République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ SAAD DAHLEB - BLIDA 1

Faculté de Science

Département d'informatique

Mémoire de Project de Fin d'Étude Pour l'obtention du diplôme de Master en Informatique

Option : *Traitement automatique du langage naturel*

Thème

Suivis les Tendances en temps réel avec Apache Kafka

Présenté par :

Zarat Fethi

Encadré par :

Dr.Kameche Abdallah Hicham

Année Universitaire : 2024 / 2025

Remerciements

Avant tout, je tiens à exprimer ma profonde gratitude à Dieu, pour m'avoir donné la force, la patience et la persévérance nécessaires à l'accomplissement de ce travail.

Je remercie sincèrement mon encadrant, Monsieur Kameche Abdallah Hicham pour son accompagnement, ses conseils précieux et son soutien tout au long de ce projet, ainsi que durant l'ensemble de mes années d'études.

Je tiens également à exprimer toute ma gratitude à Madame Mezzi Melyara pour sa gentillesse, son soutien et l'inspiration qu'elle m'a donnés tout au long de mes années d'études. Sa bienveillance et ses encouragements m'ont profondément touché et ont été pour moi une véritable source de motivation. Je lui en suis sincèrement reconnaissant.

Je remercie les membres du jury pour l'honneur qu'ils me font en acceptant d'évaluer ce travail, ainsi que pour l'attention qu'ils lui accorderont.

Enfin je souhaite adresser tous mes remerciements aux personnes qui m'ont apporté leur aide et qui ont contribué de près et de loin à l'élaboration de cette mémoire.

Dédicaces

Je dédie ce travail à ma précieuse maman et à mon très cher papa ; sans leur amour, leurs sacrifices et leur soutien, je ne serais jamais arrivé jusqu'ici, alors je tiens à les remercier précisément.

À mes deux petits frères, Abdelmalek et Anes, et à ma petite sœur Layla

A Monsieur Kameche Abdallah et madame Mezzi Melyara

À mes deux grands-mères, à mon grand-père, ainsi qu'à mes tantes Nadia, Amel et Hadjira

A tous mes amis de kolea : younes , Karim ,Lokmane , malik , Mehdi , Nemri , jojo , abdennour , tazrout , napa , djamel , zaki , yacine , adel , amine , sofitcho

Résumé

Ce projet de fin d'études aborde le défi de l'analyse en temps réel des tendances discursives sur Internet, en combinant des techniques de traitement automatique des langues (TAL) et d'architecture des données en flux. L'objectif principal était de concevoir et d'implémenter un système complet capable de collecter, traiter et visualiser en continu des contenus textuels issus de la plateforme Reddit et de flux d'actualités via l'API News. Pour ce faire, une architecture technique robuste a été mise en œuvre. Le pipeline de traitement s'appuie sur Apache Kafka pour l'ingestion et la gestion des flux de données, et sur MongoDB pour un stockage flexible et non structuré. Le cœur de l'analyse repose sur une suite de modèles de TAL, incluant VADER et des modèles de la bibliothèque Transformers pour l'analyse de sentiment, spaCy pour la reconnaissance d'entités nommées. Le résultat tangible de ce projet est un tableau de bord interactif, développé avec Flask et Dash, qui restitue les analyses de manière synthétique et dynamique.

Mots-clés : Traitement automatique des langues, Kafka, Analyse de sentiment, Extraction d’entités nommées, Visualisation, Données en temps réel, Dash, Flask, MongoDB.

Abstract

This final year project addresses the challenge of real-time analysis of discursive trends on the Internet, combining Natural Language Processing (NLP) techniques with streaming data architectures. The main objective was to design and implement a complete system capable of continuously collecting, processing, and visualizing textual content from the Reddit platform and news feeds via the News API. To achieve this, a robust technical architecture was implemented. The processing pipeline relies on Apache Kafka for data stream ingestion and management, and on MongoDB for flexible, unstructured storage. The core of the analysis is based on a suite of NLP models, including VADER and models from the Transformers library for sentiment analysis, spaCy for named entity recognition. The tangible result of this project is an interactive dashboard, developed with Flask and Dash, which delivers the analysis in a synthetic and dynamic way.

Keywords : Natural Language Processing, Kafka, Sentiment Analysis, Named Entity Recognition, Visualization, Real-Time Data, Dash, Flask, MongoDB.

الملخص

يتناول مشروع نهاية الدراسة تحدي تحليل التوجهات الخطابية على الأنترنت في الزمن الحقيقي، وذلك من خلال دمج تقنيات المعالجة الآلية للغة الطبيعية، مع هندسة البيانات المتدفقة. يتمثل الهدف الأساسي في تصميم وتنفيذ نظام متكامل قادر على جمع ومعالجة وعرض المحتوى النصي بشكل مستمر، انطلاقا من منصة "ريديت" ومن خلال واجهة برمجة التطبيقات الخاصة بالأخبار.

لتحقيق ذلك، تم اعتماد بنية تقنية قوية تتكوّن من عدّة مكوّنات متكاملة. يعتمد مسار معالجة البيانات على أباتشي كافكا لإلتقاط وإدارة تدفّقات البيانات، وعلى قاعدة البيانات مونغو دي بي لتخزين المعلومات النصيّة بطريقة مرنة وغير مهيكلة. أما التحليل اللغوي فيرتكز على مجموعة من الأدوات والنماذج المتقدمة في مجال اللغة الطبيعية، من بينها أداة " فيدر" ونماذج "المحوّلات" المستخرجة من مكتبة "هاجينغ فايس" لتحليل المشاعر والانفعالات، ومكتبة "سبايسي" للتعرّف على الكيانات المسماة.

أما النتيجة الملموسة لهذا العمل فهي لوحة تحكم تفاعلية تم تطويرها باستخدام إطار العمل "فلاسْك" بالتكامل مع مكتبة "داش"، وتسمح هذه الواجهة بعرض التحليلات بشكل مبسّط، مرئي، وتفاعلي، ما يوفر للمستخدم رؤية واضحة لحالة النقاشات والاتجاهات الجارية في الزمن الفعلي.

الكلمات المفتاحية: المعالجة الآلية للغة الطبيعية، أباتشي كافكا، تحليل المشاعر، التعرّف على الكيانات المسماة، التصور البياني، البيانات المتدفقة، فلاسْك، داش، مونغو دي بي.

Sommaire

Table des Tableaux

Table des Figures

Liste des acronymes

-**NLP** (Natural Language Processing) : Traitement automatique du langage naturel.

-**TALN** (Traitement Automatique du Langage Naturel) : Terme français équivalent à NLP.

-**NER** (Named Entity Recognition) : Reconnaissance d'entités nommées.

-**POS** (Part-of-Speech) : Étiquetage grammatical des mots (noms, verbes, adjectifs…).

-**CRF** (Conditional Random Fields) : Modèle statistique utilisé pour la reconnaissance de séquences.

-**HMM** (Hidden Markov Model) : Modèle de Markov caché, utilisé en apprentissage automatique séquentiel.

-**LSTM** (Long Short-Term Memory) : Type de réseau neuronal récurrent performant pour les séquences.

-**BERT** (Bidirectional Encoder Representations from Transformers) : Modèle de type Transformer largement utilisé en NLP.

-**RoBERTa** (Robustly Optimized BERT Pretraining Approach) : Variante améliorée de BERT.

-**VADER** (Valence Aware Dictionary for Sentiment Reasoning) : Outil de lexique pour l'analyse de sentiment.

-**API** (Application Programming Interface) : Interface de programmation d'applications.

-**UI** (User Interface) : Interface utilisateur.

-**JSON** (JavaScript Object Notation) : Format de données textuelles léger et lisible.

-**BSON** (Binary JSON) : Version binaire optimisée du format JSON, utilisée par MongoDB.

-**NoSQL** (Not only SQL) : Famille de bases de données non relationnelles.

-**CPU** (Central Processing Unit) : Processeur central d'un ordinateur.

-**RAM** (Random Access Memory) : Mémoire vive d'un système informatique.

-**IO** (Input / Output) : Entrée / sortie des données.

-IoT (Internet of Things) : Réseau d’objets connectés.

-GPU (Graphics Processing Unit) : Processeur graphique, utilisé aussi pour l'entraînement des modèles.

-CSV (Comma-Separated Values) : Format de fichier tabulaire simple.

-UML (Unified Modeling Language) : Langage de modélisation logicielle.

-URL (Uniform Resource Locator) : Adresse d'une ressource sur Internet.

-HTTP (HyperText Transfer Protocol) : Protocole de communication web.

-TCP/IP (Transmission Control Protocol / Internet Protocol) : Ensemble de protocoles réseau de base.

-JS (JavaScript) : Langage de programmation côté client pour le web.

-DB (Database) : Base de données.

-RT (Real Time) : Temps réel.

Introduction générale

Contexte : explosion des données textuelles (réseaux sociaux, articles, etc.) :

Ces dix dernières années, les données en ligne ont explosé de manière spectaculaire. Chaque jour, ce sont des milliards de messages, d'articles, de posts et les blogs. Cette avalanche de textes reflète une société hyperconnectée, où tout le monde s'exprime, partage et réagit en temps réel.

Les réseaux sociaux jouent un rôle clé dans cette dynamique. Des plateformes comme Twitter, Reddit ou Facebook permettent aux utilisateurs de réagir instantanément à l'actualité, de débattre de sujets brûlants et même de faire émerger des tendances capables d'influencer l'opinion publique ou les décisions politiques. De leur côté, les agrégateurs d'infos et les flux RSS diffusent en continu des articles provenant de sources variées.

Face à ce déluge d'informations, les entreprises, chercheurs et journalistes doivent relever un défi de taille : comment capturer et analyser ces masses de données en temps réel ? Stocker ces textes ne suffit plus. Il faut désormais des outils intelligents capables de traiter l'information à la volée, d'identifier les sujets émergents, d'analyser les opinions exprimées et d'en extraire les éléments clés.

C'est là que le traitement automatique du langage (TAL) et les architectures événementielles entrent en jeu. Grâce à des technologies comme Apache Kafka pour gérer les flux, MongoDB pour stocker efficacement les données, ou encore spaCy et les modèles Transformers pour l'analyse sémantique, il est possible de construire des systèmes performants. Ces outils transforment un torrent de texte brut en informations structurées et exploitables, offrant ainsi une vision claire des tendances en temps réel.

Problématique

C'est dans ce contexte que s'inscrit ce projet de fin d'études. La problématique centrale est la suivante : comment concevoir un système capable d'assurer une analyse sémantique continue, fiable et réactive de flux de données textuelles pour en identifier les tendances émergentes ? Pour y répondre, ce projet explore la synergie entre les architectures orientées événements et les avancées récentes en traitement automatique des langues (TAL).

Objectif du projet

L'objectif de ce travail est donc de concevoir et de mettre en œuvre un système complet de suivi des tendances en temps réel, traitant des données provenant de Reddit et de l'API News. L'architecture proposée s'articule autour d'outils modernes et complémentaires : Apache Kafka assure l'ingestion et la transmission fluide des données en flux, tandis que MongoDB offre un stockage NoSQL flexible et performant pour les résultats enrichis. Le cœur analytique du système repose sur des modèles de TAL robustes : spaCy pour le prétraitement et la reconnaissance d'entités nommées, et une double approche pour l'analyse de sentiment

Organisation du mémoire

Ce mémoire est structuré en quatre parties principales permettant de suivre une progression logique depuis les fondements théoriques jusqu'à l'évaluation concrète du système développé

- **Chapitre 1 : État de l'art et fondements théoriques**
  Explore les concepts clés du traitement de flux de données en temps réel, les approches de l'analyse sémantique, ainsi que les outils et technologies existants tels que Kafka, MongoDB, les modèles de NLP (spaCy, Transformers, VADER), et les APIs (Reddit, NewsAPI).
- **Chapitre 2 : Conception et modélisation du système**
  Décrit l'architecture globale du système, les modules principaux, les choix techniques, ainsi que les modèles de données et le fonctionnement du pipeline de traitement de tendances.
- **Chapitre 3 : Implémentation, expérimentation et résultats**
  Détaille les différentes étapes d'implémentation, les composants développés (producteurs, analyse NLP, tableau de bord), les tests effectués, et les résultats obtenus en termes de performance et d'efficacité.

Chapitre 1 : État de l'art et fondements théoriques

1.1. Introduction

Ce chapitre dresse un panorama des concepts théoriques et technologiques fondamentaux nécessaires à la compréhension du projet. Il aborde d'abord les systèmes de traitement en flux, leurs architectures et leurs différences avec le traitement par lot. Il s'intéresse ensuite à Kafka, en tant que technologie de diffusion centrale, et le compare à d'autres solutions. La seconde partie du chapitre est consacrée au Traitement Automatique du Langage Naturel (TALN), avec un focus sur les tâches clés comme la tokenisation, l'analyse de sentiment et la reconnaissance d'entités nommées (NER). Enfin, il présente les bases NoSQL, notamment MongoDB, et revient sur les systèmes de veille existants, en soulignant leurs limites et les apports de notre approche.

1.2. Les systèmes de traitement des flux de données

Avant d'aborder les détails techniques, il est essentiel de comprendre ce qu'est un système de traitement en flux et quelles sont ses principales caractéristiques.

1.2.1. Définition et caractéristiques d'un système de traitement en flux

À mesure que les technologies numériques évoluent, la manière de traiter les données connaît elle aussi une transformation profonde. L'un des paradigmes majeurs qui a émergé ces dernières années est celui du traitement en flux **(***stream processing*). Contrairement au traitement par lot (*batch processing*), qui consiste à collecter une grande quantité de données pour les analyser de façon périodique, le traitement en flux repose sur une logique continue et réactive, les données sont traitées dès leur apparition, instantanément ou presque. [1]

Ce type de traitement s'avère particulièrement adapté aux environnements où l'information est produite de manière constante et rapide : réseaux sociaux, capteurs intelligents (IoT), transactions bancaires, ou encore plateformes de messagerie. Dans un système de traitement en flux, chaque événement (message, post, signal, etc.) est capté en temps réel, analysé à la volée, et transmis immédiatement aux modules responsables de l'enrichissement ou de la prise de décision. Cela permet non seulement de réduire la latence, mais aussi de réagir immédiatement à des événements critiques ou à des tendances émergentes. [2]

Dans le cadre du présent projet, cette approche est particulièrement pertinente, car elle permet de surveiller en continu les publications issues de Reddit ou des flux d'actualités, et d'y appliquer une analyse automatique du contenu à mesure que les données arrivent.

Voici un tableau récapitulatif des caractéristiques clés d'un système de traitement en flux [1] [3] [4] :

Caractéristique	Description
Temps réel ou quasi réel	Traitement immédiat à la réception des données, avec une latence minimale.
Traitement incrémental	Chaque événement est traité individuellement ou en petits lots, évitant l'attente ou l'accumulation.
Scalabilité horizontale	Capacité à répartir la charge sur plusieurs machines ou instances, pour gérer de grands volumes.
Tolérance aux pannes	Intégration de mécanismes de reprise (re-jeu des messages, journalisation, etc.) pour assurer la fiabilité du traitement.
Persistabilité & découplage	Les messages peuvent être stockés temporairement dans des files (*topics*) pour permettre un traitement asynchrone, modulable et robuste.

Tableau 1 : tableau récapitulatif des caractéristiques clés d'un système de traitement en flux

Ce paradigme de traitement s'inscrit dans ce qu'on appelle une architecture orientée événements (event-driven architecture), dans laquelle les composants du système ne réagissent pas à des requêtes explicites, mais à des signaux ou des changements d'état. Cette logique favorise la modularité, la flexibilité et la rapidité du système. [5]

1.2.2. Comparaison entre traitement par lot et traitement en continu

Pendant longtemps, le traitement des données s'est appuyé sur des modèles dits par lot (*batch processing*), où l'on accumule d'abord les données, puis on les traite à des moments précis, généralement à intervalles réguliers (quotidien, horaire, etc.). Ce paradigme est bien adapté aux systèmes analytiques classiques, comme les rapports financiers ou les analyses de données historiques.

Cependant, dans des contextes où la fraîcheur de l'information est critique, comme le suivi des tendances sur les réseaux sociaux, la détection de fraudes ou le monitoring en temps réel, cette approche devient vite insuffisante. C'est ici que le traitement en flux (*stream processing*) prend tout son sens : il permet de réagir immédiatement à chaque événement, dès sa survenue, sans attendre la fin d'un lot. [6]

Pour bien saisir la différence entre ces deux approches, le tableau ci-dessous en résume les points clés [7]:

Critère	Traitement par lot	Traitement en continu (flux)
Mode de traitement	Données traitées en groupes (périodiquement)	Données traitées à la volée, événement par événement
Latence	Élevée (minutes à heures)	Faible (millisecondes à secondes)
Réactivité	Faible	Très élevée
Cas d'usage typiques	Statistiques, reporting, archivage	Veille, détection d'événements, systèmes temps réel
Outils courants	Hadoop, Hive, Airflow	Kafka, Spark Streaming, Flink
Complexité	Moindre, plus stable	Plus complexe, nécessite une architecture distribué

Tableau 2 : Comparaison entre les 2 approches

Dans le cadre de ce mémoire, le traitement en flux a été privilégié pour sa capacité à détecter et analyser des tendances dès leur apparition sur les plateformes ciblées. Il permet ainsi un système de veille instantanée, bien plus pertinent pour suivre l'évolution rapide des discussions en ligne.

1.2.3. Exemples d'architectures modernes orientées événements

Le traitement en flux repose très souvent sur des architectures orientées événements (*event-driven architectures*), un modèle de conception dans lequel les systèmes ne fonctionnent pas selon une logique de requêtes classiques, mais en réagissant à des événements. Chaque événement correspond à un changement d'état, une action déclenchée par une source de données, comme une nouvelle publication Reddit, un article d'actualité ou une alerte [5].

