An End-to-End MLOps Pipeline for Automated News Processing, Topic Clustering, and Structured Summarization

Keetawan Limaroon 64070501005
Punnawat Namwongsa 64070501032
Patcharaphon Santhitikul 64070501037
Pakawat Phasook 65070503430
Rapepong Pitijaroonpong 65070503434

Super AIOps Engineer Team, KMUTT

{keetawan.limaroon,punnawat.namwongsa,patcharaphon.s}@kmutt.ac.th

{pakawat.phasook,rapepong.pitijaroonpong}@kmutt.ac.th

May 28, 2025

Abstract

This paper introduces a fully automated, end-to-end MLOps pipeline designed for processing and understanding large volumes of Thai-language news articles. The system seamlessly integrates data scraping, semantic embedding using state-of-the-art multilingual models (e.g., BGE-M3), unsupervised clustering via DBSCAN, and large language model (LLM)-driven summarization using Typhoon2.1-Gemma3-4B. The architecture is orchestrated with Apache Airflow and monitored through MLflow for version control and lifecycle management. A robust retraining mechanism, powered by contrastive learning, ensures that model performance adapts to changing news dynamics over time. The pipeline is containerized and deployable via Docker Compose, with a frontend for user interaction and real-time summary access. Evaluation using the Calinski-Harabasz Index confirms the clustering effectiveness, and the system supports future scalability, automation, and cloud integration. Code is publicly available at `https://github.com/WTFPUn/MLOpsProject`. This work demonstrates how modern MLOps practices can operationalize unsupervised NLP workflows for real-world applications in media monitoring and news analytics.

1 Introduction

1.1 Background

In the modern newslandscape, information is no longer delivered as a single, self-contained article. Instead, real-world events—such as political developments, public health issues, legal cases, celebrity scandals, or social media controversies—are reported through a sequence of fragmented stories across different platforms, sources, and timelines. Each article may contain new details, public reactions, or follow-up developments. While this dynamic reporting style enhances immediacy and public engagement, it presents a new challenge: readers must navigate through multiple related articles to fully understand the complete story. The burden of connecting these fragmented narratives often falls entirely on the reader.

1.2 Problem Statement

This fragmentation of news content makes it increasingly difficult for readers to track events that unfold over time. For instance, a single court case might generate dozens of articles spread over several weeks, each containing only part of the full context. Without a structured method to group and summarize related news stories, readers can easily become overwhelmed or misinformed. The lack of semantic linkage between articles also poses challenges for automated systems and platforms attempting to provide coherent summaries or alerts. As a result, readers may miss critical developments or be forced to spend excessive time understanding a single topic.

1.3 Objectives

This project aims to:

1. Design and implement an automated news processing pipeline that integrates data scraping, semantic clustering, summarization, and model monitoring in a fully orchestrated workflow.

2. Develop an efficient clustering system to group semantically related news articles, enabling readers to follow ongoing events across multiple sources and timelines.

3. Generate concise and coherent summaries for each news cluster using large language models (LLMs), helping users quickly understand complex or evolving stories.

4. Build a scalable API and front-end interface for real-time access to clustered and summarized news, supporting both programmatic and user-facing consumption.

5. Establish comprehensive monitoring and retraining mechanisms to ensure continuous model performance, system reliability, and long-term maintainability.

2 System Architecture Overview

This section describes the full architecture of our automated news processing system, designed to handle end-to-end tasks from data collection to model deployment and retraining. The pipeline is orchestrated using **Apache Airflow**[1], enabling task scheduling, dependency management, and automated execution of each processing step. Model lifecycle is managed via **MLflow Model Registry** [2], providing robust versioning, tracking, and deployment workflows.

2.1 Architecture Diagram

The system is structured as a Directed Acyclic Graph (DAG) within Airflow and divided into three functional stages:

1. Scrape Data
2. Inference & Summarization
3. Model Monitoring & Retraining

All data is exchanged through a centralized storage system (Amazon S3), and model versioning is handled by MLflow Model Registry. Summarization is provided by a Flask [3] API backed by an LLM running on Google Colab [4], with ngrok [5] used to expose the endpoint.

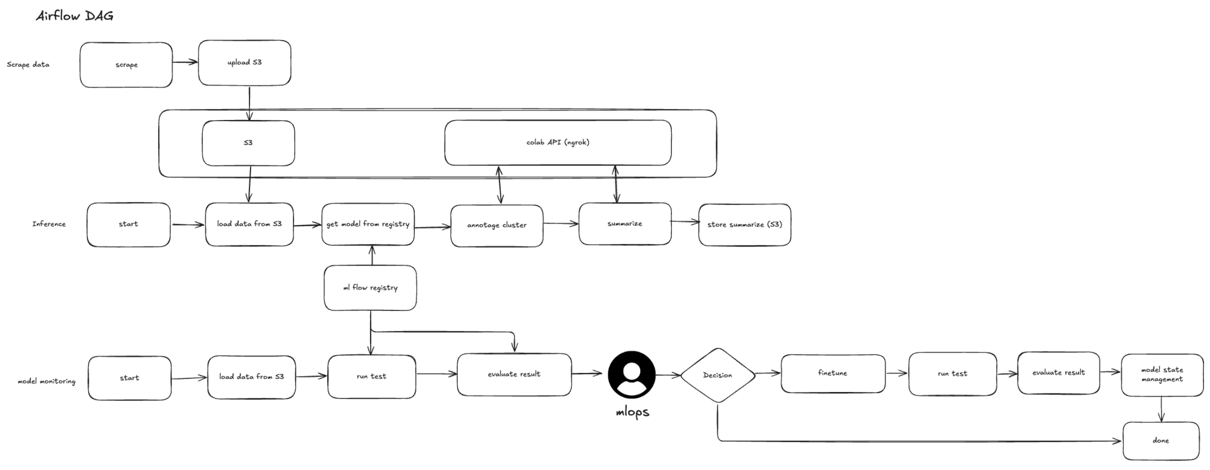


Figure 1: DAG diagram showing automated news processing, clustering, summarization, evaluation, and model management.

2.2 Detailed Pipeline Process: Airflow DAG

The automated pipeline is composed of three main stages as orchestrated by the Airflow DAG: (1) Scrape Data, (2) Inference & Summarization, and (3) Model Monitoring & Retraining. Each stage consists of several dependent tasks that are connected via centralized storage (Amazon S3) and interact with model versions stored in MLflow.

- **Scrape Data**
  - `scrape`: Extracts raw news data from online sources through scrapers or APIs.
  - `upload S3`: Uploads the scraped data to Amazon S3 for storage and downstream use.
- **Inference & Summarization**
  - `load data from S3`: Loads input data from the cloud storage.
  - `get model from registry`: Retrieves the most recent production model from MLflow Model Registry.
  - `annotate cluster`: Applies semantic embeddings (e.g., BGE-M3 [6]), followed by clustering (e.g., DBSCAN) to group related news articles.
  - `summarize`: Sends representative articles to a summarization API hosted via Flask on Colab using ngrok, and stores the generated summaries.

- **Model Monitoring & Retraining**
  - `run test` & `evaluate result`: Evaluates model performance using metrics like Calinski-Harabasz Index.
  - `finetune`: Triggered only if evaluation scores fall below a defined threshold. It retrains the summarization model on recent data.
  - `run test` (post-finetune) & `evaluate result`: Compares the retrained model's performance with the previous version.
  - `model state management`: Promotes the new model to production in MLflow if performance is improved.
  - `done`: Ends the pipeline execution.

