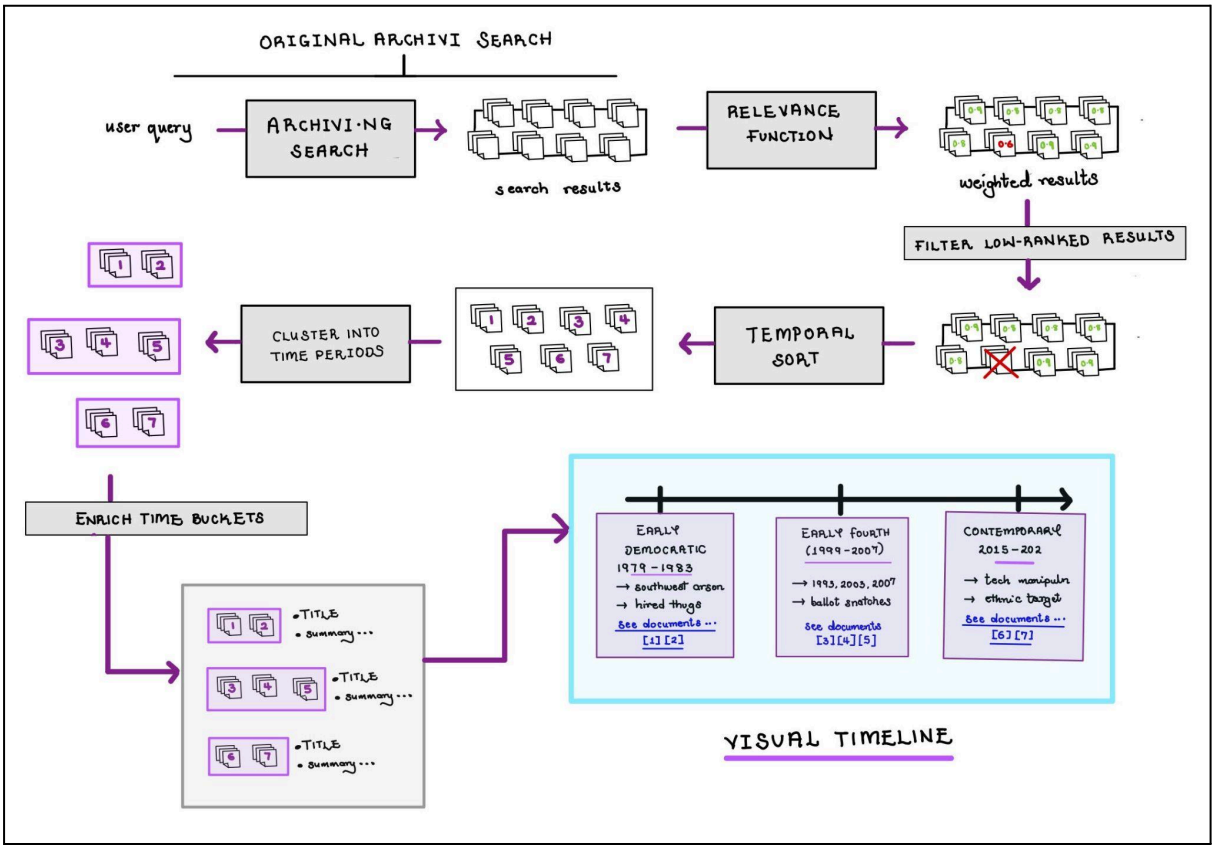

Figure 1: Chronicle extends Archivi.ng's search with temporal analysis. Search results are filtered for relevance, sorted chronologically, clustered into distinct time periods, enriched with summaries and titles, then visualized as an interactive timeline.

Chronicle: Temporal Exploration of Nigerian Archives

Relevance Filtering

The results from Archivi's classical search are filtered for relevance. This reduces the search results to high-signal documents to avoid unclear temporal clusters down the line.

Final selections could be made with a percentile-based (top N%, e.g. top 60%) strategy, or an adaptive threshold strategy (keeping documents above mean or median relevance). Testing with real archive queries should illuminate a better choice between both strategies.

Temporal Bucketing

The documents left over after relevance filtering are then separated into temporal buckets. This begins with a temporal sort which arranges the documents by the dates they were published.