Ces architectures présentent plusieurs avantages majeurs :

- Elles sont hautement découplées, ce qui signifie que les composants peuvent évoluer indépendamment.
- Elles sont asynchrones, ce qui facilite le passage à l'échelle.
- Elles offrent une résilience accrue en cas de panne d'un composant, car les messages peuvent être stockés temporairement dans des files ou *topics*.

Parmi les modèles les plus connus :

1.2.3.1. Lambda Architecture : combine à la fois le traitement en flux et le traitement par lot pour obtenir un bon équilibre entre précision et réactivité [8] .

1.2.3.2. Kappa Architecture : variante plus simple, basée uniquement sur le traitement en flux, adaptée aux systèmes modernes où la fraîcheur de la donnée prime. [9]

1.2.3.3. Microservices réactifs utilisant Kafka : chaque module (ou microservice) produit ou consomme des événements, facilitant l'orchestration modulaire de tout le système [10].

Chaque bloc est indépendant, mais tous sont reliés par la circulation des événements via Kafka. Cette organisation permet une grande flexibilité, une meilleure tolérance aux erreurs et une mise à jour continue des résultats.

1.3. Technologies de diffusion et de traitement en continu

1.3.1. Apache Kafka

À l'ère des données massives et du besoin croissant d'agilité dans le traitement de l'information, Apache Kafka s'est imposé comme l'une des solutions les plus robustes et les plus utilisées pour gérer des flux de données en temps réel. Initialement développé par LinkedIn avant d'être proposé en open source au sein de la fondation Apache, Kafka est aujourd'hui au cœur de nombreuses architectures distribuées modernes [11].

Dans sa philosophie, Kafka repose sur une idée simple mais puissante : permettre à différentes applications de communiquer entre elles par des flux continus de messages, appelés *événements*. Plutôt que d'attendre que les données soient prêtes pour être traitées en bloc, Kafka permet à chaque donnée — ou message — d'être transmise, stockée et traitée dès son apparition, dans une logique totalement asynchrone et décentralisée [11].

1.3.2. L'architecture de kafka

Kafka s'organise autour de quelques concepts clés [12] :

- **Les producteurs** (*producers*) sont les composants chargés d'envoyer les données à Kafka. Dans notre projet, les producteurs sont les scripts Python qui récupèrent les publications depuis Reddit ou les articles via NewsAPI. Ces messages sont ensuite injectés dans Kafka sous forme d'objets JSON.
- **Les topics** sont des canaux nommés dans lesquels les messages sont stockés. On peut imaginer un topic comme une grande boîte aux lettres où plusieurs producteurs peuvent déposer des messages, et où plusieurs consommateurs peuvent venir les lire.
- **Les partitions**, quant à elles, permettent de découper un topic en segments parallèles, améliorant ainsi les performances en distribuant la charge sur plusieurs serveurs.
- **Les consommateurs** (*consumers*) lisent les messages depuis un topic. Dans notre architecture, c'est le module NLP (celui qui analyse les textes en temps réel) qui agit comme consommateur principal du topic `raw-news`.
- Enfin, Kafka fonctionne à l'intérieur d'un **cluster de brokers**, c'est-à-dire plusieurs serveurs coordonnés entre eux pour stocker et distribuer les données. Kafka utilise également (dans ses versions classiques) **ZooKeeper** pour synchroniser les brokers, bien que les versions récentes proposent une architecture allégée sans ce composant.

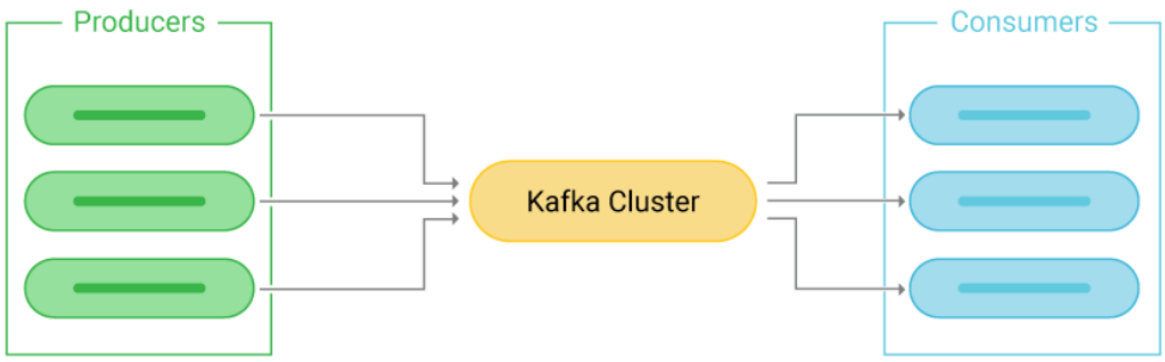

Figure 1 : Kafka architecture overview

1.4. Comparaison de Kafka avec les solutions existantes

Le traitement des flux de données en temps réel est un domaine stratégique dans les architectures modernes. Plusieurs solutions logicielles permettent aujourd'hui d'implémenter des systèmes de diffusion et de traitement événementiel. Apache Kafka est l'une des plus utilisées, mais il existe d'autres alternatives reconnues sur le marché. Cette section présente les principales solutions concurrentes, puis les compare à Kafka selon des critères techniques, opérationnels et fonctionnels.

1.4.1. Principales solutions de traitement de flux

Voici un panorama des solutions équivalentes ou alternatives à Kafka :

- **RabbitMQ** : Système de message orienté file (message broker) basé sur le protocole AMQP. Utilisé pour les communications entre services via des files de messages persistantes [13].
- **Apache Pulsar** : Plateforme de messagerie distribuée développée par Yahoo, offrant des fonctions proches de Kafka mais avec une séparation claire entre stockage et traitement [14].
- **Amazon Kinesis** : Service managé proposé par AWS pour l'ingestion, le traitement et l'analyse de flux en temps réel [15].
- **Azure Event Hubs** : Service cloud de Microsoft pour la collecte et la diffusion d'événements, intégré à l'écosystème Azure [16].
- **Redis Streams** : Fonctionnalité ajoutée à Redis pour permettre le traitement de flux en mémoire [17].

1.4.2. Comparaison entre Kafka et ses alternatives

Critère	Apache Kafka	RabbitMQ	Apache Pulsar	Amazon Kinesis	Redis Streams
Modèle de message	Pub/Sub basé sur logs persistants	File de messages (queue) avec accusé de réception	Pub/Sub segmenté par topics et partitions	Pub/Sub avec découpage par shard	Stream circulaire avec ID et offset

Persistance des données	Oui (stockage sur disque configurable)	Limité (orienté file, messages expirables)	Oui (stockage séparé via BookKeeper)	Oui (stockage temporaire)	Oui (en mémoire, persistance optionnelle)
Scalabilité	Très élevée, horizontale par partitions	Limitée, nécessite du clustering	Excellente, stockage indépendant	Bonne, mais dépend du cloud AWS	Limitée à l'échelle mémoire
Performance	Très élevée (haute volumétrie, faible latence)	Excellente pour des tâches simples	Comparable à Kafka, parfois plus stable	Optimisé pour AWS, bonnes performances	Très rapide pour petits volumes
Tolérance aux pannes	Oui, via réplication	Moyenne (selon le mode durable ou non)	Oui (bookies + brokers)	Haute (gérée par AWS)	Faible si pas de mécanisme externe
Traitement en temps réel	Oui (intégration avec Kafka Streams, Spark, etc.)	Oui, mais moins adapté à l'analyse complexe	Oui (Pulsar Functions, Flink)	Oui, intégration avec Lambda	Possible mais sans outil avancé
Simplicité de mise en œuvre	Moyenne (plus complexe à configurer)	Facile	Plus complexe à déployer	Très simple via AWS	Simple pour des cas limités
Langages supportés	Java, Python, Scala, Go, etc.	Très large (Java, Python, PHP, etc.)	Java, Python, Go	SDKs pour Java, Python, Node.js	Clients pour plusieurs langages
Communauté & Écosystème	Très vaste, mature, supporté par Confluent	Très mature aussi	Moins populaire mais en croissance	Forte, mais dépendante du cloud AWS	En croissance

Tableau 3: Comparaison entre Apache Kafka et d'autres solutions de traitement de flux

Malgré une courbe d'apprentissage initiale plus élevée, Kafka représente aujourd'hui une solution robuste et éprouvée, adaptée aux pipelines de traitement temps réel distribués, comme celui que nous avons mis en place [17] [15] [13] [14].

1.5 Fondements du Traitement Automatique du Langage Naturel

1.5.1. Définition du NLP et principales tâches

Le Traitement Automatique du Langage Naturel est une discipline qui cherche à faire le lien entre le langage humain et les machines. L'objectif est simple en apparence : permettre aux

ordinateurs de comprendre, interpréter, analyser, voire produire du texte ou de la parole de manière intelligente. En réalité, c'est un domaine complexe qui mobilise à la fois des connaissances en linguistique, en informatique, et en intelligence artificielle.

Le langage humain est riche, souvent ambigu, et fortement dépendant du contexte. Le NLP s'efforce donc de modéliser cette complexité pour que les machines puissent traiter des textes comme des humains le feraient — ou du moins, s'en rapprocher.

Au fil des années, les approches sont passées d'un traitement basé sur des règles linguistiques (grammaires formelles, dictionnaires, structures syntaxiques), à des méthodes d'apprentissage automatique, puis à des modèles d'apprentissage profond (deep learning), capables de saisir des subtilités bien plus fines du langage [18].

1.5.2. Les grandes familles de tâches en NLP

Le NLP comprend un ensemble de tâches qui s'enchaînent souvent dans un pipeline (chaîne de traitement). Voici les principales [19] :

i. Prétraitement linguistique

a. Tokenisation : c'est le point de départ. Le texte est découpé en unités significatives appelées *tokens* (mots, ponctuation, symboles).
b. Élimination des mots vides (stopwords) : certains mots très fréquents n'apportent pas de signification utile à l'analyse (comme "le", "et", "de"). On les retire souvent.
c. Lemmatisation / racinisation : les mots sont ramenés à leur forme canonique ("mangé", "mangeons", "mangeait" → "manger").

ii. Analyse morphologique et syntaxique

a. POS tagging (part-of-speech) : chaque mot est associé à sa catégorie grammaticale (nom, verbe, adjectif…).
b. Analyse syntaxique : on identifie la structure de la phrase, les dépendances entre les mots, etc.

iii. Analyse sémantique

a. Désambiguïsation **:** un mot peut avoir plusieurs sens ; le NLP tente d'en choisir le bon selon le contexte.

b. NER (Named Entity Recognition) : on détecte les noms de personnes, lieux, organisations, dates, etc., dans le texte.

iv. Analyse d'opinion et de contenu

a. Analyse de sentiment : le système évalue si un texte exprime une émotion positive, négative ou neutre.
b. Classification de texte : on attribue une catégorie ou un thème à un document.
c. Reconnaissance d'intention : très utilisée dans les chatbots ou assistants vocaux pour comprendre ce que l'utilisateur veut faire.

v. Génération de texte

a. Résumé automatique**,** traduction**,** ou génération de réponses sont des exemples de NLP "productif".

Ces opérations, appliquées en temps réel, enrichissent chaque message analysé et alimentent les visualisations du tableau de bord.

1.5.3. Étapes préliminaires : nettoyage, normalisation, tokenisation

Avant de pouvoir comprendre ce que disent un texte, un système informatique doit d'abord apprendre à le lire correctement. Mais contrairement à un humain, une machine ne sait pas ignorer les fautes de frappe, les hashtags, ou les liens superflus. C'est pourquoi une série d'étapes de prétraitement est indispensable avant toute analyse linguistique [20].

1.5.3.1. Nettoyage du texte

Le premier réflexe, c'est de nettoyer. Il s'agit ici de supprimer tout ce qui n'a pas de valeur sémantique pour le système : balises HTML, adresses web, emojis, mentions (@), caractères spéciaux ou répétitifs... Dans un tweet ou un post Reddit, le texte utile est souvent noyé dans un flot de distractions.

L'idée est simple : plus le texte est propre, plus les analyses qui suivent seront fiables. On peut aussi en profiter pour retirer les mots très fréquents (les "le", "et", "de"...) si l'objectif est de détecter les mots porteurs de sens [20].

1.5.3.2. Normalisation

Ensuite vient l'étape de normalisation, qui vise à mettre tous les textes sur un pied d'égalité. Par exemple, écrire tout en minuscules évite de distinguer inutilement entre "Politique" et "politique". On peut aussi remplacer certains caractères spéciaux ou abréviations courantes ("bcp" devient "beaucoup").

Un autre point important est la réduction des mots à leur forme de base : "marchera", "marchent" ou "marché" deviennent tous "marcher". Cela permet au système de regrouper des variantes d'un même mot, ce qui est très utile pour éviter la dispersion de l'information [20].

1.5.3.3. Tokenisation

Enfin, il faut découper le texte en petites unités, appelées *tokens*. Généralement, ce sont des mots, mais ce n'est pas toujours aussi simple : certains modèles préfèrent travailler sur des morceaux de mots (comme "##ation" dans "nation"). Cette étape, appelée tokenisation, permet à l'ordinateur de manipuler le texte morceau par morceau.

Selon la langue ou le type de texte, on peut ajuster cette découpe pour être plus fine ou plus large. Des outils comme spaCy**,** NLTK ou encore les tokenizers de Hugging Face offrent des méthodes puissantes pour faire ce travail correctement [20].

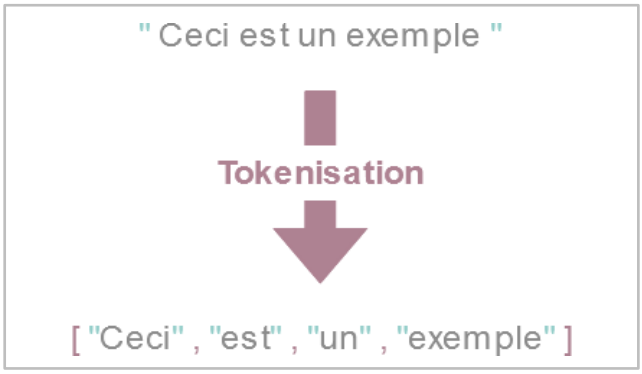

Figure 2 : exemple de tokeniation

1.5.4. Analyse de sentiment

Quand on lit un message en ligne, on ne se contente pas de comprendre les mots : on ressent aussi l'intention de l'auteur. Est-ce une critique, un éloge, une plainte déguisée ? C'est exactement ce que cherche à capter l'analyse de sentiment. Cette tâche du traitement

automatique du langage naturel (TAL) consiste à détecter automatiquement l'émotion ou l'opinion exprimée dans un texte.

Qu'il s'agisse d'un tweet, d'un commentaire Reddit ou d'un titre d'actualité, savoir si le ton est positif, négatif ou neutre peut être décisif : pour une entreprise qui surveille sa réputation, un journaliste qui suit une controverse, ou un analyste qui détecte un changement de tendance.

1.5.4.1. Origines et évolution historique

L'analyse de sentiment, aussi appelée *opinion mining*, est apparue dans les années 2000, lorsque les chercheurs ont voulu extraire automatiquement l'opinion des internautes sur des produits, des films ou des personnalités publiques [21].

Au départ, l'approche était simple : on se basait sur des listes de mots positifs ou négatifs. Mais très vite, les limites sont apparues. Par exemple, comment interpréter une phrase comme *"*Ce film était tellement mauvais que j'en ai ri*"* ? Un humain saisit le sarcasme, mais une machine non.

L'évolution a donc été progressive : d'abord des règles simples, ensuite des modèles statistiques, et enfin, des modèles neuronaux puissants capables de lire entre les lignes. L'analyse de sentiment est ainsi passée d'un outil naïf à un véritable moteur d'intelligence émotionnelle. [22]

1.5.4.2. Méthodes basées sur les lexiques (approche symbolique)

Les premières approches utilisaient des lexiques de sentiments qui sont des listes de mots associés à une polarité (positive ou négative). Par exemple, "excellent" = +3, "nul" = -2. On analysait ensuite un texte en calculant une sorte de "score moyen" d'opinion.

Parmi les outils les plus efficaces dans cette catégorie, on retrouve VADER (Valence Aware Dictionary and sentiment Reasoner), qui a été spécialement conçu pour traiter le langage informel et expressif des réseaux sociaux. VADER tient compte des majuscules, de la ponctuation excessive ("!!!!"), des emojis, et même du contexte émotionnel de certains mots.

Ces méthodes ont l'avantage d'être rapides, simples à mettre en place, et interprétables. On sait pourquoi un texte a été jugé positif ou négatif. Mais elles ont aussi des faiblesses : elles ne comprennent pas le contexte global ni la structure grammaticale. Un mot comme "terrible" peut

être positif ("terrible performance !") ou négatif ("terrible erreur…"), selon la phrase. Et là, les lexiques se trompent. [23]

1.5.4.3. Méthodes statistiques et apprentissage supervisé

Pour aller plus loin, les chercheurs ont commencé à entraîner des modèles à partir de données annotées. Le principe est simple : on donne à l'algorithme des milliers (voire des millions) de phrases avec leur sentiment associé, et on le laisse apprendre.

Ces modèles utilisent souvent des représentations du texte comme :

- **Bag-of-words** (sac de mots) : on considère uniquement quels mots sont présents, sans ordre ni grammaire. [24]
- **TF-IDF** : on pondère les mots selon leur importance relative dans un texte et dans le corpus global. [24]

Avec ces représentations, des algorithmes comme les SVM**,** naive Bayes ou les forêts aléatoires étaient utilisés pour classer le sentiment. Ces techniques ont montré une bien meilleure capacité d'adaptation aux variations de langage, surtout dans des corpus bien définis comme des avis clients [25].

Cependant, ces modèles restaient limités par leur incapacité à comprendre la structure du langage. Pour eux, *"*je ne déteste pas*"* ressemblait beaucoup à *"*je déteste*"*. [24]

1.5.3.4 Approches modernes par Transformers

Aujourd'hui, l'analyse de sentiment est dominée par les modèles de type transformer, comme BERT**,** RoBERTa**,** DistilBERT, ou encore twitter-roBERTa (formé spécialement sur des tweets).