This stage-wise breakdown aligns directly with the structure of the Airflow DAG and enables modular, scalable automation of the news processing workflow.

2.3 Orchestration with Apache Airflow

Apache Airflow orchestrates the entire pipeline using Directed Acyclic Graphs (DAGs). Each node represents a task, and edges define the execution dependencies. The key features of Airflow orchestration in our system include:

- **Scheduled Execution**: DAGs are configured to run daily, processing newly scraped news articles.
- **Dependency Management**: Tasks are executed in a strict order to ensure data and model integrity.
- **Conditional Logic**: The pipeline dynamically decides whether to trigger fine-tuning based on evaluation metrics.
- **Retries and Failure Recovery**: Airflow supports retry policies and timeout logic, improving system robustness.
- **Monitoring and Logging**: Each task logs execution metadata, accessible via the Airflow web interface.

Airflow's orchestration ensures that the system is not only automated but also traceable, debuggable, and extendable for future needs.

2.4 Model Registry with MLflow

MLflow is integrated as the centralized Model Registry to manage the entire lifecycle of machine learning models used in this system. It offers key functionalities such as version tracking, metadata logging, and deployment management. Each time the model is trained or fine-tuned:

- The model is saved to the MLflow Registry with associated training parameters and evaluation metrics.

- A version number is automatically assigned and tagged (e.g., “Staging”, “Production”).
- Airflow retrieves models directly using MLflow’s Python client or REST API for inference and evaluation.
- Post-evaluation, if the fine-tuned model outperforms the current production model, it is promoted within the registry.

By integrating MLflow, the system ensures reproducibility, safe rollback, and robust lifecycle management, which are critical for long-term maintainability in an automated MLOps pipeline.

3 Dataset Description

This section outlines the dataset used in the project, including its sources, scraping methodology, and statistical properties. The dataset serves as the foundation for downstream processing, such as semantic clustering and summarization.

3.1 Data Sources

The news articles used in this system were collected from the official website of Thairath Online[1]—one of Thailand’s most visited and widely referenced news platforms. It covers a broad range of topics including politics, society, regional news, and breaking headlines. The dataset focuses on textual news content published within the last 7 days, updated on a rolling basis.

3.2 Scraping Methodology

To systematically collect news articles, a custom Python-based web scraper was developed using the `requests` and `BeautifulSoup` libraries. The scraper is integrated into an Airflow DAG and scheduled to run daily. The process is outlined below:

Targeted Sections

The scraper is configured to extract articles from specific news categories:

- `local` – Regional or provincial news
- `society` – Social issues, events, and public interest
- `politic` – Political updates and government activities

Link Collection

- The scraper paginates through each category using the URL format: `/news/{category}/all-latest?filter=7&page={page_number}`.
- The value `filter=7` ensures that only news from the past 7 days is retrieved.

[1]`https://www.thairath.co.th`

- Article metadata such as title and URL are parsed from the JavaScript object embedded in the `__NEXT_DATA__` script tag.

Article Extraction

Once article URLs are collected, each article page is fetched individually to extract the following:

- **Title:** Retrieved from metadata or page headers.
- **Publication Date:** Parsed from the `publishLabelThai` field in the JSON or fallback date strings in the HTML.
- **Content:** Constructed from the `paragraph` elements in the JSON blob, or from HTML `<p>` tags if necessary.
- **Tags:** Extracted from structured JSON metadata or from rendered HTML tag elements.

Data Storage

All articles are stored as rows in a CSV file with UTF-8-SIG encoding to ensure Thai character compatibility. Each record contains the following fields:

- `title` – Article headline
- `url` – Direct link to the article
- `scraped_at` – Timestamp of the scraping
- `published` – Original published date (in Thai format)
- `content` – Main body of the news article
- `tags` – List of associated tags

Robustness

The scraper includes:

- **Retry logic** for failed requests and malformed responses
- **Deduplication** using a Python `set` to avoid repeated scraping of the same article
- **Fallback extraction** in case JSON blobs are missing or pages are dynamically structured
- **Respect for ethical scraping**, including request throttling and compliance with `robots.txt`

4 Data Analysis (EDA)

This section presents exploratory data analysis (EDA) to better understand the characteristics of the scraped news articles. The focus is placed on the publication trends, tag distributions, and article length statistics.

4.1 Overview of Dataset

The dataset comprises news articles scraped from *ThaiRath* across categories such as politics, society, and local news. After preprocessing, the final dataset consists of 524 articles with key metadata including title, publication date, tags, and full content.

- Total articles: **524**
- Articles with non-empty content: 524 (100%)
- Articles with at least one tag: 522 (99.6%)
- Time range: 1-month scrape window

4.2 Publication Date-Time Distribution

4.2.1 Articles Published Per Day

A bar chart showing the number of articles published per day reveals the content generation pattern of *ThaiRath*. Most news outlets appear to follow a regular publishing schedule, with peaks typically observed on weekdays. Lower volumes are seen during weekends.

Interestingly, there is a significant spike in article volume starting from May 18 onward. This increase corresponds to a period of heightened political activity in Thailand, particularly related to the **Senate election**. During this time, media outlets published a large number of politically-focused articles to cover ongoing developments, which explains the peak in news volume on May 20.

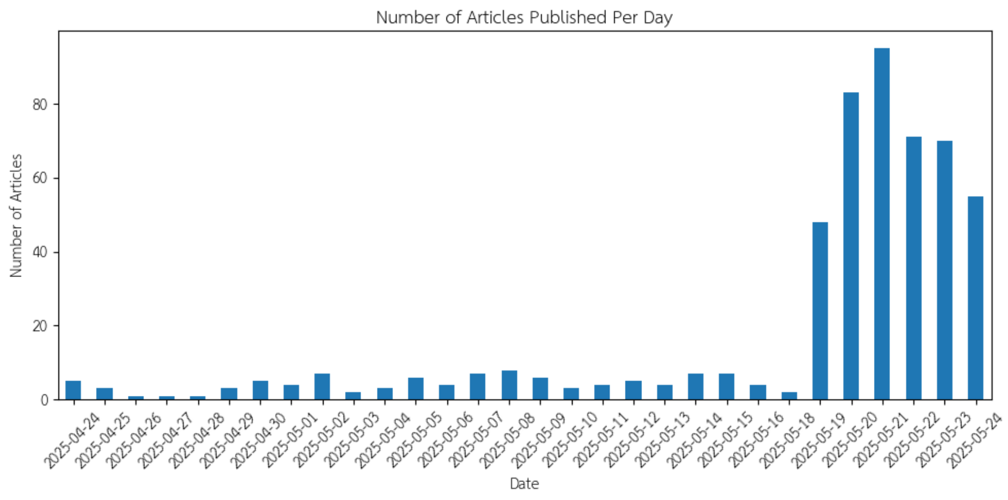


Figure 2: Number of articles published per day

4.2.2 Articles Published by Hour

A histogram of publication hours illustrates that articles are frequently published between 7 AM to 11 AM and again around 4 PM to 6 PM. This aligns with expected editorial cycles and peak readership windows.

