Data Ingestion Pipeline

# Status of Document

* Updated 15 January to reflect current design.
* Updated by Gau Meng Yew on 4 January 2024, as per discussion points on 3 January 2024
  + Kafka is considered overkill, and it is sufficient to use Docker Compose. An additional consideration is to use the Docker Image instead of the Python packages.
  + Splitting processing and IO into 2 separate layers to ensure data storage consistency.
* Updated by Gau Meng Yew on 18 December 2024, reflecting the communication protocol design and management of data flow.
* Proposed by Gau Meng Yew, 16 December 2024 on a initial design of the data ingestion pipeline

# 

# Abstract

The data transformation process requires multiple reads/writes from the internal MongoDB database. Due to the volume of data pulled on a daily basis, there is a need to be able to re-run individual steps without compromising data integrity, and remain independent of other processes in the pipeline. This is completed by separating I/O operations from processing operations, ensuring that regardless of the status of processing steps, if any step completes successfully, it will be written into the DB.

# 

# Introduction

This document provides an overview of the initial design of the data ingestion pipeline starting from pulling data from the NEWS API, transforming the data in stages while writing to the database and providing eventual ready-to-use data for the LLM.

# 

# Layers

The pipeline is split into 2 layers, and triggered by a singular event.

## Data Ingestion and Processing Layer

At this level, the system is designed to be able to integrate different data sources easily, built on a general design. As the raw data may contain imagery and video data, different sources will write to different collections. Computational efficiency is achieved via multithreading as processing steps are CPU/GPU intensive, demanding maximisation of computational resources.

As this layer is the entry component of the pipeline, it requires triggering on a workflow scheduler. We utilise Airflow for its scalability and vast documentation.

## Data I/O Layer

Each I/O operation is atomic, ensuring no corrupted or lossy data is written. It occurs after all processing steps are completed, and is background based. While data is upserted, it is not expected to have duplicates in any cases. This logic is maintained in the event of reruns.

# Communication Protocol

## Separation of services

The data pipeline are separated into 2 services:

1. Data Ingestion
2. Named Entity Recognition
   1. Text Processor
   2. Sentence Split
   3. Similarity Calculation

The data ingestion service is set to run with an Airflow scheduler, and triggers are preset timings. This separation allows the service to solely focus on database write operations, and enables future seamless integration.

The Named Entity Recognition service comprises 3 microservices, and the flow is specified in the entrant file. Upon completion, the results is stored in memory for I/O purposes