Ces modèles sont capables de :

- comprendre le contexte global de chaque mot (grâce à l'attention),
- gérer la négation, les tournures complexes, et parfois même l'ironie,
- être fine-tunés sur un domaine spécifique (politique, sport, finance…).

Ils lisent le texte comme un humain le ferait, en tenant compte de la position des mots, de leur relation syntaxique, et de l'intention générale. Ils sont entraînés sur des milliards de phrases et

savent capturer des nuances subtiles. Par exemple, *"Ce candidat est loin d'être idiot"* est bien compris comme un compliment.

Grâce à des bibliothèques comme Hugging Face Transformers, ces modèles sont devenus facilement accessibles, même pour des projets modestes. Il est possible d'utiliser des modèles pré-entraînés, ou d'en affiner un sur ses propres données.

C'est cette flexibilité et cette puissance qui font des transformers la solution de référence actuelle pour l'analyse de sentiment à grande échelle, notamment dans les systèmes en temps réel. [26]

1.5.5. Reconnaissance d'entités nommées (NER)

La reconnaissance d'entités nommées (Named Entity Recognition, ou NER) est une tâche centrale du traitement automatique du langage naturel (TALN), qui vise à détecter automatiquement les entités mentionnées dans un texte (personnes, lieux, organisations, dates, etc.). Elle est indispensable pour transformer des documents non structurés en informations exploitables [27].

Historiquement, la NER a évolué à travers plusieurs approches :

- **Approches symboliques (règles) :** Ces méthodes s'appuient sur des règles linguistiques définies manuellement. Par exemple, une règle pourrait dire : « Si un mot commence par une majuscule et suit un titre comme "Président", alors il s'agit probablement d'une personne ». Bien que précises dans des contextes spécifiques (ex. documents juridiques), elles sont difficiles à adapter à de nouveaux domaines ou langues et requièrent un travail de maintenance coûteux [28].
- **Approches statistiques (CRF, HMM) :** Ces méthodes apprennent des modèles probabilistes à partir de corpus annotés, comme CoNLL-2003. Les CRF (Conditional Random Fields) et HMM (Hidden Markov Models) évaluent la probabilité qu'un mot appartienne à une catégorie d'entité en fonction de son contexte. Elles ont représenté un progrès significatif par rapport aux règles figées, notamment en termes de généralisation et de robustesse [29].
- **Apprentissage profond et modèles Transformers** : Avec l'arrivée des réseaux neuronaux comme les LSTM ou BiLSTM, puis des architectures Transformers (ex. BERT, RoBERTa), la NER a franchi un cap. Ces modèles peuvent capter le contexte

global d'une phrase et traiter les ambiguïtés sémantiques complexes, comme différencier "Amazon" l'entreprise et "Amazon" le fleuve. Ils sont également capables d'être *fine-tunés* sur des domaines spécifiques (médical, juridique, politique...) [30].

Aujourd'hui, plusieurs bibliothèques open-source facilitent la mise en œuvre de la NER :

- **spaCy** : rapide, efficace, avec des pipelines pré-entraînés pour plusieurs langues [31].
- **Flair** (de Zalando Research) : permet d'utiliser des embeddings contextuels empilables [32].
- **Hugging Face Transformers** : offre des modèles (BERT, RoBERTa, etc.) facilement personnalisables et intégrables [30].

Ces outils permettent d'extraire des entités dans des textes bruyants (comme des tweets ou titres d'articles) avec un haut niveau de précision, et peuvent être combinés avec des bases de données ou tableaux de bord pour créer des systèmes de surveillance automatique.

1.6. Bases de données orientées documents

Dans un système de traitement en flux, les données arrivent en continu. Il ne suffit donc pas d'avoir une base de données puissante : il faut surtout qu'elle soit capable d'écrire et lire rapidement**,** sans ralentir l'analyse.

Les bases orientées documents sont particulièrement bien adaptées à ce contexte :

- Elles permettent d'insérer rapidement des documents textuels enrichis (avec sentiment, entités, mots-clés, etc.).
- Elles sont scalables horizontalement, donc faciles à adapter à un grand volume.
- Elles permettent des requêtes complexes sur des champs spécifiques, même dans des structures profondes.

Autrement dit, elles facilitent un stockage fluide, mais aussi une extraction ciblée, ce qui est essentiel dans un tableau de bord de tendances en temps réel [33].

1.6.1. MongoDB : structure documentaire, flexibilité, requêtes complexes

MongoDB est aujourd'hui l'une des bases NoSQL les plus utilisées. Elle stocke les données sous forme de documents BSON (un format binaire de JSON) regroupés dans des collections. Chaque document peut avoir une structure libre, ce qui offre une grande souplesse.

Quelques points forts de MongoDB :

- Modèle documentaire parfaitement adapté aux données textuelles enrichies (ex : résultats NLP).
- Requêtes puissantes avec des filtres, agrégations, et recherches textuelles.
- Intégration fluide avec Python via des bibliothèques comme `pymongo`, très utilisées dans les projets de data science et d'IA.
- Indexation intelligente pour accélérer les requêtes fréquentes (par date, source, sentiment...).

Dans un projet de veille sémantique, MongoDB s'impose comme un choix naturel pour stocker, organiser et interroger les données textuelles enrichies en temps réel, tout en gardant une architecture souple et évolutive [34].

1.6.2. Autres forme de stockage NoSQL

Bien que MongoDB ait été retenue comme solution principale, d'autres bases NoSQL auraient également pu être envisagées. Voici un aperçu de deux alternatives populaires, avec leurs avantages et limites [35] :

Base de données	**Avantages**	**Inconvénients**
Elasticsearch	Excellente pour la recherche plein texte et l'analyse rapide ; support natif des agrégations, intégration avec Kibana	Moins intuitive pour l'écriture de documents complexes ; surcharge mémoire possible
Apache Cassandra	Très bon support de la scalabilité horizontale ; résilience élevée	Moins adapté aux requêtes complexes ou multi-critères ; schéma plus rigide

Tableau 4 : Comparaison des Bases de Données : Elasticsearch et Apache Cassandra

- Elasticsearch aurait été un bon choix si le projet avait nécessité une fonctionnalité de recherche textuelle avancée, comme l'indexation floue ou le scoring de pertinence.
- Cassandra excelle dans les environnements distribués à très forte volumétrie, mais son modèle de données plus strict est moins adapté à des documents textuels riches et variés comme ceux de ce projet.

1.7. Systèmes de veille : approches existantes

Aujourd'hui, la capacité à analyser et visualiser des données textuelles en temps réel est devenue cruciale dans de nombreux domaines : surveillance médiatique, détection de tendances, suivi d'opinion publique ou encore gestion de crise. Ce besoin a donné naissance à divers travaux de recherche et outils technologiques combinant collecte de données, traitement automatique du langage (TAL), et restitution graphique sous forme de tableaux de bord interactifs [36].

1.7.1. Travaux et projets existants

Plusieurs initiatives, tant académiques qu'industrielles, ont proposé des solutions pour suivre le flux d'informations en ligne. Parmi les projets notables :

- **Talaia** : développé par l'Université du Pays Basque (UPV/EHU), est un système de surveillance multilingue utilisé notamment pendant des campagnes électorales. Il permet d'analyser en temps réel des flux Twitter dans plusieurs langues, en extrayant sentiments et entités. L'interface, bien que puissante, repose sur des composants JavaScript spécifiques, ce qui peut limiter son adoption hors contexte expert [37].
- D'autres projets académiques ou de recherche utilisent des combinaisons comme Tweepy + Elasticsearch + Kibana pour suivre les discussions sur Twitter. Ces systèmes offrent une visualisation efficace via Kibana, mais ils dépendent fortement d'Elasticsearch, ce qui rend leur adaptation à des traitements linguistiques personnalisés plus difficile [38].
- Côté solutions professionnelles, des plateformes telles que NewsWhip qui propose des services d'alerte et de veille en temps réel, utilisés par les rédactions, les gouvernements ou les entreprises. Ces solutions sont très performantes, mais restent propriétaires, coûteuses et peu flexibles, surtout dans un contexte académique ou open source [39].

1.7.2. Limites observées

Si ces travaux montrent clairement l'intérêt croissant pour l'analyse en temps réel, ils présentent aussi plusieurs limites [40] :

- Manque d'intégration fluide avec les bibliothèques NLP modernes : rares sont les systèmes qui intègrent de manière native des modèles de type Transformers ou spaCy, pourtant devenus des standards dans le domaine.
- Complexité technique : plusieurs outils reposent sur des architectures lourdes (clusters Spark, Elasticsearch, serveurs Java), peu adaptées aux projets de recherche individuels ou aux équipes à effectif réduit.
- Visualisation peu orientée linguistique : la plupart des interfaces se concentrent sur des métriques numériques (comptes, fréquences, pics de mentions), mais la mise en contexte sémantique des discours, l'analyse des sentiments ou des entités est souvent secondaire ou absente.
- Barrière technologique pour les non-développeurs **:** les systèmes actuels requièrent souvent des compétences avancées en DevOps, ce qui freine leur appropriation par des analystes, journalistes ou étudiants.

1.7.3. Avantage de notre système

Le système développé dans le cadre de ce mémoire a justement été pensé pour répondre à ces défis tout en restant simple, agile et ancré dans l'écosystème Python. Il apporte plusieurs éléments nouveaux :

- Une architecture épurée et 100 % Python : en utilisant Kafka pour la gestion des flux, MongoDB pour le stockage, et Dash/Flask pour la visualisation, le tout est cohérent et facilement maintenable.
- Une intégration native de modèles NLP puissants, tels que spaCy pour l'extraction d'entités, VADER + Transformers pour l'analyse de sentiments.
- Une interface interactive légère mais fonctionnelle, accessible via un simple navigateur, sans dépendre de systèmes externes ou de langages additionnels (comme JavaScript).
- Un focus sur l'interprétabilité : les données ne sont pas seulement affichées, elles sont contextualisées linguistiquement — qu'il s'agisse des entités nommées les plus mentionnées, des émotions dominantes, ou des sources les plus influentes.

1.8. Conclusion

L'étude de l'état de l'art met en lumière l'importance croissante des traitements en temps réel couplés aux méthodes de NLP modernes. Kafka, combiné à des bibliothèques comme spaCy ou Transformers, permet d'enrichir des flux d'informations avec des analyses linguistiques fines. De plus, MongoDB offre une base souple et performante pour stocker ces données hétérogènes. Ce cadre théorique pose ainsi les bases solides pour la conception du système présenté dans le chapitre suivant.

Chapitre 2 : Conception et modélisation du système

2.1. Introduction

Dans ce chapitre, on présente l'architecture générale du système conçu pour analyser des flux d'informations en temps réel. On y détaille le rôle des producteurs (Reddit et NewsAPI), l'analyse NLP avec spaCy, VADER et Transformers, le stockage flexible via MongoDB, ainsi que le mécanisme de nettoyage automatique toutes les 20 minutes. Enfin, on justifie les choix techniques faits selon les besoins du projet

2.2. Architecture logicielle du système

L'architecture logicielle du système repose sur une conception modulaire et orientée flux**,** dans laquelle chaque composant a un rôle spécifique, tout en communiquant de manière fluide avec les autres. Cette organisation permet d'assurer un traitement en temps réel**,** depuis la collecte des données jusqu'à leur affichage sur une interface interactive.

L'ensemble du système peut être vu comme un pipeline de traitement**,** où les données textuelles circulent à travers plusieurs étapes successives, chacune enrichissant ou structurant l'information.

2.2.1. Schéma général du pipeline

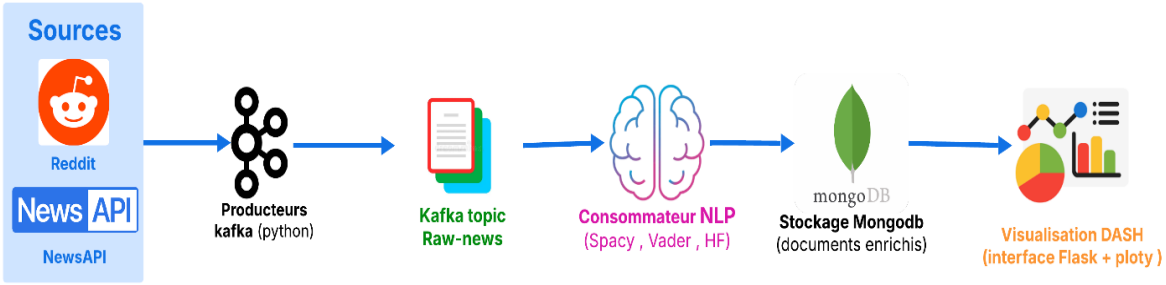

Figure 3: Architecture du pipeline de traitement des tendances

2.2.1.1. Explication des composants

- **Sources (Reddit, NewsAPI)**
  Le système interroge régulièrement deux types de flux : les messages récents de

subreddits populaires, et les articles de presse issus de NewsAPI. Ces messages sont récupérés sous forme brute (texte, titre, source, URL, date…).

- **Producteurs Kafka**

  Chaque source dispose d'un producteur Kafka dédié qui envoie les messages dans un topic Kafka commun appelé `raw-news`. Cela permet de centraliser tous les flux dans une file d'attente unique, en temps réel.

- **Consommateur NLP**

  Le cœur du traitement est réalisé par un consommateur Python, qui lit les messages dès leur publication sur Kafka. Il applique :

  1. Un nettoyage lexical (suppression des URLs, stopwords…),
  2. Une tokenisation des textes,
  3. Une reconnaissance d'entités nommées via *spaCy*,
  4. Une double analyse de sentiment : VADER (lexicale) + Transformer (contextuelle),
  5. Puis stocke le tout dans MongoDB.

- **Base de données MongoDB**

  Chaque message est stocké sous forme d'un document JSON enrichi contenant les métadonnées, les sentiments, les entités, etc. MongoDB est choisie pour sa souplesse de structure et sa rapidité.

- **Dashboard Dash**

  Enfin, un tableau de bord développé avec Dash permet d'afficher les résultats sous forme de graphiques dynamiques. L'utilisateur peut y consulter :

  1. Les messages les plus récents,
  2. Les entités les plus citées,
  3. Les mots-clés dominants,
  4. Les sources les plus actives,
  5. La répartition des sentiments en temps réel.

2.2.2. Intégration globale et orchestration du flux

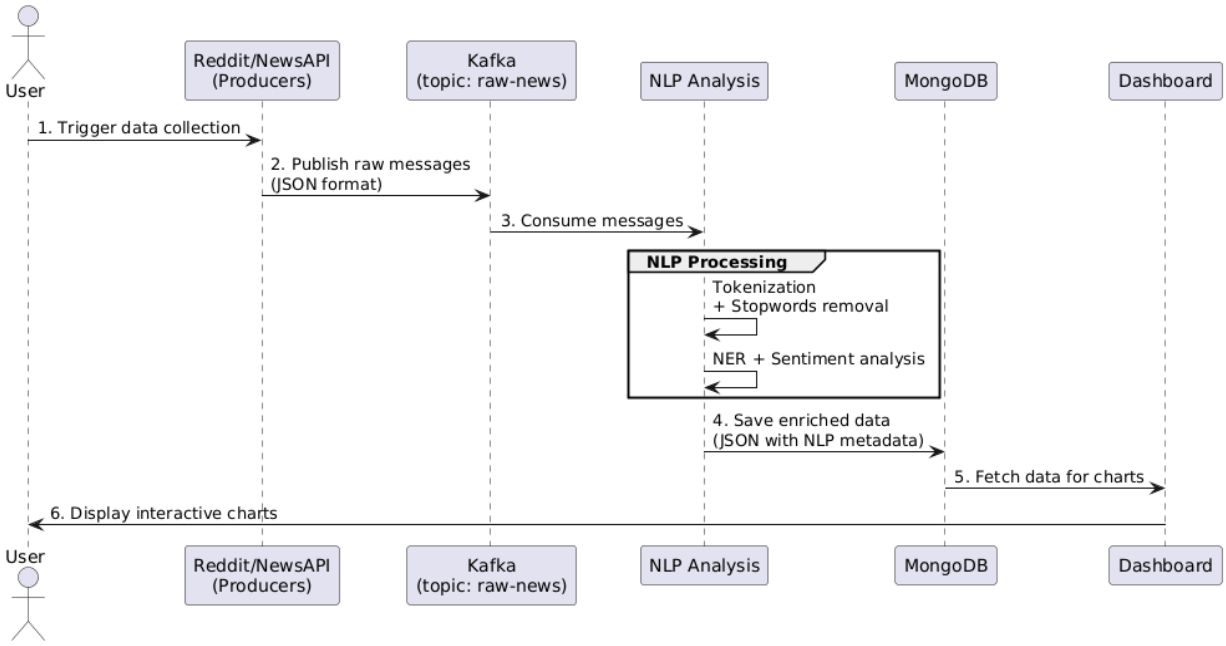

Figure 4: diagramme de sequence du pipeline de parcours d'un message

Le parcours d'un message commence lorsqu'un utilisateur déclenche la collecte des données (étape 1) en activant un bouton dans l'application Flask. Cette action lance les producteurs (reddit_producers.py et news_producer.py) qui se connectent aux API (Reddit ou NewsAPI) pour récupérer des articles ou publications en temps réel. Ces messages bruts sont ensuite publiés dans un topic Kafka nommé raw-news (étape 2), au format JSON. Le module nlp_analysis.py agit alors comme un consommateur Kafka (étape 3) : il lit ces messages, les prétraite avec une série de tâches NLP incluant la tokenisation, la suppression des stopwords, l'analyse de sentiment avec un modèle Transformers, et la reconnaissance d'entités nommées (NER) avec spaCy. Une fois enrichis avec des métadonnées NLP (étape 4), les messages sont stockés dans une base MongoDB**,** dans une collection dédiée. Ensuite, le tableau de bord Dash (connecté à Flask) interroge régulièrement MongoDB pour récupérer les données enrichies (étape 5) et afficher dynamiquement des visualisations interactives sous forme de graphiques, mots-clés, entités ou sentiments (étape 6). Ainsi, l'utilisateur visualise en temps réel les tendances collectées et analysées depuis Reddit ou NewsAPI.