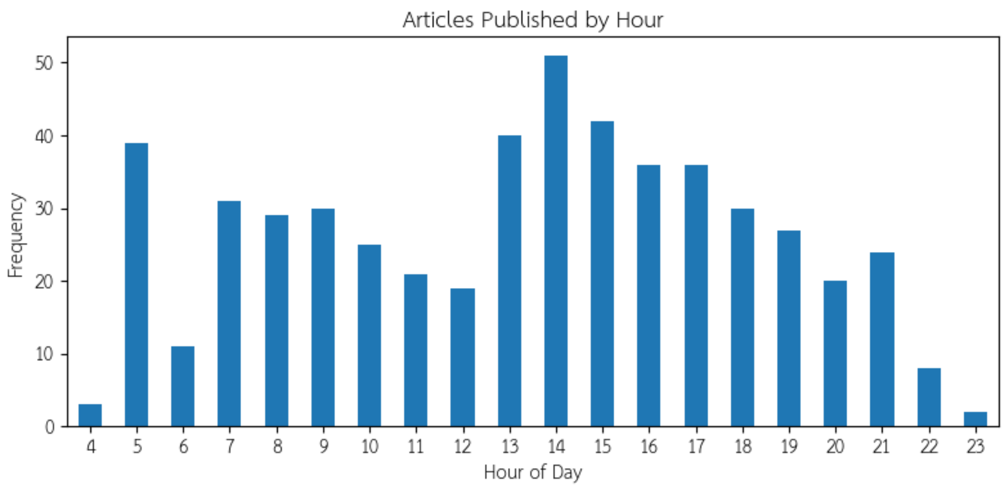


Figure 3: Distribution of publication time by hour

4.3 News Tags and Topics

4.3.1 Top 10 Frequent Tags

A bar chart of the 10 most frequently occurring tags confirms the dominance of political and societal issues. These tags offer an intuitive summary of public interest and media coverage trends.

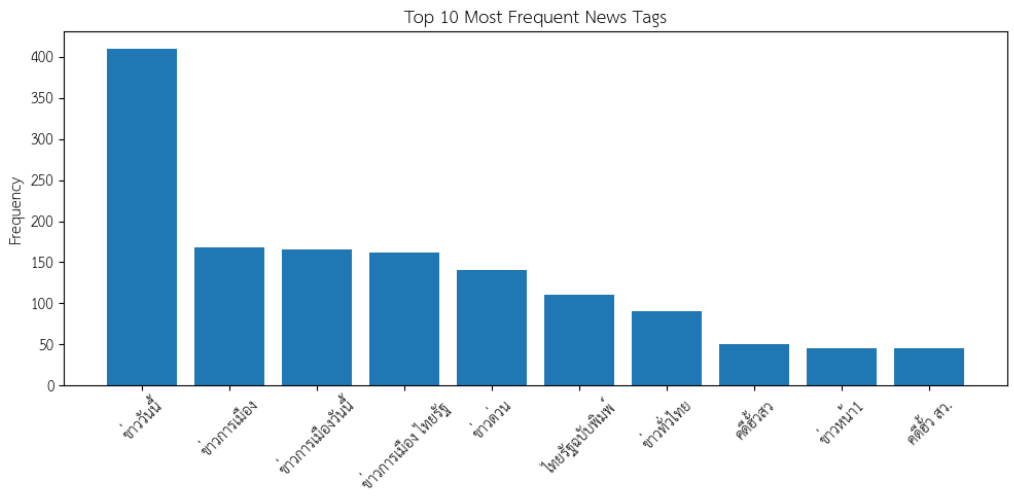


Figure 4: Top 10 most frequent news tags

4.4 Article Length Analysis

To understand the depth and complexity of content, we analyze the length of articles using word count:

- Mean article length: ∼121 words
- Median article length: 95 words

- Shortest article: 11 words (likely breaking news)
- Longest article: 1,273 words (in-depth analysis or interviews)

A histogram of word counts shows a right-skewed distribution, indicating that most articles are moderately sized, with a small number being extremely detailed.

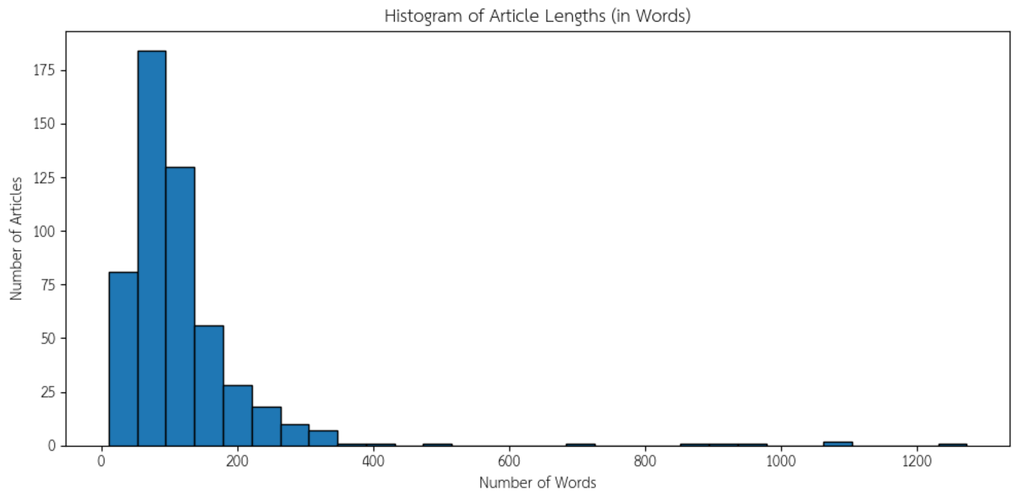


Figure 5: Histogram of article lengths (in words)

Together, these EDA findings offer a comprehensive view into the structure and trends within the *ThaiRath* news dataset, laying the groundwork for effective downstream processing and modeling.

5 Feature Engineering

5.1 Input Features

For this project, the primary input feature used in all downstream tasks was the full `content` of each news article. This field provides the richest semantic information, encompassing facts, descriptions, quotations, and other textual clues that are essential for clustering or summarization.

While the `title` and `tags` fields were also scraped and stored, they were not used as input features in our core machine learning pipeline. The decision to exclude them was based on two observations: (1) headlines are often sensationalized or stylized for clickbait purposes, and (2) tags are inconsistently applied and often contain non-informative editorial terms. These limitations reduced their value as standalone features for embedding.

In summary, the `content` field was selected for its completeness, linguistic richness, and consistency across articles.

5.2 Text Cleaning

To ensure clean and consistent input for downstream models, a light text-cleaning process was applied. Since the input data consisted of news articles written in Thai, no lowercasing or stemming was necessary. However, we did remove common formatting artifacts and visual noise.

The cleaning process used regular expressions (`re.sub`) to remove:

- Ellipses ("...")
- Redundant whitespace and line breaks

This minimal cleaning step helped reduce ambiguity during tokenization and improved the performance of subsequent sentence embedding processes. Stopword removal and advanced normalization were deliberately skipped to preserve contextual integrity and avoid unintended semantic loss.

6 ML Model Development & Comparison

6.1 Problem Formulation

The core task addressed in this project is unsupervised text clustering. The aim is to group news articles that report on the same real-world events, even if those articles are published days apart or across different sections. By clustering semantically related articles, we enable the system to summarize entire stories spanning multiple updates.

6.2 Text Embedding Model

We employ the **bge-m3** [6] multilingual sentence embedding model, which achieves state-of-the-art performance on MIRACL and Mr.TyDi Thai benchmarks. This model is used to generate dense semantic embeddings for each news article. The output vectors, known as *ColBERT vectors*, capture the contextual meaning of each document and are well-suited for clustering tasks involving Thai-language text.

6.3 Clustering Algorithms Compared

To identify the most suitable clustering method for grouping semantically similar news articles, we focus on density-based algorithms, which do not require prior knowledge of the number of clusters. This is critical in our use case, where the number of evolving news topics per week is unknown and variable.