The timespan is calculated by finding the time difference between the earliest and latest document. This is used to determine the granularity of the timeline (or the bucket size in time). For example: a span > 50 years → decade buckets, span of 20-50 years → 5-year buckets, ..., span < 1 year → monthly or weekly buckets.Documents are then grouped into their buckets.

Bucket sizes vary naturally—some periods may contain 5 documents while others contain 30+. This variance is informative rather than problematic: dense periods indicate heightened coverage, while sparse periods show reduced attention. Buckets that end up with no documents are ignored.

Bucket Enrichment

This is the step where we describe our temporal clusters (buckets).

First, the cluster is titled and summarized. These details are generated from the most relevant documents of the period via LLM query. The exact number of top documents (e.g. 5-10, or top 10%) and the LLM choices are hyperparameters that will be decided by testing.

For example: the query "economic crises" can return clusters that will ultimately be titled "Structural Adjustment Program" for a 1990-2000 cluster, and "Naira Devaluation" for the 2010-2020.

A cover story (a representative document) is selected via its relevance to the title. We run a re-query with the title as the search term and have the search restricted to the time period of that bucket. The document with the highest relevance score is the cover story.

Visualization and User Interface

Finally, the enriched buckets are displayed along a timeline interface on the Archivi.ng site. Crucially, users should be able to expand each time period and explore documents that make it up.

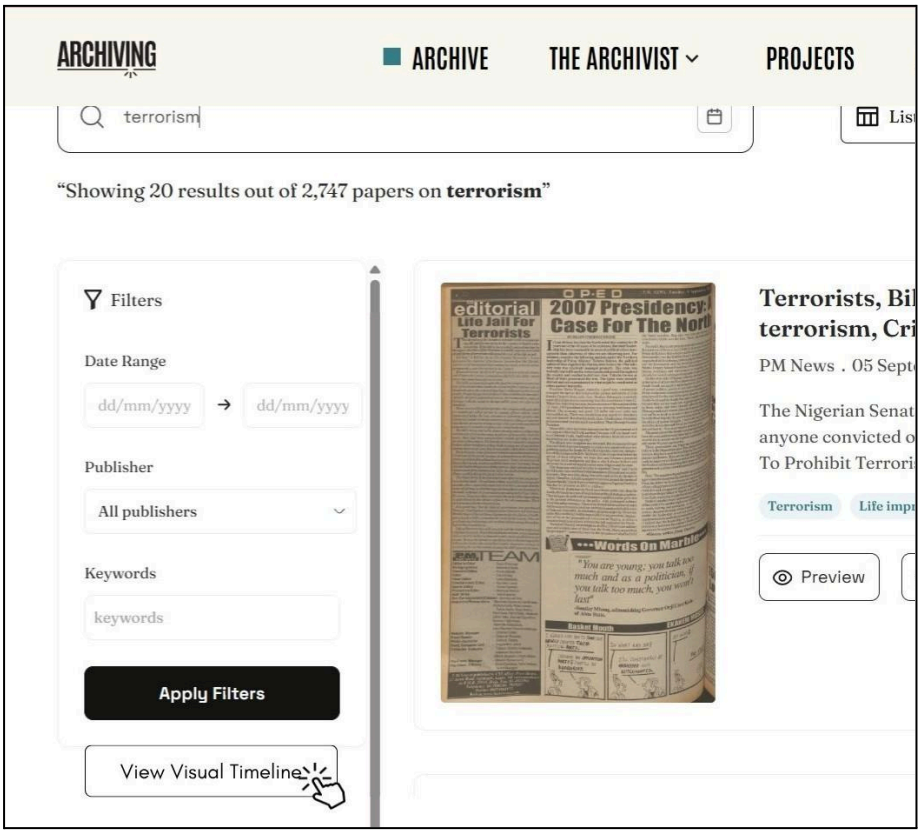

Figure 2: Archivi.ng page showing new "View Visual Timeline" button that triggers the Chronicle view.