2.3. Communication entre modules

Dans un système distribué orienté temps réel, la communication entre les différents modules est un élément central de l'architecture. Chaque composant du système (collecte, traitement, stockage, visualisation) fonctionne de manière indépendante, mais tous doivent échanger des données de manière rapide, fiable et compréhensible**.**

Ce projet s'appuie principalement sur trois mécanismes de communication : Kafka (topics), JSON (structure des messages), et HTTP (via Dash/Flask).

Kafka joue le rôle de file d'attente centrale pour les messages collectés depuis Reddit et NewsAPI. Chaque message produit est publié dans un topic appelé `raw-news`, et ce topic est ensuite consommé en continu par le module de traitement NLP.

- **Format** : chaque message Kafka est un objet JSON sérialisé**.**
- **Avantages** :
    - Découplage total entre collecte et traitement.
    - Possibilité d'ajouter d'autres consommateurs plus tard (ex. : pour l'archivage, les alertes…).
    - Garantie d'ordre et persistance temporaire.

Tous les messages échangés dans le système (Kafka → NLP → MongoDB) utilisent le format JSON**.** Ce choix est motivé par sa lisibilité, sa flexibilité (schéma libre), et sa compatibilité native avec MongoDB et Python [41].

Chaque message JSON suit une structure enrichie qui contient à la fois :

- Les données brutes : texte, source, date, URL.
- Les résultats NLP : tokens nettoyés, entités nommées, scores de sentiment (VADER + HF)

2.4. Conception des producteurs de données

La collecte des données constitue la porte d'entrée du pipeline. Elle repose sur des producteurs Kafka spécialisés, chargés de récupérer les messages en temps réel depuis deux sources principales : Reddit et NewsAPI. Ces producteurs sont conçus pour être autonomes, cycliques,

et robustes**,** afin d'assurer une alimentation constante du système avec des contenus textuels pertinents et à jour.

2.4.1. Producteur Reddit

Le producteur Reddit est conçu comme un module autonome de collecte de données textuelles. Il s'appuie sur l'API publique de Reddit pour extraire périodiquement des publications populaires à partir de plusieurs subreddits.

C'est objectifs principales sont :

- Collecter des messages textuels en temps réel depuis Reddit afin de détecter les sujets d'actualité les plus populaires**.**
- Publier ces messages sous forme d'événements Kafka dans le topic commun `raw-news`, accessible par le consommateur NLP.

2.4.1.1. Fonctionnalités clés

a. **Actualisation dynamique des subreddits surveillés** :

Toutes les 5 minutes, le système identifie automatiquement les communautés les plus actives directement depuis la page d'accueil de Reddit. Cela permet de suivre l'actualité en temps réel sans se limiter à une liste prédéfinie

b. **cyclique des publications** :

Toutes les 30 secondes**,** les nouveaux posts sont récupérés depuis c'est subreddits. Un filtrage intelligent permet d'éviter l'envoi de doublons.

c. **Strucuration des données** :

Chaque publication est transformée en un message JSON clair, comprenant le texte, la source, l'URL, ainsi que les dates de publication et de collecte. Un marqueur (reddit) indique son origine.

d. **Envoi vers Kafka** :

Les données sont ensuite transmises vers le topic raw-news sur Kafka, où elles pourront être traitées de manière asynchrone par d'autres modules (analyse NLP, stockage MongoDB, dashboard de visualisation, etc.).

Le schéma suivant illustre le fonctionnement général du producteur Reddit :

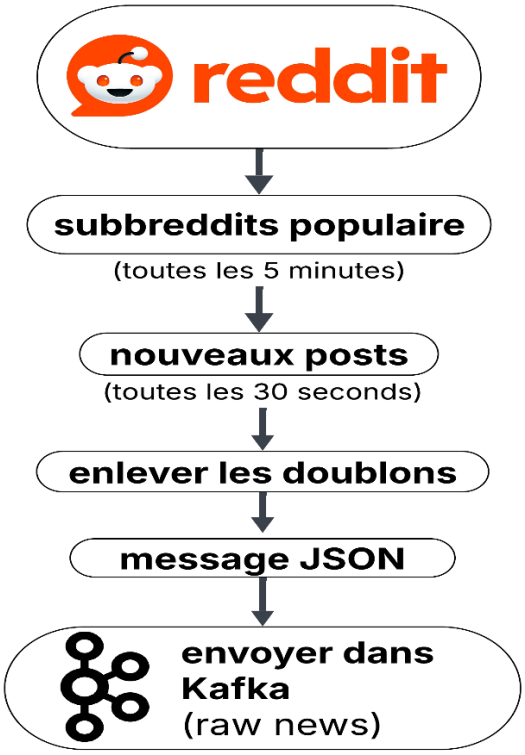

Figure 5 :Flux de collecte de données Reddit vers kafka

2.4.2. Producteur NewsAPI

Le producteur NewsAPI est conçu comme un module autonome de collecte de données textuelles, ayant pour rôle de récupérer régulièrement les titres des dernières actualités à partir d'un agrégateur de presse en ligne (NewsAPI.org). Il s'inscrit dans une architecture orientée événements, en contribuant à l'alimentation du flux Kafka centralisé.

C'est objectifs principales sont :

- Collecter des articles récents liés à des thématiques variées.
- Éviter les répétitions (doublons).
- Injecter les données dans Kafka sous un format structuré.

2.4.2.1. Fonctionnalités clés

a. Recherche aléatoire par mots clé

Un mot clé est sélectionné à chaque cycle pour interroger l'API, ce qui assure une couverture thématique variée

b. Appels réguliers à NewsAPI

Une requête est envoyée toutes les 30 secondes vers l'endpoint /v2/everything, garantissant une collecte continue.

c. Filtrage des doublons

Les titres déjà envoyés récemment sont ignorés afin de ne traiter que les actualités nouvelles.

d. Structuration uniforme des articles

Les données sont encapsulées dans des messages JSON enrichis, identifiés comme provenant de la source `"news"`.

e. Transmission vers Kafka

Les articles sont envoyés vers le topic raw-news pour être traités de façon asynchrone par les modules suivants du pipeline.

Le schéma suivant illustre le fonctionnement général du producteur NewsAPI :

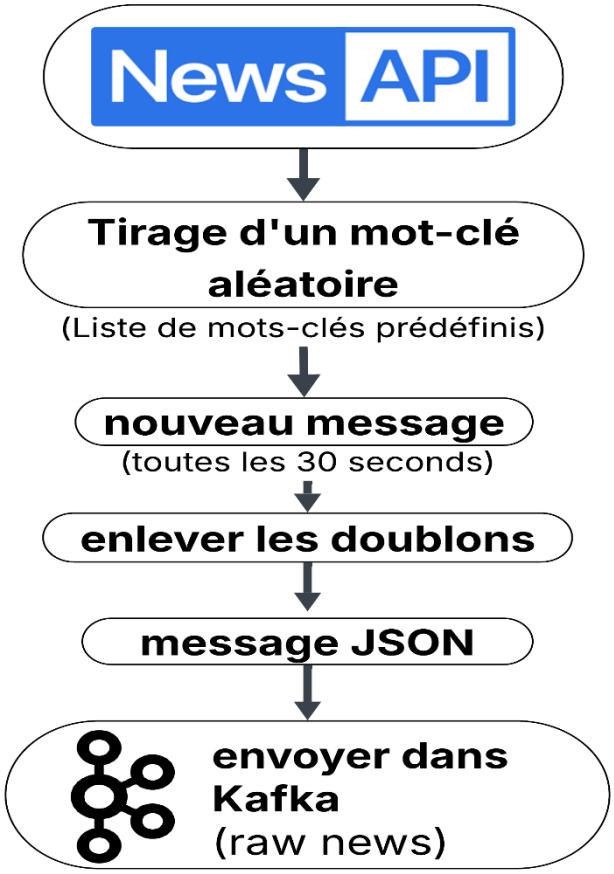

Figure 6: Flux de collecte de données NewsAPI vers kafka

2.4.3. Synchronisation avec l'analyseur NLP

Les deux producteurs (Reddit et NewsAPI) sont conçus pour fonctionner de manière asynchrone mais coordonnée avec le module d'analyse NLP. Les messages qu'ils produisent sont immédiatement envoyés dans le topic Kafka raw-news, qui agit comme une zone tampon centralisée. Ce topic est ensuite consommé en temps réel par l'analyseur NLP , garantissant ainsi une séparation claire des responsabilités. Les producteurs collectent et envoient, tandis que le NLP consomme, traite et enrichit.

Cette architecture permet :

- une modularité maximale, chaque composant peut être redémarré ou mis à jour indépendamment.
- une réactivité en continu, sans besoin d'attendre l'accumulation de données (contrairement au traitement par lot).

- un flux constant et fluide de données enrichies vers MongoDB et le tableau de bord.

2.5. Modélisation des données stockées

L'ensemble des données collectées puis enrichies par le module d'analyse NLP est stocké dans une base de données NoSQL (MongoDB), sous la forme de documents JSON. Chaque document représente une unité d'information complète, intégrant à la fois les métadonnées de la source, les résultats de l'analyse linguistique, ainsi que les éléments de contexte temporel.

Le schéma de données adopté est volontairement souple afin de s'adapter à différents types de sources (médias sociaux, articles de presse, etc.), tout en assurant une cohérence minimale dans les champs structurés.

Le format principal utilisé pour l'insertion dans MongoDB est nommé `final_result`. Il regroupe les informations brutes et enrichies sous la forme suivante :

2.5.1. Structure du document final_result

Voici un example de document [42] :

Figure 7 : structure de document final_results JSON

Champ	Description
`text`	Le contenu brut du message (titre ou texte).
`cleaned_tokens`	Liste des tokens nettoyés (sans stopwords, sans ponctuation).
`collected_at`	Timestamp de collecte par le producteur.
`source_type`	Type de source : `"reddit"` ou `"news"`.
`source`	Nom de la source ou du subreddit.
`url`	Lien direct vers l'article ou la publication.
`published_at`	Date de publication initiale du contenu.
`vader_sentiment`	Sentiment détecté par l'analyse VADER (POSITIVE, NEGATIVE, NEUTRAL).
`vader_confidence`	Score de confiance associé au résultat de VADER.
`hf_sentiment`	Sentiment détecté par le modèle Transformers (Hugging Face).
`hf_confidence`	Score de probabilité (softmax) du modèle Transformers.
`entities`	Liste des entités nommées extraites via spaCy (ex. : personnes, lieux, etc.).

Tableau 5 : Description des champs de données

2.5.2. Indexation et organisation dans MongoDB

Les données enrichies par le module d'analyse NLP sont stockées dans la collection analyzed_data de la base MongoDB trend_database, selon la structure du document final_result présentée plus haut (voir Figure 7). Ce format regroupe l'ensemble des informations brutes, les résultats du traitement linguistique, ainsi que les métadonnées temporelles et contextuelles.

La nature documentaire de MongoDB permet de gérer cette structure de façon flexible, en s'adaptant à la diversité des sources sans nécessiter de schéma rigide.

Pour garantir la rapidité des traitements et optimiser les requêtes (notamment pour l'affichage dans le tableau de bord), des index peuvent être appliqués sur certains champs stratégiques :

- collected_at : pour la suppression automatique des documents obsolètes toutes les 20 minutes ;
- hf_sentiment / vader_sentiment : pour la répartition des sentiments ;
- entities : pour identifier les entités les plus citées ;
- source et source_type : pour les regroupements par origine (Reddit, NewsAPI).

Ce modèle d'organisation assure un équilibre optimal entre performance, évolutivité et simplicité dans un contexte de traitement en temps réel [43].

2.5.3. Gestion des entités, sentiments, mots-clés et métadonnées

Le système de stockage ne se limite pas à enregistrer le contenu brut des messages collectés. Chaque document inséré dans MongoDB est enrichi avec des informations sémantiques extraites par le module NLP, ce qui permet une exploitation fine et orientée tendance de chaque message.

2.5.3.1. Entités nommées (`entities`)

À l'aide de spaCy, chaque texte est analysé pour détecter les entités nommées telles que les personnes, organisations, lieux ou événements. Ces entités sont stockées sous forme de paires (valeur, type) dans une liste structurée, facilitant leur agrégation ultérieure (ex. : top entités par jour).

2.5.3.2. Sentiment (vader_sentiment, hf_sentiment)

Deux approches complémentaires sont utilisées pour détecter le ton émotionnel des messages :

a. VADER : analyse de sentiment basée sur lexique

Dans ce projet, VADER est utilisé pour attribuer à chaque message un score de sentiment global**,** ainsi qu'une catégorie émotionnelle (positive, neutre, négative).

Parmis les avantages de vader :

- Très rapide à exécuter
- Simple à intégrer dans un pipeline Python
- Adapté aux formats courts et au langage informel

Cela en fait un excellent premier filtre pour l'analyse des émotions dans les messages textuels [23].

b. Transformers (Hugging Face) : classification avec modèles pré-entraînés

Pour obtenir une analyse plus fine et plus contextuelle du sentiment, le projet intègre également un modèle pré-entraîné basé sur une architecture Transformer, via la bibliothèque Hugging Face Transformers.

Le modèle utilisé est :

cardiffnlp/twitter-roberta-base-sentiment

Ce modèle a été entraîné sur un très grand volume de tweets et est donc parfaitement adapté aux messages courts et informels. Contrairement à VADER, il prend en compte la totalité du contexte linguistique, ce qui améliore la précision de la classification.

Le pipeline NLP intègre ce modèle de la façon suivante :

- Tokenisation du texte avec le tokenizer Hugging Face
- Prédiction du sentiment à l'aide du modèle RoBERTa
- Extraction d'un score de confiance, utilisé pour valider ou compléter les résultats de VADER

L'approche hybride (VADER + Transformers) combine les avantages des deux méthodes : la rapidité de l'analyse lexicale et la précision de l'analyse contextuelle [44].

2.5.3.3. Mots-clés (cleaned_tokens)

Une étape de prétraitement effectue le nettoyage du texte (suppression des liens, ponctuations, stopwords) suivi d'une tokenisation. Le résultat est une liste de mots utiles pouvant être utilisés pour des analyses de fréquence, de cooccurrence ou de clustering thématique.

2.5.3.4. Métadonnées (source, source_type, url, collected_at, published_at)

Ces informations permettent de conserver le contexte de chaque message : son origine (Reddit ou média), sa source exacte, le moment de sa collecte, sa date de publication initiale, ainsi que le lien original. Elles sont essentielles pour filtrer, agréger ou visualiser les tendances en fonction du temps et de l'origine.

2.6. Stratégie de nettoyage automatique

Dans un système de traitement de flux en temps réel, il est essentiel de ne pas accumuler indéfiniment les données, au risque de saturer la mémoire, d'altérer les performances, et surtout de compromettre la pertinence des résultats affichés. Pour répondre à ce besoin, une stratégie de nettoyage automatique a été intégrée au cœur de l'architecture [45].

2.6.1. Mécanisme de Rafraîchissement Automatique (toutes les 20 minutes)

Le but c'est de Supprimer régulièrement les anciennes données de MongoDB et relancer les producteurs pour garantir un affichage actualisé et léger dans le dashboard.

Le fonctionnement se déroule en plusieurs étapes :

2.6.1.1. Démarrage différé du thread de surveillance :

Lors du lancement de l'application Flask, un délai de 60 secondes (time.sleep(60)) est appliqué pour laisser Kafka et MongoDB démarrer correctement. Ensuite, un thread en arrière-plan (rolling_refresh_loop) est lancé pour gérer le cycle de nettoyage toutes les 20 minutes.

2.6.1.2. Détection de l'arrivée des premières données :

Le thread attend que le premier document arrive dans la base MongoDB (get_first_document_time()). Cela évite de lancer le cycle si aucune donnée n'a encore été collectée.

2.6.1.3. Décompte de 20 minutes :

Une fois qu'un premier document est détecté, le thread lance un compte à rebours de 20 minutes (time.sleep(1200)).

2.6.1.4. Réinitialisation automatique :

Une requête HTTP POST est envoyée à /reset_mongo :
→ Cette route supprime tous les documents de la collection MongoDB via delete_many({}).

Ensuite, les producteurs sont redémarrés en appelant /start_producer avec le nom de la source (news ou reddit) uniquement si ceux-ci ont déjà été initialisés (producers_initialized).

2.6.1.5. Cycle incrémental :

Le numéro de cycle (cycle_number) est affiché dans les logs à chaque boucle, ce qui facilite le suivi dans la console.

L'avantage de ce processus c'est :

- Assure un flux frais et régulier de données dans les graphes.
- Allège MongoDB, évitant une croissance continue de la base.
- Permet un affichage interactif toujours basé sur des tendances récentes.

Le diagramme ci-dessous illustre le mécanisme de rafraîchissement automatique du système décrit précédemment (suppression des données MongoDB et relance des producteurs toutes les 20 minutes)."