We compare the following clustering methods:

- **DBSCAN**: A density-based clustering algorithm that groups together closely packed points while marking outliers as noise. It is robust to variations in cluster shape and does not require a predefined number of clusters.
- **HDBSCAN**: A hierarchical extension of DBSCAN that can detect clusters of varying densities more effectively. It also eliminates the need for manual tuning of the epsilon parameter.
- **OPTICS**: Orders points based on density reachability and produces a reachability plot that allows cluster extraction at different densities.

We exclude **K-Means** from consideration in this task. K-Means requires a fixed value of k, which is impractical for streaming or weekly-updated news content. Furthermore, determining the optimal k using elbow or silhouette methods is computationally intensive due to the scale of the data.

6.4 Evaluation Strategy

To evaluate clustering quality in the absence of ground-truth labels, we use the **Calinski-Harabasz Index (CHI)**. This metric assesses the ratio of between-cluster dispersion to within-cluster cohesion. Higher CHI scores indicate that clusters are compact and well-separated, suggesting high clustering quality.

We exclude noise points (i.e., articles labeled -1) from CHI calculations, as these represent singleton or weakly connected articles that do not belong to any meaningful cluster.

6.5 Results and Comparison

The clustering results, evaluated using the Calinski-Harabasz Index, are summarized in Table 1. The scores reflect the performance of each algorithm on pooled ColBERT vectors across multiple days of Thai news articles.

Table 1: Clustering Performance Comparison (CHI Score)

Method	**Calinski-Harabasz Index ↑**
DBSCAN	**9.64**
HDBSCAN	8.76
OPTICS	6.98

Among all methods, **DBSCAN** achieves the highest CHI score, indicating it forms the most compact and well-separated clusters under our setup. While HDBSCAN and OPTICS are competitive, they fall slightly short in this dataset's context.

6.6 Final Model Selection

We select **DBSCAN** as the final clustering algorithm for our news processing pipeline. It consistently performs well in our evaluation and is well-suited for real-world news scenarios where topic density varies and cluster counts are not known a priori. Combined with BGE-M3's token-level embeddings, DBSCAN offers a robust approach to unsupervised event clustering in Thai-language news.

7 LLM-Based Summarization of Clustered News

After clustering semantically similar news articles, we employ a large language model (LLM) to generate human-readable summaries and timelines for each cluster. This step enhances interpretability and enables users to quickly grasp the essence of trending news topics without reading individual articles.

7.1 Motivation

Clustering alone groups related articles together but lacks readable summaries that convey the key points and event sequences. To bridge this gap, we use the `scb10x/typhoon2.1-gemma3-4b` [7] model—an instruction-tuned Thai-capable LLM—to produce structured summaries with temporal ordering of events.

7.2 Summarization Pipeline

For each cluster (excluding noise clusters, i.e., cluster ID = -1), all articles' content are concatenated into a single prompt, which is then passed to the LLM. The prompt instructs the model to:

- Provide a concise summary of the news cluster in Thai
- Reconstruct the timeline of events, including dates and sequence where available

7.3 LLM Configuration

We utilize `Typhoon2.1-Gemma3-4B`, an instruction-tuned Thai large language model with 4 billion parameters, 128K context length, and function-calling capabilities. The model is based on `Gemma3 4B` and fine-tuned for Thai-language tasks, making it well-suited for summarizing local news content.

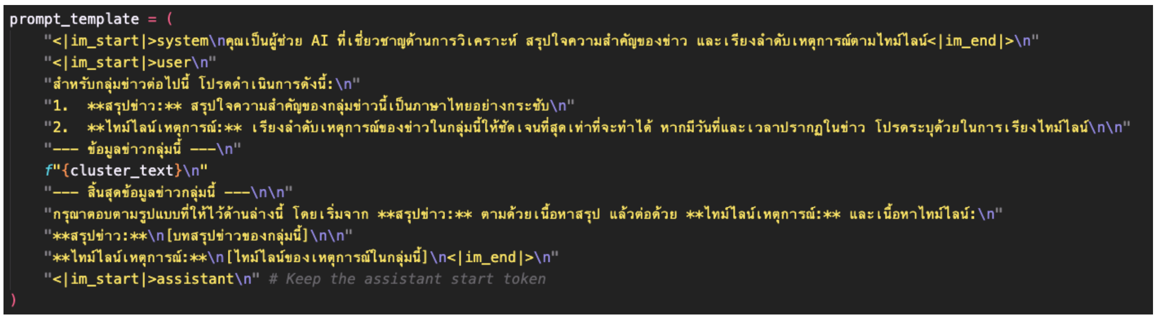
It is accessed via the Hugging Face `transformers` pipeline with the following configurations:

- Mixed precision enabled with `torch.bfloat16`
- Sampling with temperature = 0.6, top-p = 0.9, and top-k = 40
- Prompt formatting adheres to the `<|im_start|>` and `<|im_end|>` format to align with instruction-following behavior

7.4 Prompt Design

The prompt is designed to elicit a structured response from the LLM, following the `<|im_start|>` and `<|im_end|>` conversational formatting. It contains both system-level instructions and a detailed user request in Thai, instructing the model to summarize and construct a timeline of the given news cluster.

The screenshot below shows the actual prompt template passed to the LLM during summarization:



```python
prompt_template = (
    "<|im_start|>system\nคุณเป็นผู้ช่วย AI ที่เชี่ยวชาญด้านการวิเคราะห์ สรุปใจความสำคัญของข่าว และเรียงลำดับเหตุการณ์ตามไทม์ไลน์<|im_end|>\n"
    "<|im_start|>user\n"
    "สำหรับกลุ่มข่าวต่อไปนี้ โปรดดำเนินการดังนี้:\n"
    "1.  **สรุปข่าว:** สรุปใจความสำคัญของกลุ่มข่าวนี้เป็นภาษาไทยอย่างกระชับ\n"
    "2.  **ไทม์ไลน์เหตุการณ์:** เรียงลำดับเหตุการณ์ของข่าวในกลุ่มนี้ให้ชัดเจนที่สุดเท่าที่จะทำได้ หากมีวันที่และเวลาปรากฏในข่าว โปรดระบุด้วยในการเรียงไทม์ไลน์\n\n"
    "--- ข้อมูลข่าวกลุ่มนี้ ---\n"
    f"{cluster_text}\n"
    "--- สิ้นสุดข้อมูลข่าวกลุ่มนี้ ---\n\n"
    "กรุณาตอบตามรูปแบบที่ให้ไว้ด้านล่างนี้ โดยเริ่มจาก **สรุปข่าว:** ตามด้วยเนื้อหาสรุป แล้วต่อด้วย **ไทม์ไลน์เหตุการณ์:** และเนื้อหาไทม์ไลน์:\n"
    "**สรุปข่าว:**\n[บทสรุปข่าวของกลุ่มนี้]\n\n"
    "**ไทม์ไลน์เหตุการณ์:**\n[ไทม์ไลน์ของเหตุการณ์ในกลุ่มนี้]\n<|im_end|>\n"
    "<|im_start|>assistant\n" # Keep the assistant start token
)
```