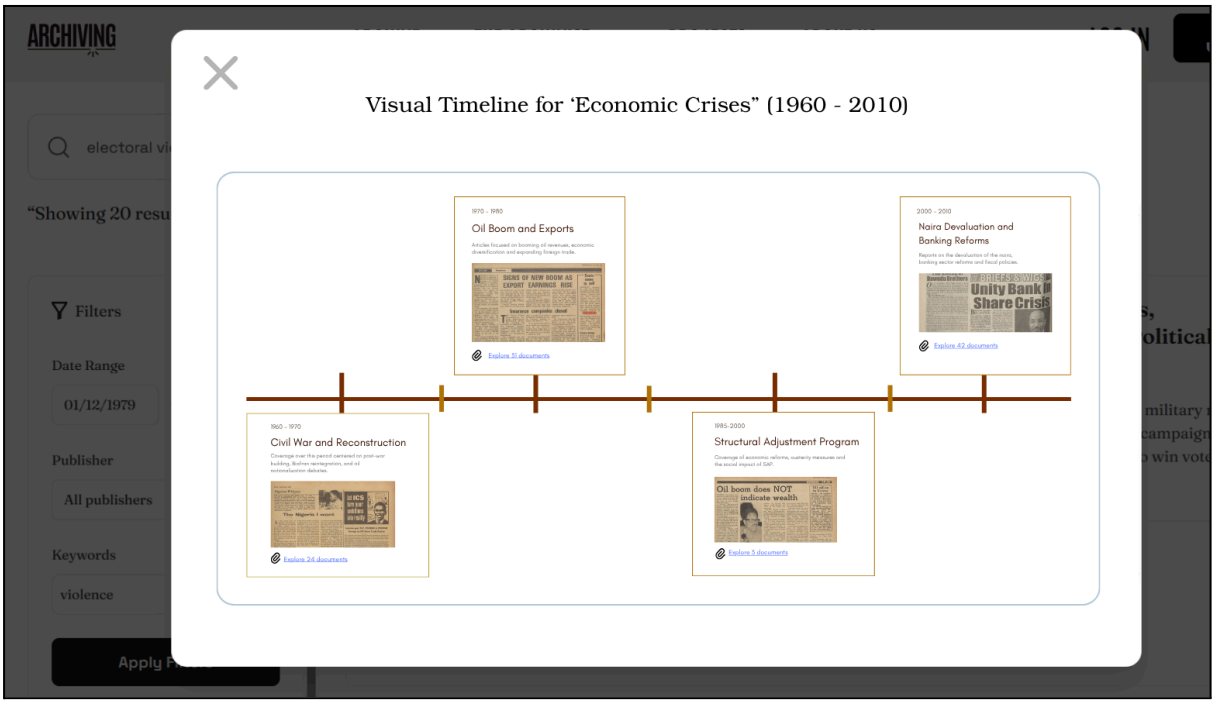

Figure 3: The Chronicle view pops up on the Archivi.ng site.

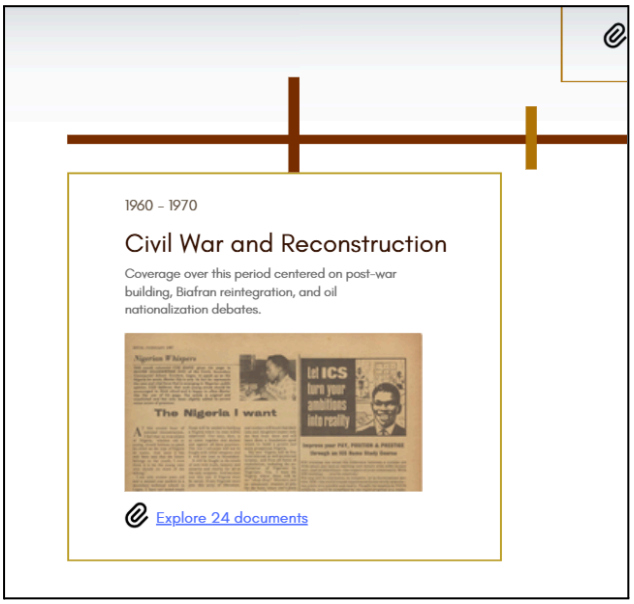

Figure 4: Zoomed-in view of time bucket.

Caching

A performance optimization is to cache requests and results from this feature. The benefits include faster response times and fewer API calls (saved costs).

Cache keys are made up of the natural language query + filters. These keys and their response objects are stored in a key-value store (like Redis). The cache will

be a fixed size and is refreshed based on the recency of the query. The size of the cache is chosen by finding a sensible balance between storage costs and API calls (to the search engines, and LLMs), but we can expect it to be fairly large— and sized to accommodate the majority of daily query patterns.

A design choice to consider is asking users to cache their own requests. A small dismissable prompt that says “will you re-run this query? Help archivi cut costs by saving it.” Affirming that it will re-run prioritizes that response’s life in the cache.