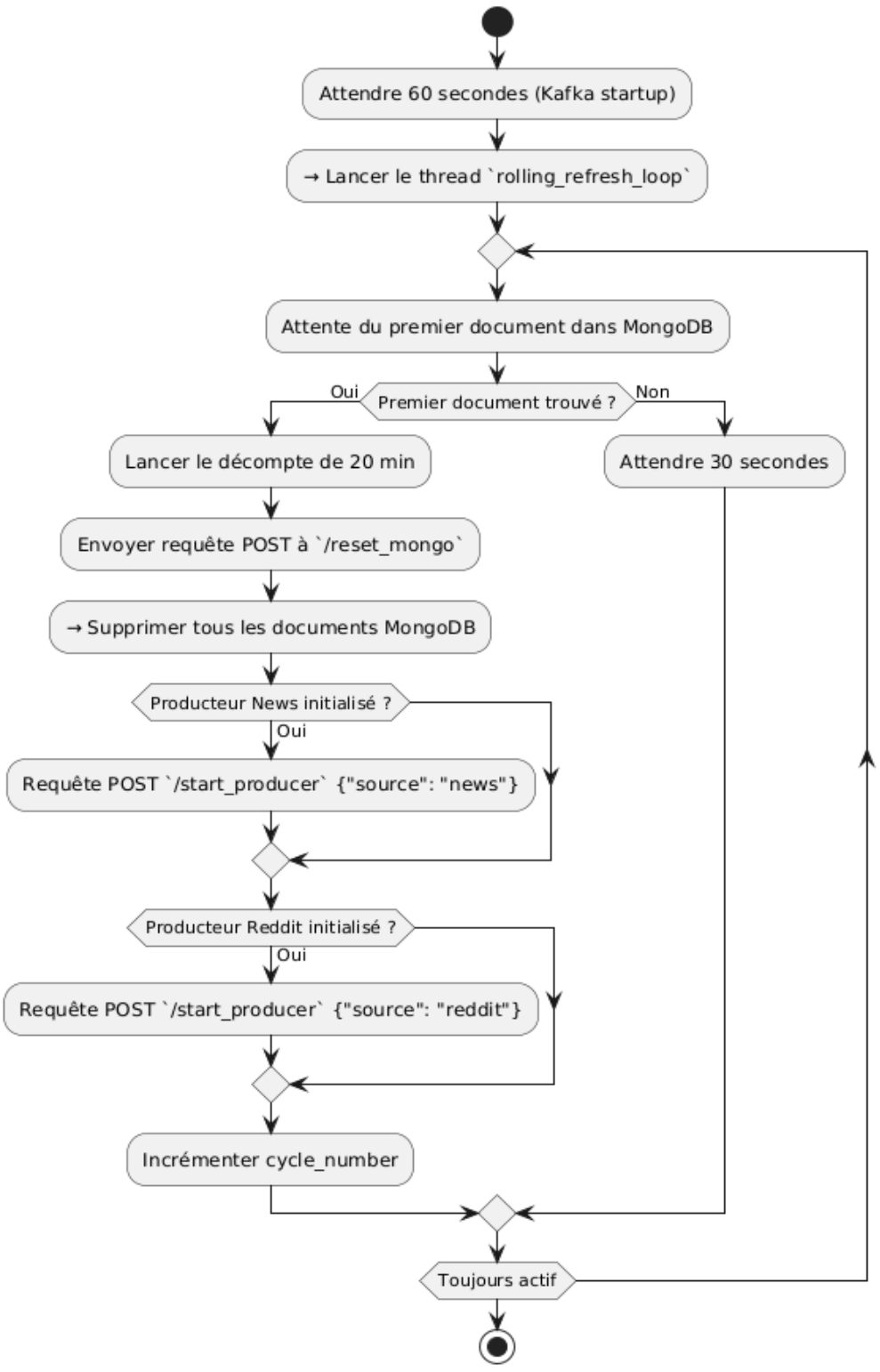

Figure 8 : Cycle de rafraîchissement automatique toutes les 20 minutes

2.7. Justification des choix technologiques

Le choix des technologies utilisées dans ce projet n'est pas arbitraire. Il résulte d'un équilibre entre les besoins du système en temps réel, la nature des données manipulées, les contraintes de performance et l'expressivité des outils disponibles. Cette section présente les principales raisons ayant guidé la sélection des composants clés de l'architecture.

2.7.1. Traitement de flux

Au début du projet, des solutions comme Faust ont été explorées. En théorie, Faust permet de consommer des messages Kafka tout en écrivant du code Python clair et asynchrone. Mais dans la pratique, son intégration avec des bibliothèques de NLP modernes — notamment les modèles de sentiment basés sur Transformers — s'est révélées problématique. Le framework semblait mal gérer les tâches computationnelles lourdes, provoquant parfois des ralentissements, voire des erreurs, difficiles à résoudre.

Quant à Kafka Streams et Spark Streaming, leur puissance n'est pas à remettre en cause. Mais leur complexité, leurs prérequis techniques (Java ou cluster Spark), et leur manque d'intégration native avec Python en faisaient des choix peu adaptés à un projet académique à taille humaine.

Ce choix de simplicité n'a rien d'une faiblesse. Bien au contraire : il a permis de concevoir un pipeline stable, clair et entièrement maîtrisé, capable de fonctionner en temps réel, tout en assurant la compatibilité avec les outils NLP les plus performants du moment.

Il ne s'agit pas de réinventer Kafka Streams ou Spark, mais de répondre efficacement à un besoin ciblé : traiter les données dès leur arrivée, appliquer une analyse linguistique avancée, et stocker les résultats pour une visualisation immédiate.

2.7.2. kafka

Apache Kafka a été choisi comme système de gestion des flux de données pour plusieurs raisons [46] :

- **Haute performance et faible latence :** Kafka est conçu pour gérer des flux massifs avec une latence minimale, ce qui est essentiel dans un contexte de surveillance en temps réel.

- **Tolérance aux pannes** : Kafka assure la réplication des messages entre brokers, garantissant ainsi la fiabilité même en cas de défaillance de nœud.
- **Stockage persistant :** contrairement à d'autres systèmes comme RabbitMQ ou Redis Streams, Kafka conserve les messages pendant une durée configurable, même après consommation. Cela permet d'avoir un historique tampon exploitable par d'autres consommateurs.
- **Scalabilité horizontale** : Kafka peut être étendu facilement pour gérer des volumes de données croissants, sans refonte majeure de l'infrastructure.
- **Écosystème mature :** Kafka dispose d'outils natifs pour le monitoring, le partitionnement, la gestion des offsets, etc., ce qui en fait un standard industriel pour le stream processing.

2.7.3. MongoDB

Le choix de MongoDB comme base de données repose sur sa nature documentaire et sa souplesse de modélisation [47] :

- **Schéma flexible :** MongoDB permet de stocker des documents aux structures variables, ce qui est idéal pour un système où les messages peuvent provenir de sources très différentes (Reddit, NewsAPI).
- **Requêtes puissantes :** grâce à son moteur d'agrégation, MongoDB permet d'effectuer des opérations complexes (filtrage temporel, regroupement par entité, top keywords...) directement au sein de la base, sans surcharge côté application.
- **Performance sur les lectures rapides :** une fois indexée, la base offre de très bons temps de réponse pour les requêtes nécessaires à la visualisation en direct dans Dash.
- **Facilité d'intégration avec Python :** MongoDB est bien supporté par des bibliothèques comme `pymongo`, ce qui simplifie son intégration dans un pipeline Python full-stack.

2.7.4. Choix de spaCy, VADER et Transformers dans le pipeline NLP

Le traitement sémantique repose sur une combinaison de trois outils complémentaires, chacun jouant un rôle précis dans l'analyse :

- **spaCy :**
  Utilisé principalement pour la reconnaissance d'entités nommées (NER**)**. C'est une

bibliothèque rapide, robuste et bien adaptée à des traitements en production. Elle offre également la tokenisation, le POS-tagging et d'autres outils utiles si le pipeline est étendu [48].

- **VADER :**
  Un outil léger pour l'analyse de sentiment basée sur un lexique. Il est très efficace pour les textes courts et informels (comme ceux issus de Reddit), et ne nécessite aucun entraînement. Son intégration est rapide et ses scores sont facilement interprétables.
- **Transformers (Hugging Face) :**
  Pour affiner l'analyse de sentiment, un modèle cardiffnlp/twitter-roberta-base-sentiment a été intégré. Il permet de comprendre le contexte global d'un message (ce que VADER ne peut pas faire), et fournit une classification plus fine, notamment dans des cas ambigus ou longs.

Le croisement des résultats entre ces outils permet de renforcer la robustesse des analyses et d'assurer une meilleure couverture des cas réels.

2.8. Conclusion

La conception du système repose sur des composants simples, modulaires et adaptés au traitement en continu. Les technologies choisies offrent un bon équilibre entre performance, flexibilité et intégration NLP. Cette base solide ouvre la voie à l'implémentation détaillée et à l'évaluation pratique dans le chapitre suivant.

Chapitre 3 : Implémentation, expérimentation et résultats

3.1. Introduction

Ce chapitre présente la mise en œuvre technique du système d'analyse de tendances en temps réel. Il décrit l'environnement de développement, les outils utilisés (Kafka, MongoDB, Dash, etc.), et l'organisation des fichiers du projet. Chaque module est expliqué en détail, depuis la collecte des données jusqu'à leur affichage graphique. Ensuite, les résultats obtenus sont analysés, avec des exemples concrets, notamment à travers une étude de cas sur un événement réel

3.2. Environnement de développement et configuration technique

3.2.1. Ressources matérielles et configuration locale

L'application a été développée et testée sur une machine locale dotée des caractéristiques suivantes :

- **-Processeur** : Intel Core i5 (7e génération)
- **Mémoire RAM** : 8 Go
- **Stockage** : SSD 512 Go

Le modèle Hugging Face utilisé (`cardiffnlp/twitter-roberta-base-sentiment`) nécessite l'installation de PyTorch et peut s'exécuter sur CPU ou GPU. Dans le cadre de ce projet, l'exécution s'est faite sur CPU, avec un temps de traitement moyen de 100 à 300 ms par message.

3.2.2. Stack technologique globale

Le projet repose sur une stack technologique moderne, orientée flux et traitement sémantique. Elle combine des outils robustes de collecte, de traitement et de visualisation :

- **Langage de programmation** : *Python 3.10*, choisi pour sa richesse en bibliothèques NLP et sa compatibilité avec Apache Kafka.

- **Framework web** : *Flask*, utilisé pour structurer l'application en tant que serveur principal et gérer les routes HTTP.
- **Interface de visualisation** : *Dash* (par Plotly), intégré dans Flask, permet de créer des tableaux de bord interactifs en Python pur.
- **Système de messagerie en continu** : *Apache Kafka*, utilisé comme bus de messages pour la communication entre producteurs de données et analyseurs NLP.
- **Base de données NoSQL** : *MongoDB*, sélectionnée pour sa flexibilité en matière de stockage de documents semi-structurés.
- **Outils NLP** : *spaCy*, *NLTK* (VADER), et *Hugging Face Transformers*.

3.2.3. Dépendances logicielles et bibliothèques Python

Le projet s'appuie sur plusieurs bibliothèques spécialisées dans le traitement de texte, la visualisation et la communication interservices :

Bibliothèque	Rôle
Flask	Création du serveur principal
Dash	Visualisation dynamique via composants HTML & graphiques
Pymongo	Connexion et requêtes vers MongoDB
Kafka-python	Communication avec Apache Kafka
Spacy	reconnaissance d'entités nommées
NLTK	Traitement linguistique et VADER pour l'analyse de sentiments lexicale
Transformers	Intégration de modèles pré entraînés (analyse de sentiments)
Pandas	Manipulation de données tabulaires pour les graphiques
Ploty	Génération de graphiques interactifs
Python-dateutil, uuid , threading	Fonctions utilitaires diverses (dates, UUID, threading, etc.)

Tableau 6 : Bibliothèques et Leurs Rôles

3.2.4. Configuration technique et infrastructure

Le système repose sur une architecture locale orientée microservices, avec les ports et paramètres suivants :

3.2.4.1. Kafka Broker

- Adresse : localhost:9092
- Topic principal : raw-news pour les messages bruts

3.2.4.2. MongoDB

- Port : 27017
- Base de données : trend_database
- Collection principale : analyzed-data

3.2.4.3. Dash / Flask

- Application accessible via : http://127.0.0.1:5000
- Communication avec MongoDB et les producteurs via appels internes ou threading

L'orchestration globale repose sur des modules lancés en parallèle. Pour assurer cette exécution simultanée :

- Le module app.py joue le rôle de coordinateur général.
- Les producteurs (reddit_producer.py, news_producer.py) fonctionnent via des threads.
- Le module nlp_analysis.py consomme les messages Kafka et les analyse en temps réel.

3.2.5. Organisation des fichiers sources

Le projet suit une organisation modulaire, facilitant la maintenance et l'extensibilité :

Fichier	Description
app.py	Lancement de l'application Flask + Dash, coordination des boutons et modules
dashboard_layout.py	Construction de l'interface utilisateur avec mise à jour dynamique des graphiques
reddit_producer.py	Collecte de messages depuis Reddit toutes les 30 secondes (hot posts)
news_producer.py	Récupération d'articles via l'API NewsAPI toutes les 60 secondes
nlp_analysis.py	Analyse NLP : nettoyage, tokenisation, sentiment (VADER + Transformers), NER
mongodb_connector.py	Fonctions d'accès à MongoDB pour les requêtes des graphiques

config.py	Paramètres de configuration centralisés (API keys, ports, chemins, etc.)
control_panel.html	Interface de commande graphique pour déclencher chaque étape du pipeline

Tableau 7 : Description des Fichiers du Projet

3.3. Implémentation détaillée des modules

Cette section décrit le rôle, le fonctionnement et les interactions de chaque module constituant l'architecture logicielle. L'ensemble est conçu selon une approche modulaire, facilitant la maintenabilité et l'extensibilité.

3.3.1. app.py – Serveur principal et coordinateur du système

Le fichier `app.py` joue un rôle fondamental dans l'architecture de l'application. Il combine les fonctions de serveur web Flask, de gestionnaire de threads pour les modules producteurs et de traitement, ainsi que d'hôte du tableau de bord interactif Dash. Ce module agit comme le chef d'orchestre du système, permettant une exécution fluide, modulaire et interactive du pipeline de traitement de tendances en temps réel.

3.3.1.1. Objectifs principaux

a. Serveur Flask – Interface de contrôle

Le serveur Flask gère les interactions utilisateur à travers des routes HTTP. Il permet notamment :

- L'affichage d'un panneau de contrôle HTML (/control) contenant des boutons pour lancer manuellement les différentes étapes du pipeline.
- L'exécution des producteurs de données (reddit_producer.py, news_producer.py) et du module d'analyse linguistique (nlp_analysis.py) via des appels AJAX déclenchés par ces boutons.
- La gestion des requêtes entre les composants backend (analyse) et frontend (visualisation).

b. Serveur Dash – Tableau de bord en temps réel

Le serveur Dash, intégré dans le même fichier, est accessible via la route /dashboard/. Il fournit :

- Une interface interactive composée de graphiques dynamiques.
- Une visualisation en temps réel des résultats issus du traitement NLP, extraits de MongoDB : évolution des sentiments, mots-clés dominants, entités nommées, messages récents, etc.
- Un rafraîchissement automatique des composants visuels (via dcc.Interval)

3.3.1.2. Fonctionnalités techniques

Fonctionnalité	Détail
Lancement unifié	Accès via http://127.0.0.1:5000, regroupant toutes les interfaces.
Exécution asynchrone	Chaque producteur et le module NLP sont lancés dans des threads indépendants (via threading.Thread), assurant la non-blocking execution.
Communication Kafka	Les producteurs envoient leurs données vers le topic Kafka raw-news, que le module NLP consomme ensuite.
Connexion MongoDB	Résultats enrichis stockés dans trend_database.analyzed-data, exploitables par le tableau de bord.
Modularité	Architecture facilement extensible : nouvelles sources de données ou nouveaux modules d'analyse peuvent être ajoutés sans perturber le système existant.

Tableau 8 : L'ensemble des Fonctionnalités technique du Système

3.3.2. Interface de contrôle centralisée (control_panel.html)

Afin de simplifier l'interaction avec le système, une interface de contrôle conviviale a été développée en HTML et rendue via Flask. Le fichier control_panel.html, accessible depuis la racine via /control, permet à l'utilisateur de :

- Lancer manuellement les services backend (Kafka, MongoDB)
- Démarrer les producteurs de données (reddit et news)
- Exécuter l'analyse linguistique (run nlp analysis)
- Accéder au tableau de bord de visualisation (open Dashboard)

Le bouton de lancement de l'analyse NLP doit être utilisé avant de démarrer les producteurs. Cela garantit que les messages produits seront analysés immédiatement après leur arrivée, et non ignorés comme données anciennes.

L'interface est divisée en trois sections fonctionnelles :

- Server Controls qui permet de lancer Kafka et MongoDB.
- Data Pipeline qui déclenche les producteurs Kafka et le module d'analyse.
- View Charts qui redirige vers /dashboard/, qui héberge l'application Dash contenant les graphiques.

Ce panneau de contrôle centralisé constitue un point d'entrée unique et ergonomique pour manipuler le système, sans avoir à exécuter chaque script manuellement en ligne de commande. Il est particulièrement utile pour les tests et démonstrations en temps réel.

3.3.3. reddit_producer.py – Producteur de données Reddit

Ce module utilise l'API Reddit (via PRAW) pour récupérer périodiquement les publications les plus populaires dans des subreddits prédéfinis (ex. : r/worldnews, r/technology).

3.3.3.1. Comportement

- Récupération toutes les 30 secondes
- Suppression des doublons basée sur l'ID et le titre
- Envoi vers le topic Kafka raw-news
- Rafraîchissement de la liste des subreddits toutes les 5 minutes

3.3.3.2. Étapes techniques

1. Authentification via client_id, client_secret, user_agent
2. Parcours de plusieurs subreddits en parallèle
3. Publication sur Kafka via KafkaProducer

3.3.4. news_producer.py – Producteur NewsAPI

Ce module interroge l'API publique NewsAPI.org avec une clé privée pour obtenir des articles récents.

3.3.4.1. Fonctionnalités

- Exécution toutes les 30 secondes
- Filtrage par langue (Fr ou En) et pertinence
- Requête basée sur une liste de mots-clés : ["économie", "politique", "environnement", etc.]
- Insertion dans Kafka sous forme JSON vers raw-news

3.3.4.2. Gestion

- Limitation de requêtes respectée (API rate limit)
- Nettoyage des doublons par URL de l'article

3.3.5. nlp_analysis.py – Analyse linguistique en temps réel

Ce module consomme les données de Kafka, les traites avec des outils NLP, puis les insère dans MongoDB.