Figure 6: Prompt template used to instruct the Thai LLM for summarization

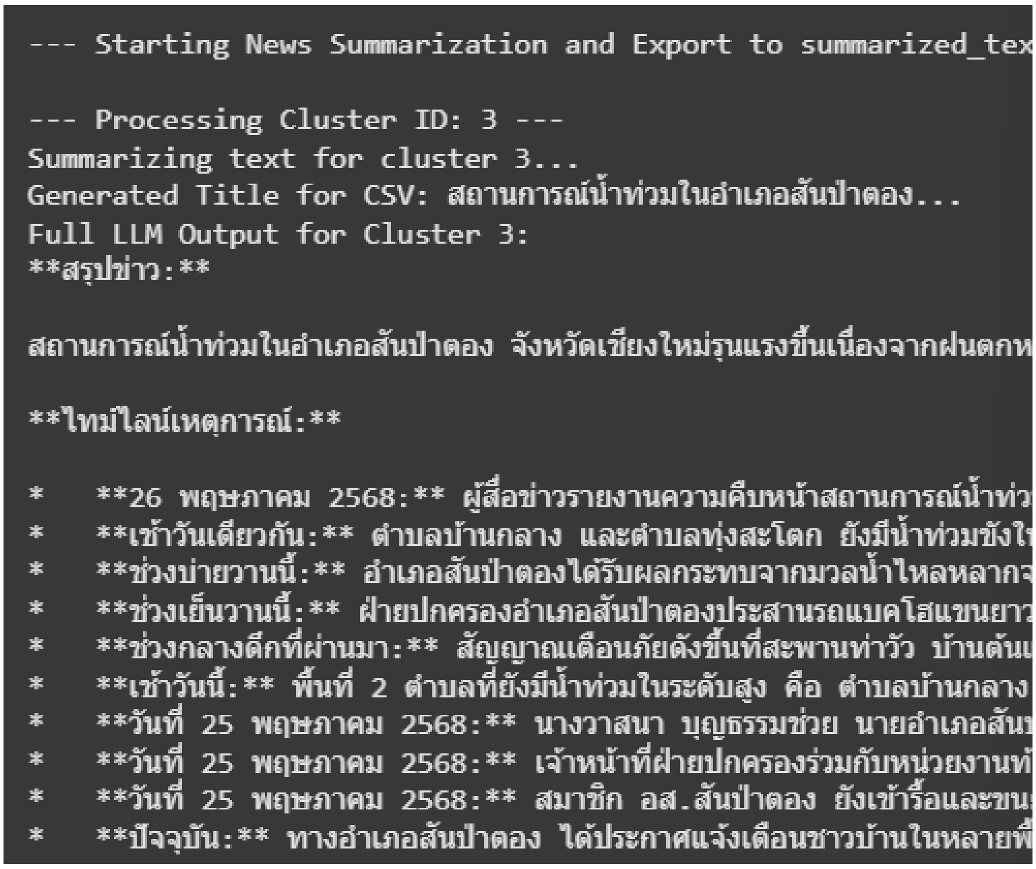
7.5 Output and Postprocessing

The model's output contains both the **summary** and a **timeline of events**. A parser is used to extract these parts, and a dynamic title is generated from the first sentence of the summary. Each record is saved into a CSV with the following fields:

- `title`: Auto-generated short headline
- `cluster_id`: Corresponding cluster ID
- `summarized_news`: Full summary and timeline content
- `date`: Processing date

7.6 Example

For a cluster discussing a political event, the LLM produced the following summary and timeline. The screenshot below shows the actual model output rendered in the application interface.



```
--- Starting News Summarization and Export to summarized_tex

--- Processing Cluster ID: 3 ---
Summarizing text for cluster 3...
Generated Title for CSV: สถานการณ์น้ำท่วมในอำเภอสันป่าตอง...
Full LLM Output for Cluster 3:
**สรุปข่าว:**

สถานการณ์น้ำท่วมในอำเภอสันป่าตอง จังหวัดเชียงใหม่รุนแรงขึ้นเนื่องจากฝนตกห

**ไทม์ไลน์เหตุการณ์:**

*   **26 พฤษภาคม 2568:** ผู้สื่อข่าวรายงานความคืบหน้าสถานการณ์น้ำท่ว
*   **เช้าวันเดียวกัน:** ตำบลบ้านกลาง และตำบลทุ่งสะโตก ยังมีน้ำท่วมขัง
*   **ช่วงบ่ายวานนี้:** อำเภอสันป่าตองได้รับผลกระทบจากมวลน้ำไหลหลาก
*   **ช่วงเย็นวานนี้:** ฝ่ายปกครองอำเภอสันป่าตองประสานรถแบคโฮแขนยาว
*   **ช่วงกลางดึกที่ผ่านมา:** สัญญาณเตือนภัยดังขึ้นที่สะพานท่าวัว บ้านต้น
*   **เช้าวันนี้:** พื้นที่ 2 ตำบลที่ยังมีน้ำท่วมในระดับสูง คือ ตำบลบ้านกลาง
*   **วันที่ 25 พฤษภาคม 2568:** นางวาสนา บุญธรรมช่วย นายอำเภอสัน
*   **วันที่ 25 พฤษภาคม 2568:** เจ้าหน้าที่ฝ่ายปกครองร่วมกับหน่วยงาน
*   **วันที่ 25 พฤษภาคม 2568:** สมาชิก อส.สันป่าตอง ยังเข้ารื้อและขน
*   **ปัจจุบัน:** ทางอำเภอสันป่าตอง ได้ประกาศแจ้งเตือนชาวบ้านในหลาย
```

Figure 7: Example Thai news summary generated by Typhoon2.1-Gemma3-4B (Cluster: Flood Event)

7.7 Discussion

This LLM-driven summarization provides two key benefits:

1. It distills clustered news into a concise, readable form for end-users
2. It transforms unsupervised clustering outputs into structured knowledge with semantic and temporal coherence

8 Containerization & Deployment

Containerization and deployment are core to the news processing system's MLOps pipeline, enabling reliable, scalable, and reproducible services. This section outlines how the system leverages Docker Compose to orchestrate the FastAPI application, Apache Airflow, and supporting infrastructure.

8.1 Dockerization of Components

Dockerization of Components All major components—Apache Airflow services (webserver, scheduler, worker, and triggerer), supporting scripts, and the backend FastAPI application—are containerized using Docker. Each service is defined with environment variables in a shared `.env` file, ensuring consistency across environments.

Airflow is deployed using the official `apache/airflow:2.10.5-python3.10` Docker image. DAGs, plugins, logs, and other shared resources are mounted via volumes:

```
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
  - ./plugins:/opt/airflow/plugins
  - ./lib:/opt/airflow/lib
```

This volume-based structure allows for dynamic code updates without rebuilding containers. Health checks are configured for all services to ensure system stability.