3.3.5.1. Pipeline NLP

1. Nettoyage : suppression des URL, mentions, ponctuations
2. Tokenisation & stopwords : via NLTK
3. NER (Reconnaissance d'entités nommées) : via spaCy
4. Analyse de sentiment :
    - VADER (lexicale)
    - Hugging Face Transformers (cardiffnlp/twitter-roberta-base-sentiment)
5. Structuration JSON et insertion dans trend_database.analyzed-data

3.3.6. dashboard_layout.py – Interface graphique (Dash)

Ce module construit l'ensemble du tableau de bord Dash, en interaction avec MongoDB pour les données.

3.3.6.1. Composants

- Graphiques Plotly auto-rafraîchis
- Intervalles de mise à jour paramétrables (ex. toutes les 20 secondes)

- Appels aux fonctions de mongodb_connector.py pour récupérer les données

3.3.6.2. Graphiques inclus

- Line charts de sentiments (vader & hugging face)
- Bar chart de mots-clés
- Bar chart des entités nommées
- Derniers messages triés par source
- Distribution des sources (Reddit vs News)
- Lien les plus partagé

3.3.7 mongodb_connector.py – Couche d'accès aux données

Contient les fonctions de connexion et extraction depuis MongoDB avec pymongo.

3.3.7.1. Fonctions typiques :

- get_sentiment_over_time_hf()
- get_sentiment_over_time_vader()
- get_top_entities()
- get_top_keywords()
- get_recent_messages()
- get_top_sources()
- get_most_shared_links

Chaque fonction repose sur des pipelines d'agrégation MongoDB, optimisés pour une récupération rapide des statistiques.

3.3.8. config.py – Paramétrage centralisé

Contient tous les paramètres sensibles et réutilisables**,** comme :

- Clés API : reddit_client_id, newsapi_key
- Ports Kafka et MongoDB
- Topics utilisés (raw-news, etc.)

Cela garantit une modularité du système : aucun paramètre n'est codé en dur dans les modules de traitement.

3.4. Résultats et visualisation des tendances

3.4.1. Entités nommées les plus fréquentes

Le graphe ci-dessus présente les entités nommées les plus mentionnées dans les données collectées en temps réel à partir de Reddit et de NewsAPI. Ces entités ont été extraites à l'aide de l'outil de reconnaissance d'entités nommées (NER) de spaCy.

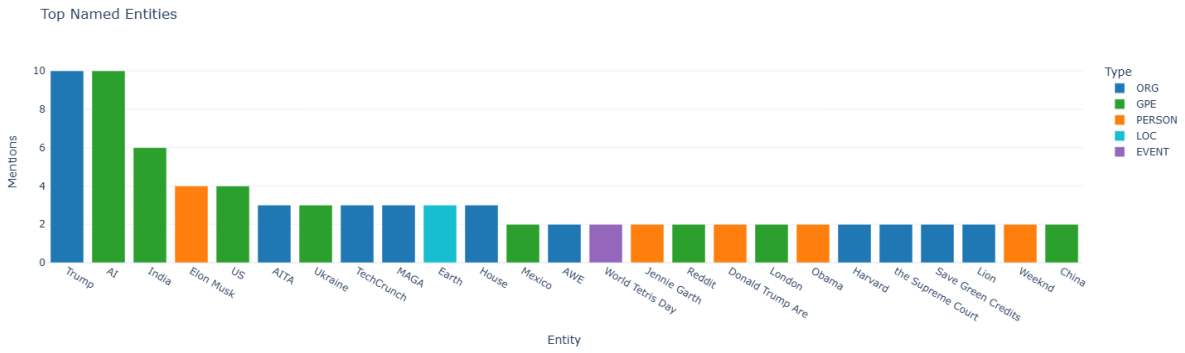

Figure 9 : Graphique à Barres des Entités Nommées les Plus Fréquentes par Type

- Trump et AI sont les deux entités les plus fréquemment mentionnées (10 fois), ce qui témoigne d'une forte présence de discussions politiques et technologiques dans les données collectées.
- D'autres personnalités publiques comme Elon Musk, Obama, Jennie Garth et Donald Trump apparaissent également, reflétant l'intérêt constant des utilisateurs pour des figures connues du monde politique et technologique.
- Des pays ou régions géographiques comme India, US, Ukraine, Mexico, China, et London sont également fréquents, ce qui montre que l'actualité internationale est bien représentée dans les contenus traités.
- Le graphe révèle aussi la présence de noms d'organisations et de médias comme TechCrunch, Reddit, Harvard, ainsi que des entités institutionnelles comme the Supreme Court.
- Notons la détection d'un événement : *World Tetris Day*, ce qui démontre la capacité du système à reconnaître non seulement des noms propres, mais aussi des événements ponctuels ou culturels.

Ces résultats suggèrent que le système peut efficacement identifier et quantifier les entités dominantes dans le flux d'actualité, ce qui est crucial pour la détection de tendances

thématiques (technologie, politique, international). Il offre une vue synthétique des sujets les plus discutés dans les sources surveillées.

3.4.2. Sentiment over time hf

Le graphe ci-dessus représente l'évolution du sentiment global détecté dans les messages collectés à l'aide du modèle cardiffnlp/twitter-roberta-base-sentiment. Chaque point du graphe correspond au nombre de messages classés comme positifs, neutres ou négatifs sur une période de temps (par minute).

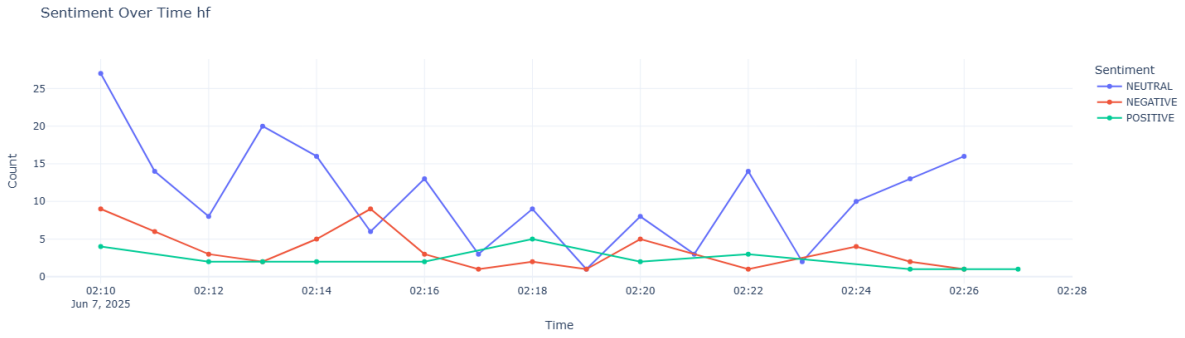

Figure 10 : Graphique Linéaire Évolution du Sentiment au Fil du Temps (Modèle Hf)

- On observe une dominance importante du sentiment neutre (courbe bleue), ce qui est typique dans les contenus d'information brute (ex. : titres d'articles ou posts informatifs sans opinion marquée).
- Le sentiment négatif (courbe rouge) est le deuxième plus fréquent, apparaissant de façon intermittente avec des pics aux alentours de 02:10, 02:16, 02:18 et 02:20. Cela pourrait être lié à la publication d'articles ou de posts évoquant des conflits, des scandales ou des critiques.
- Le sentiment positif (courbe verte) est relativement stable mais moins fréquent, ce qui est également courant dans les contextes d'actualités où les émotions positives sont généralement sous-représentées.
- Les variations soudaines (par exemple, le pic de neutralité à 02:10 puis une chute rapide) reflètent probablement l'arrivée en masse de nouveaux messages dans le flux analysé, ce qui démontre le bon fonctionnement du système en temps réel.

Ce graphe montre que la majeure partie du contenu analysé est de nature neutre, ce qui est cohérent avec des flux d'informations provenant de Reddit et de sites d'actualité. Il permet aussi d'identifier les pics de sentiment négatif ou positif**,** utiles pour la détection d'événements forts ou polarisants dans le flux d'information.

3.4.3. Sentiment over time vader

Le graphe ci-dessus illustre l'évolution du sentiment global détecté dans les messages collectés à l'aide du modèle VADER (Valence Aware Dictionary and sEntiment Reasoner). Chaque point représente le nombre de messages classés comme positifs, neutres ou négatifs par minute, sur une période temporelle.

Contrairement au modèle `cardiffnlp/twitter-roberta-base-sentiment`, qui repose sur des transformers pré-entraînés et affinés sur des données issues de Twitter, VADER est un lexique à base de règles spécifiquement conçu pour l'analyse de sentiment dans des contextes sociaux. Il est particulièrement léger, rapide et adapté aux textes courts comme les messages ou commentaires en ligne, mais moins sensible aux nuances contextuelles qu'un modèle basé sur l'apprentissage profond.

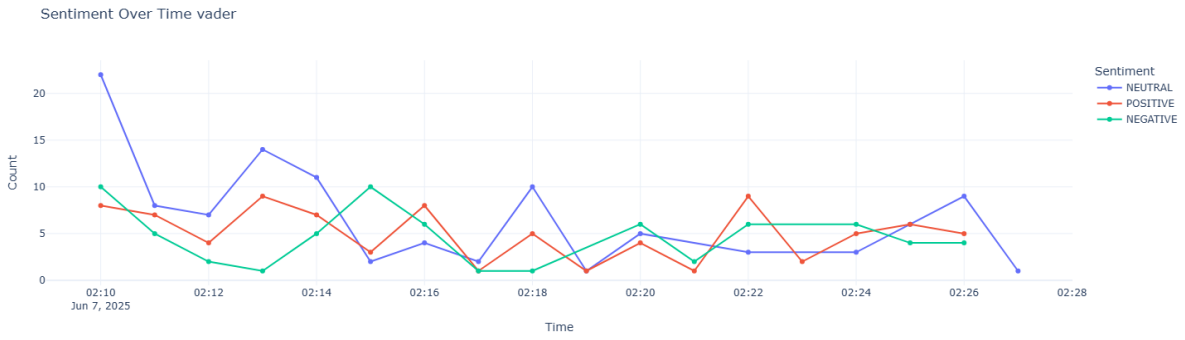

Figure 11 : Graphique Linéaire Évolution du Sentiment au Fil du Temps (Modèle VADER)

• On observe une domination nette du sentiment neutre (courbe rouge), avec un pic initial à plus de 20 messages vers 02:10, puis une chute rapide. Cela traduit probablement une forte densité de contenus factuels ou informatifs, typiques des forums comme Reddit.
• Le sentiment négatif (courbe bleue) apparaît de manière régulière avec des pics autour de 02:15, 02:18 et 02:20. Ces pics pourraient refléter des publications à caractère critique,

alarmiste ou polémique.

• Le sentiment positif (courbe verte) est plus modéré mais relativement stable, avec un pic remarqué à 02:22. Cette tendance est conforme aux observations habituelles sur les plateformes d'information, où les émotions positives sont souvent moins exprimées.

• Les variations brutales entre les segments temporels (comme la forte baisse des neutres après 02:10) reflètent une dynamique en temps réel du flux de données, mettant en évidence la réactivité du système d'analyse.

Ce graphique, produit avec le modèle VADER, révèle une structure globale des sentiments similaire à celle observée avec le modèle cardiffnlp/twitter-roberta-base-sentiment, à savoir une prévalence du neutre. Toutefois, VADER a tendance à surclasser les messages neutres et à être moins précis dans la détection des subtilités émotionnelles que les modèles de type transformer, qui intègrent un contexte linguistique plus riche.

Ainsi, l'usage de plusieurs modèles en parallèle permet de croiser les résultats et renforcer la fiabilité de l'analyse sémantique**.**

3.4.4. Mots-clés récurrents ou trending

Le graphe ci-dessus met en évidence les mots les plus fréquents extraits des articles et publications Reddit collectés pendant la période d'analyse. Il s'agit d'un diagramme à barres horizontales où l'axe vertical liste les mots-clés par ordre décroissant de fréquence, tandis que l'axe horizontal indique le nombre d'occurrences de chaque mot.

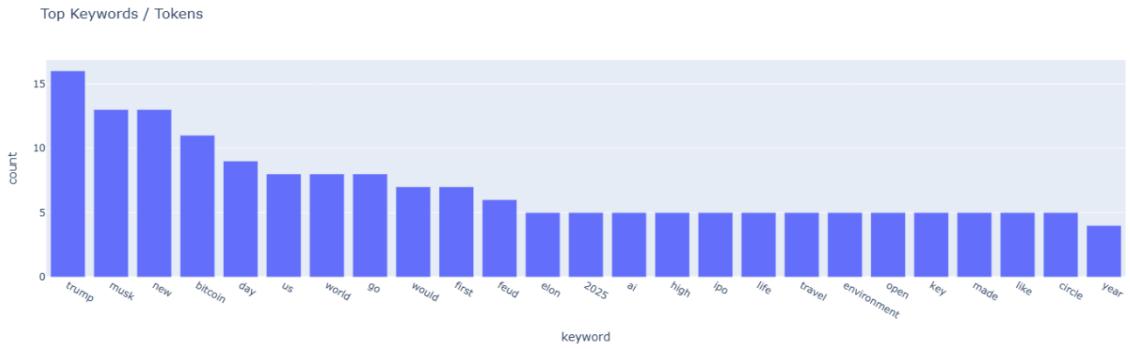

Figure 12 : Graphique à Barres de la Fréquence des Mots-clés / Tokens les Plus Pertinents

- "trump" est le mot le plus mentionné (~16 fois), ce qui indique un intérêt marqué pour les sujets politiques ou liés à l'actualité de Donald Trump.

- Des termes comme "musk" (13) et "elon" (5) témoignent d'une forte présence médiatique d'Elon Musk, souvent associée à l'actualité technologique.
- Des concepts tels que "bitcoin" (11), "ai" (5)**,** et "ipo" (5) révèlent un engouement pour les thématiques liées à la finance et à l'innovation technologique.
- Certains mots courants comme "new", "day", "us", "world", "go", "would" apparaissent fréquemment car ils sont polysémiques et utilisés dans de nombreux contextes.
- D'autres mots comme "feud" (5)**,** "travel" (5)**,** "environment" (5) ou encore "circle" peuvent être interprétés comme révélateurs de débats ou tendances spécifiques.
- On note également une redondance ou ambiguïté sur des termes proches comme "musk" et "elon"**,** ou encore une duplication du terme "ipo"**,** ce qui pourrait être corrigé par une normalisation plus fine en amont du traitement.

Ce graphe reflète les préoccupations, centres d'intérêt et événements les plus discutés dans les données collectées en temps réel. Il constitue un excellent indicateur des tendances thématiques, que ce soit dans les domaines politique, économique, technologique ou environnemental.

3.4.5. Fil d'actualités en direct – Kafka Stream

L'interface "Live Feed – Stream Kafka Messages" permet d'observer en temps réel les messages analysés par le pipeline NLP. Chaque message affiché contient :

- La source du message (ex. : *AskReddit*, *DW [English]*, *New York Post*, etc.)
- L'horodatage précis de réception et traitement
- Le sentiment prédit, grâce au modèle Hugging Face `cardiffnlp/twitter-roberta-base-sentiment`

L'interface se rafraîchit automatiquement toutes les quelques secondes pour afficher les 20 messages les plus récents**.**

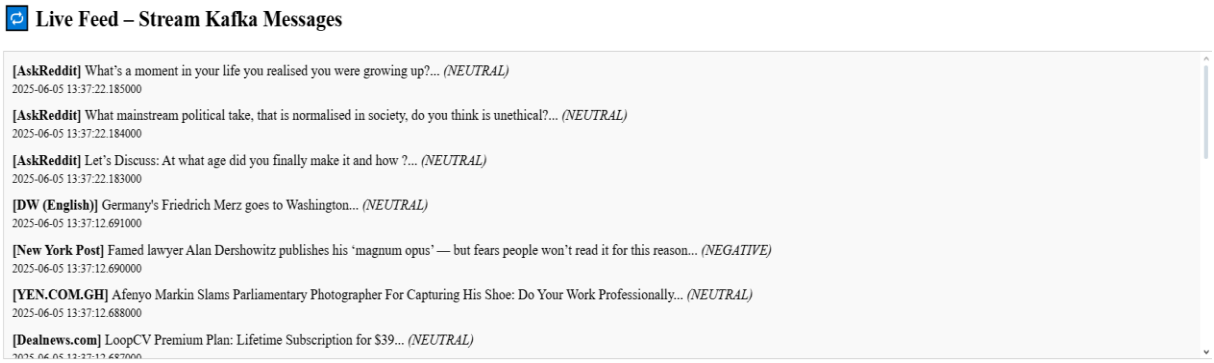

Figure 13: Flux en Direct - Messages Kafka

3.4.5.1. Exemples de messages affichés :

Source	Message (abrégé)	Sentiment	Horodatage
AskReddit	"What's a moment in your life you realised you were growing up?"	NEUTRAL	2025-06-05 13:37:22.185000
DW [English]	"Germany's Friedrich Merz goes to Washington"	NEUTRAL	2025-06-05 13:37:12.691000
New York Post	"Famed lawyer Alan Dershowitz publishes his 'magnum opus' — but fears people won't read it…"	NEGATIVE	2025-06-05 13:37:12.690000
YEN.COM.GH	"Afeyu Markin Slams Parliamentary Photographer…"	NEUTRAL	2025-06-05 13:37:12.688000
Dealnews.com	"LoopCV Premium Plan: Lifetime Subscription for $39."	NEUTRAL	2025-06-05 13:37:12.670000

Tableau 9 : Exemple de mesages affichés

Le but de cette interface c'est :

- Donne une visibilité immédiate sur les résultats de l'analyse sémantique en direct.
- Permet d'évaluer manuellement la pertinence des sentiments détectés par le modèle HF.
- Montre la variété des sources couvertes : médias, forums, plateformes d'actualités, etc.
- Utile pour capturer rapidement les tendances émergentes**,** particulièrement lors d'événements en temps réel.