8.2 API Deployment

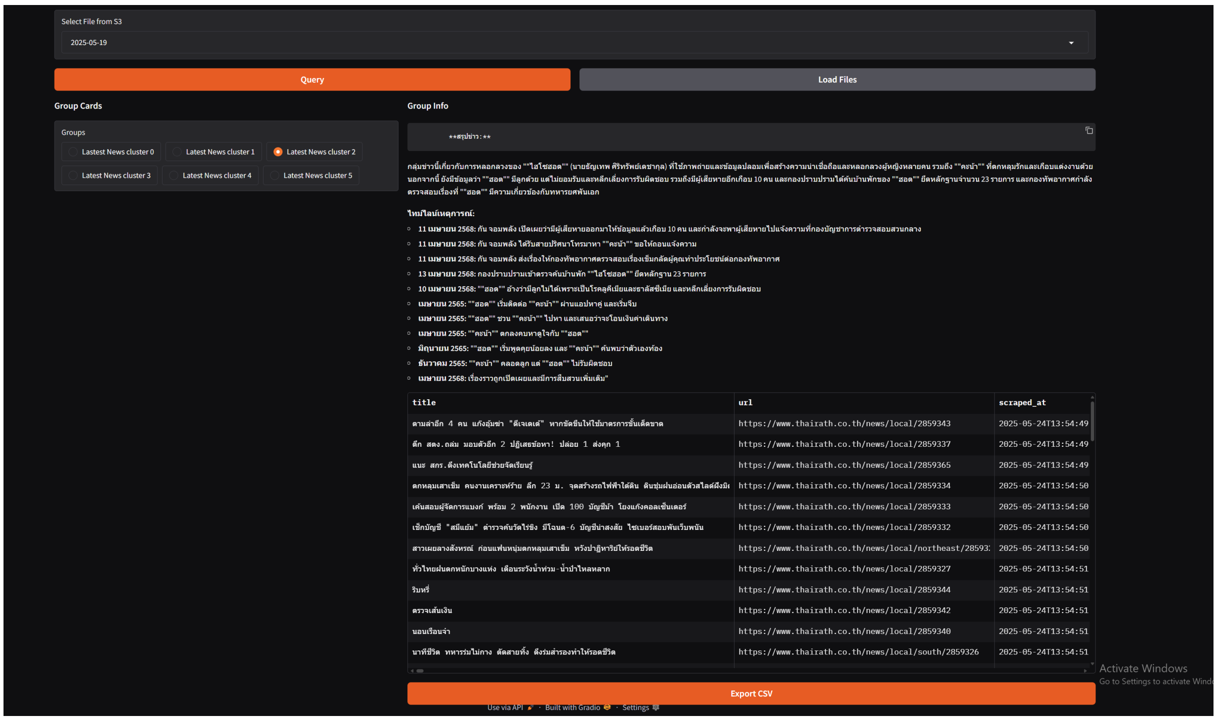
Although the Docker Compose file primarily focuses on Airflow services, the FastAPI application is assumed to be deployed in a similar manner—containerized using its own `Dockerfile`, exposed through a designated port, and configured via a `.env` file. In a production environment, the FastAPI application, being stateless, can be horizontally scaled by increasing the number of running containers.

For certain API endpoints, we integrate `Redis` to enable fast retrieval of frequently accessed data, such as news from the previous week or the past two to three weeks.

8.3 Orchestration with Docker Compose and Airflow

For local development and small-scale deployments, Docker Compose is used to orchestrate multiple containers, including Airflow, the API, and supporting services such as a PostgreSQL database for Airflow metadata. Docker Compose files define service dependencies, network configurations, and persistent volumes, making it easy to spin up the entire stack with a single command.

In production, orchestration can be extended to Kubernetes (K8s) for advanced scheduling, auto-scaling, and fault tolerance. Airflow can be deployed as a set of Kubernetes pods,



**สรุปข่าว : **

กลุ่มข่าวนี้เกี่ยวกับการหลอกลวงของ ""ไฮโซฮอต"" (นายธัญเทพ ศิริทรัพย์เดชากุล) ที่ใช้ภาพถ่ายและข้อมูลปลอมเพื่อสร้างความน่าเชื่อถือและหลอกลวงผู้หญิงหลายคน รวมถึง ""คะน้า"" ที่ตกหลุมรักและเกือบแต่งงานด้วย นอกจากนี้ ยังมีข้อมูลว่า ""ฮอต"" มีลูกด้วย แต่ไม่ยอมรับและหลีกเลี่ยงการรับผิดชอบ รวมถึงมีผู้เสียหายอีกเกือบ 10 คน และกองปราบปรามได้ค้นบ้านพักของ ""ฮอต"" ยึดหลักฐานจำนวน 23 รายการ และกองทัพอากาศกำลังตรวจสอบเรื่องที่ ""ฮอต"" มีความเกี่ยวข้องกับทหารยศพันเอก

ไทม์ไลน์เหตุการณ์:

- **11 เมษายน 2568:** กัน จอมพลัง เปิดเผยว่ามีผู้เสียหายออกมาให้ข้อมูลแล้วเกือบ 10 คน และกำลังจะพาผู้เสียหายไปแจ้งความที่กองบัญชาการตำรวจสอบสวนกลาง
- **11 เมษายน 2568:** กัน จอมพลัง ได้รับสายปริศนาโทรมาหา ""คะน้า"" ขอให้ถอนแจ้งความ
- **11 เมษายน 2568:** กัน จอมพลัง ส่งเรื่องให้กองทัพอากาศตรวจสอบเรื่องเข็มกลัดผู้คุณทำประโยชน์ต่อกองทัพอากาศ
- **13 เมษายน 2568:** กองปราบปรามเข้าตรวจค้นบ้านพัก ""ไฮโซฮอต"" ยึดหลักฐาน 23 รายการ
- **10 เมษายน 2568:** ""ฮอต"" อ้างว่ามีลูกไม่ได้เพราะเป็นโรคลูคีเมียและธาลัสซีเมีย และหลีกเลี่ยงการรับผิดชอบ
- **เมษายน 2565:** ""ฮอต"" เริ่มติดต่อ ""คะน้า"" ผ่านแอปหาคู่ และเริ่มจีบ
- **เมษายน 2565:** ""ฮอต"" ชวน ""คะน้า"" ไปหา และเสนอว่าจะโอนเงินค่าเดินทาง
- **เมษายน 2565:** ""คะน้า"" ตกลงคบหาดูใจกับ ""ฮอต""
- **มิถุนายน 2565:** ""ฮอต"" เริ่มพูดคุยน้อยลง และ ""คะน้า"" ค้นพบว่าตัวเองท้อง
- **ธันวาคม 2565:** ""คะน้า"" คลอดลูก แต่ ""ฮอต"" ไม่รับผิดชอบ
- **เมษายน 2568:** เรื่องราวถูกเปิดเผยและมีการสืบสวนเพิ่มเติม"

title	url	scraped_at
ตามล่าอีก 4 คน แก๊งอุ้มฆ่า "ดีเจเต้" หากขัดขืนให้ใช้มาตรการขั้นเด็ดขาด	https://www.thairath.co.th/news/local/2859343	2025-05-24T13:54:49
ตึก สตง.ถล่ม มอบตัวอีก 2 ปฏิเสธข้อหา! ปล่อย 1 ส่งคุก 1	https://www.thairath.co.th/news/local/2859337	2025-05-24T13:54:49
แนะ สกร.ดึงเทคโนโลยีช่วยจัดเรียนรู้	https://www.thairath.co.th/news/local/2859365	2025-05-24T13:54:49
ตกหลุมเสาเข็ม คนงานเคราะห์ร้าย ลึก 23 ม. จุดสร้างรถไฟฟ้าใต้ดิน ดินชุ่มฝนอ่อนตัวสไลด์ฝังมิ	https://www.thairath.co.th/news/local/2859334	2025-05-24T13:54:50
เค้นสอบผู้จัดการแบงก์ พร้อม 2 พนักงาน เปิด 100 บัญชีม้า โยงแก๊งคอลเซ็นเตอร์	https://www.thairath.co.th/news/local/2859333	2025-05-24T13:54:50
เช็กบัญชี "สมีแย้ม" ตำรวจค้นวัดไร่ขิง มีโฉนด-6 บัญชีน่าสงสัย ไซเบอร์สอบพันเว็บพนัน	https://www.thairath.co.th/news/local/2859332	2025-05-24T13:54:50
สาวเผยลางสังหรณ์ ก่อนแฟนหนุ่มตกหลุมเสาเข็ม หวังปาฏิหาริย์ให้รอดชีวิต	https://www.thairath.co.th/news/local/northeast/28593	2025-05-24T13:54:50
ทั่วไทยฝนตกหนักบางแห่ง เตือนระวังน้ำท่วม-น้ำป่าไหลหลาก	https://www.thairath.co.th/news/local/2859327	2025-05-24T13:54:51
ริบหรี่	https://www.thairath.co.th/news/local/2859344	2025-05-24T13:54:51
ตรวจเส้นเงิน	https://www.thairath.co.th/news/local/2859342	2025-05-24T13:54:51
นอนเรือนจำ	https://www.thairath.co.th/news/local/2859340	2025-05-24T13:54:51
นาทีชีวิต ทหารร่มไม่กาง ตัดสายทิ้ง ดึงร่มสำรองทำให้รอดชีวิต	https://www.thairath.co.th/news/local/south/2859326	2025-05-24T13:54:51

Figure 8: Gradio-based user interface for API testing

with the scheduler, workers, and webserver running as independent, scalable services. This enables the pipeline to handle large-scale workloads and recover gracefully from failures.