3.4.6. Sources les plus actives (News & Reddit)

Ce graphique présente les 20 sources les plus actives en termes de volume de messages collectés et analysés par le système en temps réel. Les données proviennent à la fois de Reddit (en rouge) et de sources d'actualités en ligne (en bleu). La distinction visuelle entre les types permet d'observer la répartition entre contenu social et contenu journalistique.

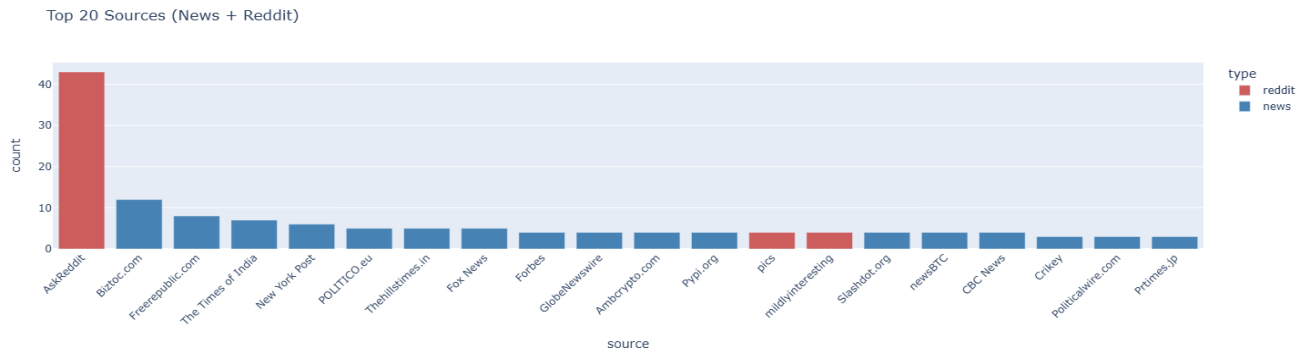

Figure 14 : Graphique a Barre de Top 20 des Sources (Actualités + Reddit)

- AskReddit est de loin la source la plus prolifique avec plus de 40 messages, ce qui s'explique par la nature participative et dynamique de Reddit.
- Du côté des sources d'actualités, Biztoc.com domine avec environ 11 à 12 publications, suivi par :
    - Freerepublic.com (8-9),
    - The Times of India, New York Post, POLITICO.eu, etc., chacun générant entre 5 à 7 messages.
- D'autres subreddits actifs incluent mildlyinteresting et pics, chacun avec environ 4 messages, montrant que le système ne se limite pas à un seul subreddit.

L'intérêt de cette visualisation c'est :

- Permet d'identifier les sources dominantes, utiles pour le filtrage, la pondération ou l'analyse ciblée.
- Donne une vue comparative entre la popularité des sources Reddit et celles des médias traditionnels.
- Utile pour évaluer la diversité des canaux couverts par le pipeline, et potentiellement détecter les biais de source.

3.4.7. Liens les plus partagés

Cette section recense les liens les plus fréquemment partagés dans les messages capturés depuis les différentes sources (Reddit, sites d'actualités, etc.). Bien que chaque lien n'apparaisse ici qu'une seule fois (avec un compte de `1`), cette visualisation est utile pour détecter les URL récurrentes ou virales**.**

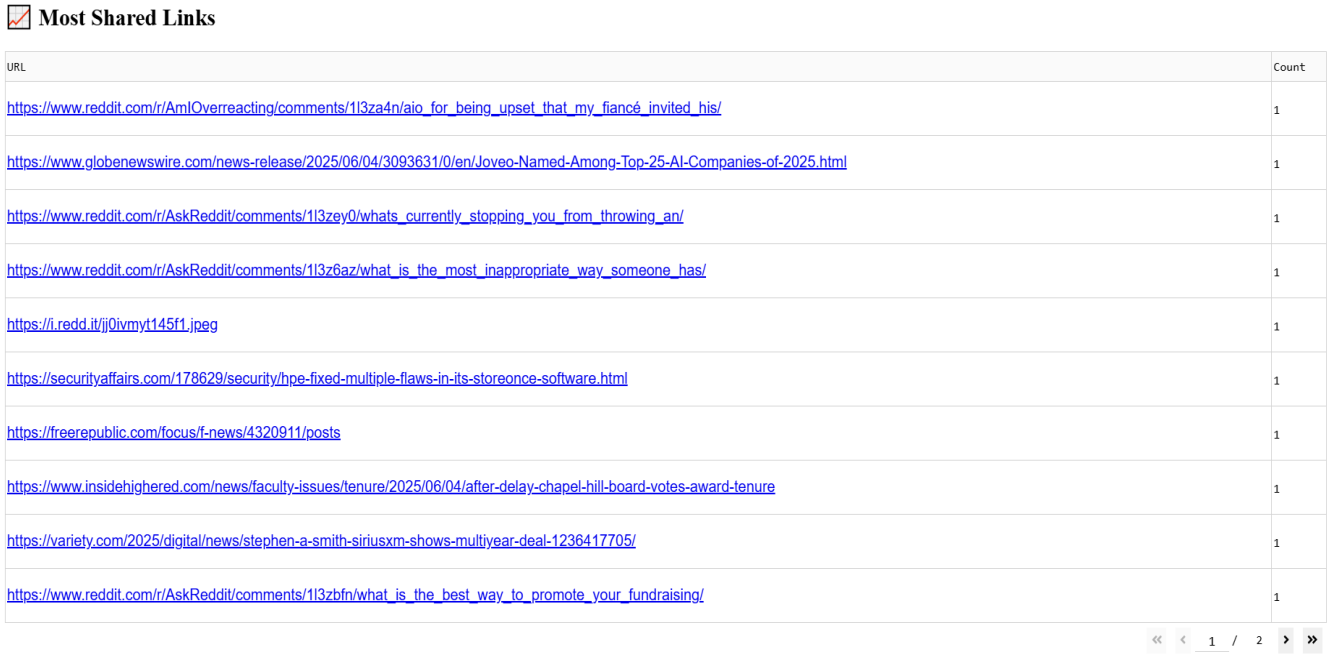

Figure 15 : Liens les Plus Partagés

Un total de 20 liens sont affiché dans cette section, répartis sur plusieurs pages grâce à un système de pagination dynamique. L'ensemble est mis à jour automatiquement à chaque insertion d'un nouveau message dans la base de données, garantissant ainsi une visualisation en temps réel des liens les plus récemment partagés.

L'intérêt de cette visualisation c'est :

- Permet de suivre les articles ou sujets viraux en temps réel.
- Peut servir de point de départ pour une exploration manuelle ou une validation qualitative du contenu.
- Utile pour repérer les sources d'information partagées fréquemment.

3.5. Cas réel analysé : Tendance "Trump – Musk"

Le 7 juin 2025, notre système de veille en temps réel a capturé une montée spectaculaire d'activité autour de la tendance "Trump – Musk", reflet fidèle d'un clash inattendu entre deux figures géantes : Donald Trump, Le président des États-Unis, et Elon Musk, patron de X (anciennement Twitter) et PDG de Tesla et SpaceX.
Si leurs noms figurent régulièrement dans les classements médiatiques, ce jour-là, la tension entre eux a littéralement enflammé les réseaux**.**

3.5.1. Volume de mentions détecté

Notre système a repéré un volume exceptionnel de mentions autour de ces deux figures dans les publications Reddit et les articles indexés par notre crawler :

- **"Trump"** : 16 occurrences
  Figure centrale du débat politique, son nom revient massivement, notamment dans les titres et accroches.
- **"Musk"** : 13 occurrences ; **"Elon"** : 5 occurrences
  Généralement associé à l'innovation, Elon Musk s'est retrouvé ici au cœur d'une controverse éminemment politique**.**

- **"feud", "epstein", "government contracts"**
  Ils traduisent les axes sensibles du clash : accusations, tensions morales, et argent public.
- **"ai", "bitcoin", "ipo"**
  Indices que le conflit a aussi débouchés sur des sujets économiques, technologiques et financiers, dans lesquels Musk est omniprésent.

3.5.2. Entités les plus évoquées (NER)

Grâce à notre module NER, nous avons identifié les figures et concepts dominants dans les échanges :

- • **Trump et l'IA :** 10 mentions chacun
  Un duo inattendu. Leur association révèle à quel point l'intelligence artificielle devient

un terrain de débat politique intense. Deux mondes qui se croisent, chacun avec ses visions, ses craintes, ses promesses.

- **• Elon Musk, Obama, US, China, Mexico**
  Des noms, des pays, des histoires. Ce mélange montre que le débat sur l'IA ou les grandes tendances technologiques ne connaît pas de frontières. C'est une conversation à l'échelle de la planète, nourrie par le passé autant que par l'avenir.
- **Événement annexe détecté** : World Tetris Day
  Exemple de la diversité des signaux détectés, et de la complexité du bruit médiatique.

3.5.3. Analyse du sentiment

Notre module d'analyse émotionnelle a révélé une dynamique affective très contrastée :

- Sentiment neutre dominant Majoritairement généré par des articles factuels ou des résumés automatisés.
- Pics de négativité marqués à 02:10, 02:16, 02:18, 02:20 , Coïncident avec :
  - Des publications virales signées Trump et Musk,
  - La diffusion d'un lien polémique sur Reddit,
  - Et les réactions en chaîne des communautés tech et politique.
- Sentiment positif : minoritaire, souvent dans :
  - Des commentaires ironiques,
  - Des messages de soutien humoristiques,
  - Ou des memes et détournements.

3.5.4. Lien le plus partagé : article de la BBC

Un article publié par la BBC Afrique, intitulé *« Trump vs Musk : que se passe-t-il lorsque l'homme le plus riche du monde et le politicien le plus puissant de la planète s'affrontent ? »*, a suscité un large écho sur les réseaux sociaux, notamment sur Reddit et X (anciennement Twitter).

L'article retrace la montée des tensions entre Donald Trump et Elon Musk, qui s'étaient jusqu'alors affichés comme des alliés stratégiques. Ce lien s'est rompu brutalement, après des critiques publiques et des menaces croisées : Trump évoquant la fin des contrats publics de SpaceX, Musk insinuant des révélations explosives à venir.

Le ton est grave, presque personnel, entre deux figures dont les décisions peuvent avoir un impact mondial. Ce face-à-face soulève des questions profondes sur les relations entre pouvoir politique et grandes entreprises technologiques. L'article a été massivement relayé, en grande partie parce qu'il capte un moment de rupture entre deux des hommes les plus influents du monde[1].

Les signaux captés ne sont pas de simples données brutes**.** Ils racontent une histoire humaine, sociale et émotionnelle. L'affrontement entre Trump et Musk a suivi une dynamique virale, amplifiée par :

- La vitesse de propagation sur les réseaux sociaux**,** sans modération immédiate,
- La puissance des communautés Reddit et Twitter, véritables catalyseurs,
- La réactivité des médias à sensation**,** toujours prompts à creuser les conflits.

La tendance Trump–Musk**,** captée le 7 juin 2025**,** constitue un cas d'école **:** Un affrontement entre pouvoir politique et influence technologique, aux conséquences économiques, sociales et médiatiques immédiates**.**

Grâce à l'analyse croisée de mots-clés, d'entités, de sentiments et de liens partagés, nous obtenons un thermomètre précis de l'opinion publique en ligne, et des signaux faibles qui annoncent parfois les prochaines grandes ruptures.

3.6. Analyse des performances des modèles NLP

3.6.1. Métriques techniques

Afin d'évaluer de manière rigoureuse les performances de notre pipeline NLP en contexte temps réel, nous avons procédé à une mesure systématique de la rapidité d'exécution pour chaque composant du traitement. L'objectif était de déterminer le temps moyen nécessaire au traitement d'un message, critère essentiel pour toute application en flux continu.
Pour cela, nous avons intégré un système de journalisation des performances dans le code de chaque composant. Ce système enregistre, pour chaque message traité :
• le temps de traitement,

[1] https://www.bbc.com/afrique/articles/cz0dj79j0d7o

• un horodatage (timestamp),
• et le nom du composant en cours (model_name).

Ces données ont ensuite été sauvegardées dans un fichier CSV nommé *performance_logs.csv***,** ce qui a permis une analyse statistique et visuelle via les bibliothèques *pandas* et *matplotlib***.**

3.6.1.1. Temps moyen de traitement par module NLP

Afin d'évaluer les performances de la chaîne de traitement en temps réel, une analyse comparative a été réalisée sur les durées de traitement des différents modules du pipeline : VADER**,** spaCy NER**,** Transformers (Hugging Face), et la TotalPipeline qui regroupe tous les traitements appliqués à un message depuis l'ingestion jusqu'à son stockage.

Le graphique ci-dessous présente la durée moyenne de traitement par message, mesurée en secondes, pour chaque modèle :

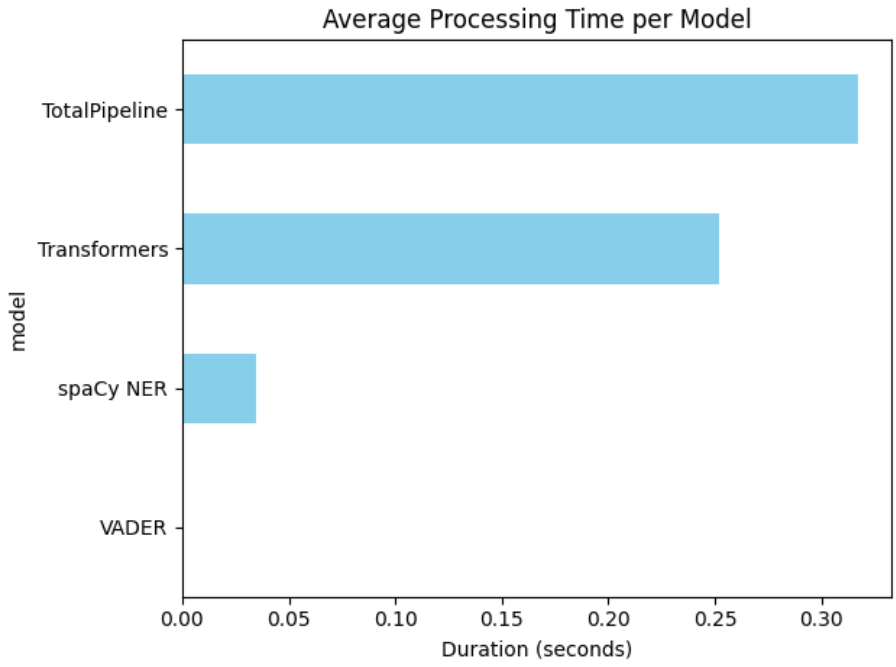

Figure 16 : Temps de Traitement Moyen par Modèle

On constate que :

- Le module VADER, utilisé pour l'analyse de sentiment rapide, est extrêmement rapide avec une durée moyenne quasi-négligeable (inférieure à 0,01 seconde).

- Le NER spaCy (reconnaissance d'entités nommées) affiche également une bonne performance, avec un temps de traitement moyen d'environ 0,04 seconde.
- Le module Transformers, bien que plus coûteux, reste raisonnablement rapide avec environ 0,25 seconde par message, ce qui reste acceptable dans un contexte de traitement quasi temps réel.
- Enfin, la TotalPipeline (regroupant tous les traitements successifs) affiche un temps moyen d'environ 0,33 seconde par message. Cela signifie que le système peut, en théorie, traiter environ 3 messages par seconde sans accumulation, ce qui est cohérent avec les objectifs de réactivité de l'application.

Cette évaluation permet de valider la faisabilité d'une application en temps quasi réel tout en soulignant l'intérêt d'optimiser les modèles utilisés (notamment les Transformers) si des performances encore plus élevées sont requises à grande échelle.

3.6.1.2. Messages traités par minute

Pour évaluer la capacité de notre pipeline à gérer un flux continu, nous avons mesuré le nombre de messages traités par minute, en nous basant sur les horodatages des documents insérés dans MongoDB. Le graphique ci-dessous montre une activité irrégulière entre 05:58 et 06:10, avec des pics au-dessus de 100 messages/minute, suivis de baisses soudaines à 5–8 messages/minute, avant de remonter.

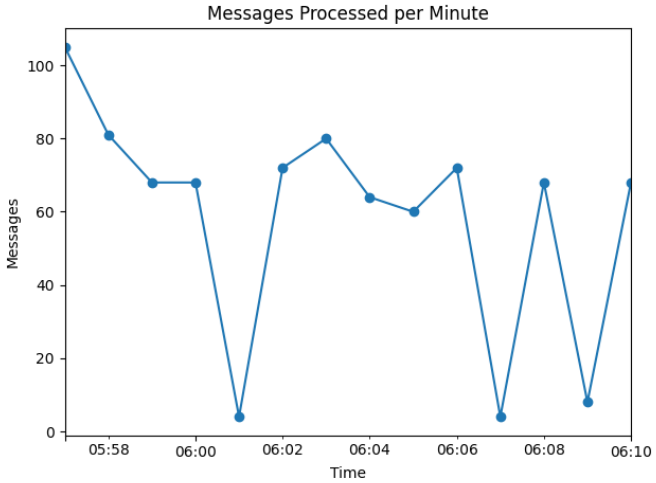

Figure 17 : Messages Traités par Minute

Ces variations s'expliquent principalement par :

- la charge temporaire des modules NLP (comme Transformers),
- et des temps de réponse variables des APIs externes.

Malgré ces fluctuations, le système montre une bonne capacité de récupération automatique, avec un retour rapide à un rythme soutenu (environ 70-80 messages/minute). Ce comportement est acceptable pour une application de suivi de tendances en temps réel.