8.4 User Interface Deployment

To quickly test our API service, we designed and implemented a demo application using `Gradio`, which runs on port 7860.

As shown in Figure 8, the interface includes a dropdown menu that lists available weeks for which summaries are stored in the database. Below the dropdown, on the left side of the interface, a radio button group displays the topics for the selected week. When a user selects a topic, the application displays a summary of that topic along with a downloadable `.csv` file containing the raw data for the selected week.

8.5 Cloud Integration and Security

The deployed containers interact with cloud services such as Amazon S3 for data storage of relational database and datalake. IAM roles and policies are configured to grant least-privilege access to each service, ensuring data security and compliance. Secrets and credentials are managed using environment variables or secret management tools (e.g., AWS Secrets Manager), never hard-coded in the codebase or Docker images.

8.6 Monitoring and Logging

All containers are equipped with logging and monitoring agents. Logs and metrics from the FastAPI services are collected by `Prometheus` and visualized using `Grafana`, as illustrated in Figure 9.

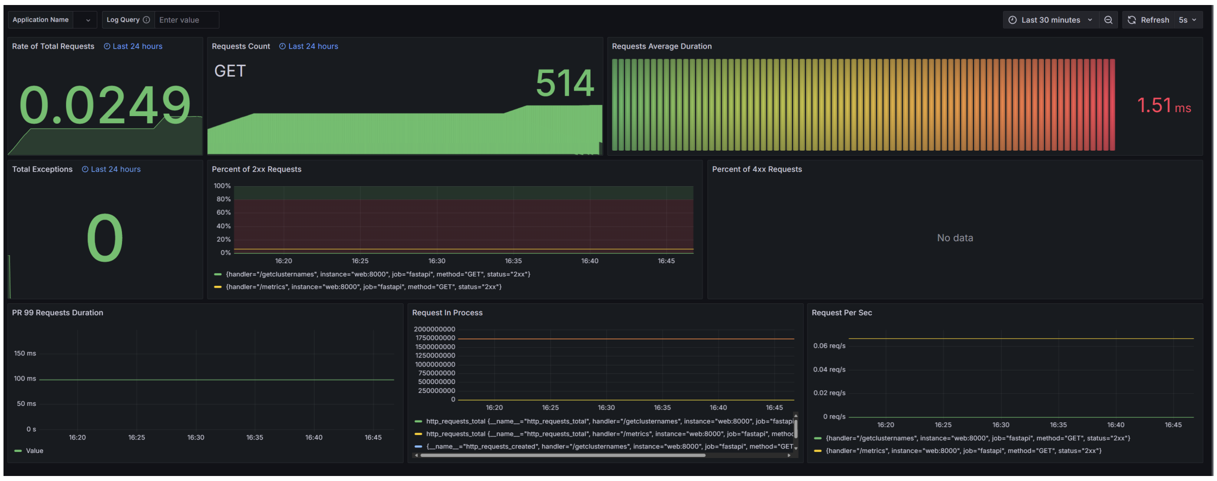


Figure 9: Grafana UI displaying logs and metrics for FastAPI services

The monitoring dashboard displays the number of responses by type for each endpoint, as well as the processing rate and duration of API requests for detect any anomaly of every api endpoints.

8.7 Summary

Containerization and automated deployment form the backbone of the MLOps pipeline, enabling reliable, scalable, and secure operation of the news processing system. By leveraging Docker, orchestration tools, the project achieves high availability, rapid iteration, and robust integration with cloud services, ensuring that models and APIs are always up-to-date and production-ready.

9 Model Monitoring & Retraining

Model monitoring and retraining are essential components of a robust MLOps pipeline, ensuring that deployed models maintain high performance and adapt to changes in data distributions over time. This section outlines the strategies, metrics, and automation mechanisms used to monitor model quality, trigger retraining, and manage the lifecycle of machine learning models in production.

9.1 Continuous Monitoring of Model Performance

Once the clustering model is deployed, its performance is continuously monitored using both quantitative metrics and qualitative inspection. For this task, the primary metric employed is the Calinski-Harabasz Index (CHI), which evaluates the compactness and separation of clusters. Additional metrics, such as silhouette score, cluster purity, and the number of noise points (outliers), are also logged. These metrics are computed periodically on both recent and historical data to detect phenomena such as data drift, concept drift, or a degradation in model performance.

A dedicated Airflow DAG is scheduled to run regularly (e.g., daily or weekly). It ingests fresh data from cloud storage (e.g., Amazon S3), applies the current production model, and calculates evaluation metrics. The results are stored and visualized in a real-time dashboard system (e.g., Grafana), enabling data scientists and engineers to inspect model behavior. Alerts are automatically triggered if any metric falls below predefined thresholds, signaling potential issues that may warrant retraining.

9.2 Integration with the Pipeline

The monitoring DAG is tightly coupled with both the inference and retraining pipelines. If performance degradation is detected, the system automatically triggers the retraining DAG. This seamless integration enables a fully automated response to changing data distributions without requiring manual intervention. The retraining pipeline, also orchestrated by Airflow, shares access to common data repositories, configurations, and the MLflow Model Registry.

9.3 Retraining Triggers and Automation

Retraining is initiated under the following conditions:

- A sustained drop in Calinski-Harabasz Index or other key metrics over multiple runs
- Evidence of data drift, determined by changes in feature distributions or cluster assignments
- Manual triggers from data scientists due to business needs or external events

Once triggered, the pipeline loads the latest dataset, performs preprocessing, and retrains the clustering and summarization models. Hyperparameter tuning is optionally conducted using techniques such as grid search or Bayesian optimization. The new model is then evaluated against a hold-out set and compared with the existing production model.

9.4 Evaluation and Model Promotion

If the retrained model outperforms the current production model based on CHI and other qualitative checks, it is promoted to production via the MLflow Model Registry. The registry stores metadata, performance scores, and version history, enabling seamless rollbacks if the new model fails in production. If the retrained model does not meet the required criteria, the current model is retained, and logs are archived for future analysis.

9.5 Automation and Auditing

All monitoring, retraining, and promotion actions are fully automated and logged. Airflow ensures an auditable trail of all operations, including parameter settings, metric values, and model transitions. This transparency supports debugging, compliance requirements, and continuous improvement of the overall system.

9.6 Retraining Implementation: Contrastive Fine-Tuning of Sentence Embeddings

The retraining process is based on contrastive learning, using the Sentence Transformers framework to fine-tune the multilingual embedding model (`BAAI/bge-m3`) on Thai news data. The objective is to align the model's semantic space more closely with the specific language structure and reporting style found in Thai news articles.

Data Preparation Triplet data is constructed by identifying news article pairs that share a high degree of tag overlap, suggesting they report on the same topic. Tags that are too general are excluded. Human verification is applied to ensure semantic relevance of positive pairs. For each positive pair, one or more negative samples are selected such that the third article has disjoint tags, implying semantic dissimilarity. The final dataset consists of triplets `(anchor, positive, negative)` and is split into training and evaluation sets (80/20), then converted into HuggingFace Dataset format for efficient processing.

Model Initialization The base model is `BAAI/bge-m3`, a state-of-the-art multilingual sentence embedding model. It is loaded via the `SentenceTransformer` class, which offers support for both training and inference.

Training Objective The retraining uses the `MultipleNegativesRankingLoss`, a contrastive loss function designed for sentence similarity tasks. This loss encourages the anchor and positive samples to be closer in embedding space than the anchor and negative samples, enhancing semantic clustering.

Training Configuration Training is configured with the following hyperparameters:

- **Epochs:** 50
- **Batch Size:** 32
- **Warmup Ratio:** 0.1 for smoother learning rate scheduling
- **Mixed Precision:** Enabled (FP16) for speed and memory efficiency
- **Batch Sampler:** No duplicate triplets per batch to optimize contrastive learning
- **Evaluation:** Performed at the end of each epoch with checkpointing based on performance

Model Evaluation and Registry Integration Upon training completion, the model is evaluated on the test set using both CHI and manual inspection of semantic coherence. If the model shows improved clustering performance, it is versioned and registered into MLflow, replacing the previous production version. The integration ensures that inference and monitoring pipelines always reference the latest validated model.

By embedding contrastive fine-tuning into the automated MLOps pipeline, the system maintains high semantic clustering accuracy even as language patterns and news topics evolve. This continuous learning capability enhances the robustness of downstream tasks such as news summarization and topic tracking.

10 Model Evaluation

10.1 Clustering-Based Embedding Evaluation

To evaluate the quality of the embedding model used for clustering news articles, we employ the **Calinski-Harabasz (CHI) score** as the primary metric. The CHI score measures the ratio between the within-cluster dispersion and between-cluster dispersion, and it is commonly used to assess the separation of clusters. A higher CHI score indicates well-separated and compact clusters, implying that the embedding captures the semantic structure of the news articles effectively.

10.2 Score Monitoring and Drift Detection

We maintain a historical log of CHI scores for each model version using the MLflow tracking system. When a new embedding model is registered, we automatically retrieve the CHI scores of the two most recent versions and compute the difference:

$$\Delta_{\text{CHI}} = \text{CHI}_{\text{current}} - \text{CHI}_{\text{previous}}$$

If $\Delta_{\text{CHI}} < 0$, it suggests a drop in clustering performance. To assess statistical significance, we compare the current score against a dynamic threshold based on historical statistics:

$$\text{Threshold} = \mu - 2\sigma$$

Where μ and σ are the mean and standard deviation of all CH scores logged for the current experiment. If the current CHI score falls below this threshold, we trigger a model retraining pipeline.

10.3 Decision Rule for Model Replacement

We define the following rule to decide whether to replace the current embedding model:

- **OK (No Action):** $\Delta_{\text{CHI}} \geq 0$ or current CHI score $\geq \mu - 2\sigma$
- **Retrain Needed:** $\Delta_{\text{CHI}} < 0$ **and** current CHI score $< \mu - 2\sigma$

This approach provides a statistically grounded mechanism to detect performance degradation or data drift and ensures that only high-quality embedding models are used in the downstream news summarization and clustering pipeline.

10.4 Automation via Model Registry

The entire evaluation logic is integrated with MLflow's model registry. When a new model is registered:

1. It is evaluated using the CHI score.
2. If the score significantly degrades compared to historical models, the system logs the event and triggers a retraining workflow.
3. If acceptable, the model is promoted to production and used in subsequent data ingestion rounds.

10.5 Implementation Summary

The evaluation function `check_embedding()` is implemented to:

- Retrieve past runs with the same embedding model
- Compute statistical thresholds
- Decide whether to trigger finetuning

This ensures continuous monitoring of embedding quality and supports robust model lifecycle management within the MLOps pipeline.

11 Conclusion

This work presents an end-to-end MLOps pipeline for automated news processing, emphasizing the seamless integration of data ingestion, semantic embedding, clustering, summarization, monitoring, and retraining. The pipeline is designed for robustness, scalability, and maintainability, supporting real-world deployment needs for continuously evolving data sources such as news media.

At the core of the system is a tightly orchestrated workflow built on Airflow, enabling scheduled execution of scraping, embedding generation, clustering, and summarization modules. The pipeline includes automated evaluation using intrinsic metrics (e.g., Calinski-Harabasz score) and MLflow-based model tracking to ensure reproducibility and governance. Model monitoring components detect performance degradation and trigger fine-tuning routines automatically, ensuring sustained model quality over time.

This MLOps framework demonstrates how unsupervised machine learning workflows can be deployed and maintained at scale with minimal manual effort. By automatically clustering and summarizing large volumes of unstructured news content, the system streamlines the extraction of key information—enabling users to quickly grasp emerging events and developments without wading through individual articles, thereby saving valuable time in daily news monitoring and analysis

12 Future Work

Several technical enhancements are planned to further improve the robustness, scalability, and usability of the system.

12.1 Automated Fine-Tuning and Model Promotion

Currently, model fine-tuning is partially manual and integrated within the MLOps pipeline. Future iterations aim to fully automate this process—enabling the system to not only detect performance degradation but also trigger fine-tuning and promote improved models autonomously. However, achieving this requires resolving current infrastructure limitations. Due to insufficient GPU resources on the primary server, model retraining must be offloaded to external GPU services, which complicates automation and introduces additional latency and cost. Streamlining this integration will be a key focus moving forward.

12.2 API Deployment on Scalable Cloud Infrastructure

The current deployment of APIs using tools such as Ngrok is suitable for prototyping but not optimal for production. Migrating the system to a stable, cloud-native infrastructure (e.g., AWS, GCP, or Azure) will improve latency, reliability, and security. It will also allow for seamless scaling and better integration with monitoring and CI/CD tools.

12.3 Frontend Modernization and UX Improvements

The current user interface, built with Gradio, serves as a functional demo platform. However, a future release will transition to a frontend built with modern JavaScript frameworks such as React or Vue.js. This upgrade will enable better UX/UI design, improve responsiveness, and support scalable user interactions, making the system more usable for a wider range of stakeholders.

References

[1] Apache Software Foundation. *Apache Airflow*. `https://airflow.apache.org/`. Version 2.0+. 2019.

[2] MLflow Developers. *MLflow: Open Source Platform for the Machine Learning Lifecycle*. Accessed: 2025-05-28. 2024. URL: `https://github.com/mlflow/mlflow`.

[3] Armin Ronacher. *Flask: A lightweight WSGI web application framework*. `https://flask.palletsprojects.com/`. Accessed: 2025-05-28. 2010.

[4] Google Research. *Google Colaboratory*. `https://colab.research.google.com/`. Accessed: 2025-05-28. 2018.

[5] Alan Shreve. *ngrok: Secure introspectable tunnels to localhost*. `https://ngrok.com/`. Accessed: 2025-05-28. 2015.

[6] Jianlv Chen et al. *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. 2024. arXiv: `2402.03216 [cs.CL]`.

[7] Kunat Pipatanakul et al. *Typhoon 2: A Family of Open Text and Multimodal Thai Large Language Models*. 2024. arXiv: `2412.13702 [cs.CL]`. URL: `https://arxiv.org/abs/2412.13702`.