3.6.1.3. Comparaison Temps / Qualité : VADER vs Transformers

Dans notre pipeline, deux modèles ont été utilisés pour l'analyse des sentiments : VADER et Transformers (Hugging Face)**.** Leur comparaison s'est appuyée à la fois sur les performances (vitesse) et la qualité de l'analyse**.**

Les mesures montrent que VADER est nettement plus rapide que Transformers. Il affiche en moyenne un temps de traitement de 0.003 secondes par message, contre environ 0.25 secondes pour le modèle Transformers, soit une différence d'un facteur 60x. Cette rapidité s'explique par la nature de VADER : un modèle basé sur des règles et des lexiques, beaucoup plus léger.

Cependant, cette rapidité se fait au détriment de la précision et du contexte. VADER donne des résultats acceptables pour des textes simples ou très courts (comme des titres ou tweets), mais montre ses limites sur :

- les textes ambigus ou nuancés,
- les formulations négatives complexes (ex : "ce n'est pas si mauvais"),
- et le langage ironique.

À l'inverse, Transformers (basé sur un modèle BERT fine-tuné pour la classification de sentiments) parvient à mieux capturer le contexte, les subtilités linguistiques et les tournures implicites. Il est donc plus fiable pour une analyse fine, notamment sur les articles de presse ou les discussions Reddit.

3.6.2. Robustesse du Système

La robustesse est une composante essentielle de toute application en temps réel. Dans notre architecture basée sur Kafka, MongoDB et des modules d'analyse NLP, plusieurs mécanismes

ont été envisagés pour faire face aux pannes potentielles et assurer une reprise fiable sans perte de données critique.

Si le broker Kafka tombe en panne [49] :

- Les producteurs ne pourront plus publier de messages, mais la plupart (comme notre reddit_producer ou news_producer) sont conçus pour réessayer automatiquement grâce à des boucles de temporisation et des exceptions gérées (try/except avec sleep).
- Les consommateurs comme nlp_analysis.py se reconnecteront automatiquement une fois le broker relancé, car Kafka conserve l'offset du dernier message traité.
- Aucune donnée n'est perdue : Kafka garantit la persistance des messages non consommés tant que leur délai de rétention (par défaut 7 jours) n'est pas dépassé.

Kafka assure une file d'attente tampon fiable, permettant une reprise fluide après redémarrage.

Si MongoDB devient indisponible pendant le traitement :

- Le module nlp_analysis.py lève une exception lors de l'écriture des résultats analysés.
- Une stratégie de journalisation locale peut être envisagée pour stocker temporairement les messages analysés non insérés.

En cas d'échec, aucun crash n'est fatal, et des logs permettent une reprise ciblée.

Notre système repose sur une architecture modulaire, ce qui facilite une reprise indépendante des composants :

- Kafka gère les files d'attente.
- Les scripts sont idempotents : redémarrer nlp_analysis.py ou dashboard.py ne provoque pas de doublons ni de conflits.
- Les producteurs peuvent être redémarrés sans perte de données, puisque chaque message reste dans Kafka jusqu'à sa consommation complète.

De plus, la base MongoDB peut être purgée régulièrement (ex : toutes les 20 minutes dans notre cycle expérimental), ce qui permet de tester la reprise avec un état propre.

Grâce à l'utilisation combinée de Kafka (buffer robuste), MongoDB (base NoSQL tolérante) et de scripts découplés, notre architecture assure :

- Tolérance aux pannes,
- Reprise rapide sans perte majeure,
- Et continuité de service même en environnement instable.

Des améliorations futures pourraient inclure :

- La mise en place de sauvegardes périodiques MongoDB**,**
- Et un service de supervision automatique pour redémarrer les composants en cas de défaillance.

3.6.3. Scalabilité

L'un des points forts de notre système est sa capacité à s'adapter facilement à l'ajout de nouvelles sources de données, sans avoir à modifier l'architecture globale. Chaque source (comme Reddit ou NewsAPI) est gérée par un producteur indépendant qui envoie ses messages vers Kafka. Cela signifie que si demain on souhaite intégrer une source comme YouTube Trends ou X (anciennement Twitter), il suffirait de créer un nouveau producteur dédié. Le reste du pipeline — traitement NLP, stockage et visualisation — restera inchangé.

Pendant les tests, deux sources ont été utilisées simultanément avec un bon rythme de collecte. D'après les mesures, notre système peut sans problème gérer une dizaine de sources en parallèle, à condition de disposer de suffisamment de ressources côté Kafka et traitement NLP.

Côté traitement, le script nlp_analysis.py peut être amélioré pour suivre la montée en charge. Il est aujourd'hui centralisé, mais peut être facilement parallélisé : soit en utilisant plusieurs threads ou processus, soit en le découpant en microservices (par exemple un pour la reconnaissance d'entités, un autre pour l'analyse de sentiment, etc.). Des technologies comme Faust ou Kafka Streams permettraient aussi de répartir les traitements automatiquement si besoin.

Enfin, pour le stockage, MongoDB est bien adapté à ce type de flux continu. Grâce à ses capacités de montée en charge horizontale (sharding), il pourra continuer à absorber des volumes importants de données sans perte de performance. De même, le tableau de bord Dash s'appuie sur des agrégations, ce qui permet de garder des temps de réponse rapides, même avec une base de données qui grossit.

Notre pipeline est modulaire, extensible et prêt à évoluer avec l'ajout de nouvelles sources ou une augmentation du volume des données.

3.7. Conclusion

Le système implémenté a montré qu'il était capable de traiter des données textuelles en continu de manière efficace. Grâce à son architecture modulaire et à l'utilisation de modèles NLP modernes, il offre une visualisation claire des tendances en ligne. Les résultats sont prometteurs, mais certaines limites techniques subsistent (scalabilité, précision, dépendance aux APIs). Ce travail ouvre ainsi la voie à des améliorations futures

Conclusion générale

Ce projet de fin d'étude a permis de concevoir et de mettre en œuvre une application complète de suivi des tendances en temps réel à partir de sources variées comme Reddit et NewsAPI. En combinant technologies de streaming (Kafka), analyse linguistique (NER, sentiment analysis) et visualisation interactive (Dash), nous avons atteint nos objectifs principaux. Le système repose sur une architecture modulaire intégrant MongoDB et Flask, et permet de collecter, traiter et visualiser dynamiquement les tendances en ligne à travers un tableau de bord riche et interactif.

Cependant, la réalisation du projet n'a pas été exempte de défis. La gestion de la volumétrie, les limites des APIs externes, et les contraintes de performance ont nécessité des ajustements constants, comme l'optimisation du pipeline NLP, l'introduction du cache, ou la réduction du batch size. Malgré ces obstacles, le système constitue une base solide pour de futures évolutions plus avancées.

Plusieurs pistes concrètes d'amélioration sont envisagées pour renforcer la robustesse, la scalabilité et l'utilisabilité de la plateforme. Parmi celles-ci :

- L'intégration de frameworks de traitement distribués comme Kafka Streams, Faust ou Spark Streaming permettrait de mieux gérer les flux massifs en temps réel et d'augmenter la tolérance aux pannes.
- L'ajout de nouveaux canaux de données tels que Twitter ou YouTube Trends élargirait la couverture des tendances et améliorerait la détection des signaux faibles.
- L'enrichissement de l'interface utilisateur par des filtres temporels, une recherche textuelle, des exports, ou un affichage multilingue rendrait la plateforme plus exploitable pour les analystes et journalistes.
- La suppression du cycle de réinitialisation des données toutes les 20 minutes au profit d'un stockage historique durable dans le cloud (MongoDB Atlas, Data Lake) ouvrirait la voie à des analyses longitudinales et comparatives.
- L'intégration d'un modèle de résumé automatique (comme T5, BART ou Pegasus-XSum) offrirait une lecture synthétique des tendances, en extrayant les idées clés des articles ou posts populaires.
- Un déploiement cloud conteneurisé (via Docker et CI/CD) garantirait une meilleure disponibilité, facilité de mise à jour et accessibilité multiplateforme.

En somme, ce projet pose les fondations d'un système intelligent de veille médiatique et d'analyse de discours en ligne, capable de détecter et suivre en temps réel les dynamiques informationnelles. Il offre une plateforme évolutive, à la croisée du NLP, de l'analyse de données et du journalisme augment.

Références

[1] Splunk, «Traitement de flux : définition, outils et défis,» 2024. [En ligne]. Available: https://www.splunk.com/fr_fr/blog/learn/traitement-de-flux.html.

[2] U. Ç. a. S. Z. M. Stonebraker, «The 8 Requirements of Real-Time Stream Processing,» *ACM SIGMOD Record,* vol. 34, n° %14, pp. 42-47, 2005.

[3] M. S. e. B. Selic, «Conception de logiciel en temps réel – Progrès actuels et défis à venir,» *IEEE Canada,* 2006.

[4] D. Tivelet, «La complexité de la scalabilité, réelle problématique ou challenge à relever ?,» 11 mars 2024. [En ligne]. Available: https://www.kaliop.com/fr/scalabilite-developpement-informatique/.

[5] ibm, «Qu'est-ce qu'une architecture orientée évènements,» 26 décembre 2024. [En ligne]. Available: https://www.ibm.com/fr-fr/topics/event-driven-architecture.

[6] T. A. a. S. C. a. R. Lax, «The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing,» *Proceedings of the VLDB Endowment,* vol. 8, n° %112, pp. 1792--1803, 2015.

[7] T. Lu, «batch-vs-stream-processing,» 15 aout 2024. [En ligne]. Available: https://www.datacamp.com/blog/batch-vs-stream-processing.

[8] M. U. Demirezen, «Performance Analysis of Lambda Architecture-Based Big-Data Systems on Air/Ground Surveillance Application with ADS-B Data,» 2023.

[9] «Kappa architecture,» 17 septembre 2024. [En ligne]. Available: https://www.synnada.ai/glossary/kappa-architecture.

[10] J.-F. SIMON, «Event-driven architecture - Comment se lancer avec Kafka ? (partie 1),» 29 juin 2021. [En ligne]. Available: http://blog.ippon.fr/2021/06/29/comment-se-lancer-avec-kafka-partie-1/.

[11] M. Jumelle, «Apache Kafka : tout savoir sur le Data Streaming,» 10 janvier 2022. [En ligne]. Available: https://blent.ai/blog/a/apache-kafka-tout-savoir-sur-le-data-streaming.

[12] J. A. Shaheen, «Apache Kafka: Real Time Implementation with Kafka,» *International Journal of Advanced Science and Technology,* vol. 109, pp. 35-42, 2017.

[13] a. amazon, «the-difference-between-rabbitmq-and-kafka,» [En ligne]. Available: https://aws.amazon.com/fr/compare/the-difference-between-rabbitmq-and-kafka/.

[14] d. alex, «Kafka vs Pulsar: Streaming data platforms compared,» 2023.

[15] E. d. team, «Kafka vs. Kinesis: How They Do Real-Time Analytics, Differences, Challenges, and Best Practices,» 7 juin 2024. [En ligne]. Available: https://edgedelta.com/company/blog/kafka-vs-kinesis.

[16] m. l. team, «What is Azure Event Hubs for Apache Kafka?,» 18 décembre 2024. [En ligne]. Available: https://learn.microsoft.com/en-us/azure/event-hubs/azure-event-hubs-apache-kafka-overview.

[17] P. Brebner, «Redis Streams vs Apache Kafka,» 16 févriér 2022. [En ligne]. Available: https://www.instaclustr.com/blog/redis-streams-vs-apache-kafka/.

[18] J. C. a. B. White, «Jumping NLP Curves: A Review of Natural Language Processing Research,» *IEEE Computational Intelligence Magazine,* vol. 9, n° %12, pp. 48-57, 2014.

[19] C. Stryker, «Qu'est-ce que le NLP (traitement automatique du langage naturel),» 11 aout 2024. [En ligne]. Available: https://www.ibm.com/fr-fr/think/topics/natural-language-processing#:~:text=Le%20NLP%20permet%20aux%20ordinateurs,learning%20et%20l%27apprentissage%20profond..

[20] H. J. Aleqabie, «A Review Of Text Mining Techniques,» *Iraqi Journal for Computer Science and Mathematics,* vol. 5, pp. 125-141, 2023.

[21] M. a. G. D. a. K. M. Mäntylä, «The Evolution of Sentiment Analysis - A Review of Research Topics, Venues, and Top Cited Papers,» *Computer Science Review,* vol. 27, 2016.

[22] B. Liu, «Sentiment Analysis: A Fascinating Problem,» *Sentiment Analysis and Opinion Mining,* pp. 1--8, 2012.

[23] C. H. a. E. Gilbert, «VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Tex,» *ICWSM,* vol. 8, n° %11, pp. 216-225, 2014.

[24] S. a. L. K. a. L. Z. a. Z. P. a. H. B. a. Z. A. a. L. J. Dai, «AI-based NLP section discusses the application and effect of bag-of-words models and TF-IDF in NLP tasks,» *Journal of Artificial Intelligence General science (JAIGS) ISSN:3006-4023,* vol. 5, n° %11, p. 13–21, 2024.

[25] M. B. S. J. a. I. K. H. Basri, «Enhancing Usability Testing Through Sentiment Analysis: A Comparative Study Using SVM, Naive Bayes, Decision Trees and Random Forest,» *JTSIA,* vol. 7, n° %14, pp. 1603-1610, 2024.

[26] g. jagadeesan, «A Comprehensive Technical Analysis of NLP Models for Sentiment Analysis: BERT, RoBERTa, DistilBERT, ALBERT, and XLNet,» linkedin, 19 aout 2024. [En ligne]. Available: https://www.linkedin.com/pulse/comprehensive-technical-analysis-nlp-models-sentiment-jagadeesan-0khgc.

[27] S. S. David Nadeau, «A survey of named entity recognition and classification,» *Lingvisticæ Investigationes,* vol. 30, n° %11, pp. 3-26, 2007.

[28] N. a. E. N. A. S. Sugiarta, «Location Named-Entity Recognition using Rule-Based Approach for Balinese Texts,» *Jurnal Elektronik Ilmu Komputer Udayana* , vol. 9, n° %13, pp. 435-442, 2021.

[29] J. R. a. E. A. A. Ponomareva, «Conditional Random Fields vs. Hidden Markov Models in a Biomedical Named Entity Recognition Task," Proceedings of the Recent Advances in Natural Language Processing,» *RANLP*, 2007.

[30] S. H. A. Abdulhady Abas Abdullah, «NER- RoBERTa: Fine-Tuning RoBERTa for Named Entity,» *arxiv*, 2024.

[31] R. Kassel, «spaCy : la bibliothèque Python Open Source de NLP,» 14 juin 2021. [En ligne]. Available: https://datascientest.com/spacy.

[32] github, «FlairNLP,» [En ligne]. Available: https://github.com/flairNLP/flair.

[33] MongoDB, «L'analyse en temps réel pour un meilleur engagement client,» [En ligne]. Available: https://www.mongodb.com/fr-fr/solutions/use-cases/analytics/real-time-analytics.

[34] R. Kassel, «MongoDB : tout savoir sur la base de données NoSQL orientée document,» 3 fevrier 2021. [En ligne]. Available: https://datascientest.com/mongodb.

[35] C. Williams, «Apache Cassandra vs Elasticsearch: Choosing a Vector Database for Your Needs,» 07 septembre 2024. [En ligne]. Available: https://zilliz.com/blog/apache-cassandra-vs-elasticsearch-a-comprehensive-vector-database-comparison.

[36] M. S. a. J. G. A. Kröll, «Visual Trend Analytics for Streaming Text Data,» chez *in Proceedings of the 20th International Conference on Information Visualisation (iV)*, Lisbon, Portugal, 2016.

[37] I. a. S. X. a. A. R. San Vicente, «Talaia: a Real time Monitor of Social Media and Digital Press,» 2018.

[38] P. Tripathi, «Stream twitter data to Elasticsearch with Python & dashboard with Kibana,» 11 septembre 2017. [En ligne]. Available: https://www.linkedin.com/pulse/stream-twitter-data-elasticsearch-dashboard-kibana-python-tripathi.

[39] «newswhip,» [En ligne]. Available: https://www.newswhip.com.

[40] M. Meriem, «LLM ou NLP ? Choisir la meilleure approche IA pour votre projet,» 21 mai 2025. [En ligne]. Available: https://www.stemapartners.com/blog/llm-ou-nlp-choisir-la-meilleure-approche-ia-pour-votre-projet.

[41] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems,* 1st ed., Sebastopol, CA, USA: O'Reilly Media, 2017.

[42] m. db, «JSON Databases Explained,» [En ligne]. Available: https://www.mongodb.com/resources/basics/databases/json-database.

[43] M. N. a. R. Z. a. O. Y. a. I. K. a. A. Chupaev, «Enhancing MongoDB query performance through index optimization,» *E3S Web of Conferences,* 2024.

[44] M. a. S. A. a. W. Y. a. A. M. Rahman, «RoBERTa-BiLSTM: A Context-Aware Hybrid Model for Sentiment Analysis,» *IEEE Transactions on Emerging Topics in Computational Intelligence,* vol. pp, pp. 1-18, 2025.

[45] «Quelles sont les meilleures techniques pour le nettoyage des données en temps réel,» [En ligne]. Available: https://www.linkedin.com/advice/0/what-best-techniques-real-time-data-cleaning?lang=fr&originalSubdomain=fr.

[46] G. Lawton, «Les bons et les mauvais cas d'usage d'Apache Kafka,» 19 mai 2020. [En ligne]. Available: https://www.lemagit.fr/conseil/Les-bons-et-les-mauvais-cas-dusage-dApache-Kafka.

[47] D. Ailyn, «Real-time Data Synchronization: Implementing real-time data synchronization between Python applications and MongoDB Atlas,» 2024.

[48] É. Blent, «SpaCy : comment l'utiliser sous Python?,» 12 juilliet 2022. [En ligne]. Available: https://blent.ai/blog/a/spacy-comment-utiliser.

[49] «Broker hors ligne et basculement du client,» [En ligne]. Available: https://docs.aws.amazon.com/fr_fr/msk/latest/developerguide/troubleshooting-offlinebroker-clientfailover.html.

[50] J. Romero, «Introduction au traitement automatique du langage,» paris.