Rapport - Projet NLP Text Mining

Application RUCHE

Romain BUONO, Yassine CHENIOUR,
Miléna GORDIEN-PIQUET, Anne-Camille VIAL

Master 2 SISE - Promotion 2025-2026

Responsable pédagogique : M. Ricco RAKOTOMALALA

11 janvier 2026

Résumé

Le projet RUCHE constitue un système intégré d'acquisition, de structuration et d'analyse d'offres d'emploi dans les domaines de la data science et de l'intelligence artificielle. Le système s'articule autour de quatre composantes principales : un ensemble de scrapers hétérogènes collectant des données depuis quatre plateformes majeures, une base NoSQL MongoDB assurant le stockage intermédiaire, un entrepôt de données MotherDuck structuré selon un modèle dimensionnel en étoile, et une application Streamlit multi-pages offrant des capacités de recherche sémantique et d'analyse géospatiale. L'architecture exploite des techniques avancées de NLP incluant la vectorisation sémantique par sentence-transformers, le filtrage par classification TF-IDF et régression logistique, ainsi que l'extraction assistée par modèles de langage. Le système agrège plusieurs milliers d'offres d'emploi géolocalisées et permet une exploration interactive du marché professionnel via des visualisations analytiques et cartographiques.

Table des matières

1 Introduction

Le présent rapport décrit l'ensemble des procédures, des stratégies méthodologiques et des dispositifs techniques mis en œuvre par le groupe n°6 de la promotion 2025-2026 du Master 2 SISE, composé des auteurs du rapport, dans le cadre du projet de NLP et de text mining réalisé sous la supervision de M. Ricco RAKOTOMALALA.

Le projet s'articule autour de plusieurs objectifs principaux : (i) la sélection et la constitution d'un corpus issu de plateformes en ligne de diffusion d'offres d'emploi ; (ii) la structuration et le stockage de ce corpus au sein d'un entrepôt de données reposant sur un système de gestion de bases de données libre ; et (iii) le développement d'une application connectée à cet entrepôt, destinée à l'analyse multidimensionnelle des annonces d'emploi, avec une attention particulière portée à la dimension géographique.

Afin d'en faciliter la lecture, le document adopte une organisation chronologique, fidèle aux différentes étapes du projet. La première section est consacrée à la justification du choix du corpus ainsi qu'aux modalités de son acquisition. La deuxième section traite des enjeux liés à la conception et à l'alimentation de l'entrepôt de données. La troisième section présente l'architecture de l'application développée, en mettant en lumière les problématiques spécifiques rencontrées lors de sa conception et de son implémentation. Enfin, une dernière section synthétise les principaux résultats obtenus, tant sur le plan technique que fonctionnel.

2 Corpus documentaire, Base NoSQL sous MongoDB avant le Data Warehouse

Afin d'assurer la cohérence avec la thématique de la formation, le périmètre de l'étude a été volontairement restreint aux métiers relevant des domaines de la data et de l'intelligence artificielle, dans leur dimension d'exploitation technique. Ainsi les métiers liés à ces technologies mais ne nécessitant pas une manipulation et/ou exploitation de celles-ci n'ont pas été retenus (par exemple : commercial, ressources humaines, etc.).

L'objectif général du projet étant de produire une représentation aussi fidèle et exhaustive que possible du marché professionnel concerné, il est apparu indispensable de s'appuyer sur des sources multiples et hétérogènes, afin de constituer un échantillon d'offres d'emploi suffisamment large et diversifié. Dans cette perspective, chaque auteur s'est vu confier l'exploitation d'une source spécifique et a développé une méthode de scraping adaptée aux particularités de celle-ci, tout en contribuant à l'alimentation automatisée d'une base commune NoSQL.

La suite de cette analyse vise à expliciter les raisons pour lesquelles le recours à la technologie NoSQL s'est imposé face à la diversité des sources mobilisées, avant de présenter l'architecture retenue ainsi que les enjeux techniques associés à chacune des méthodes de scraping mises en œuvre.

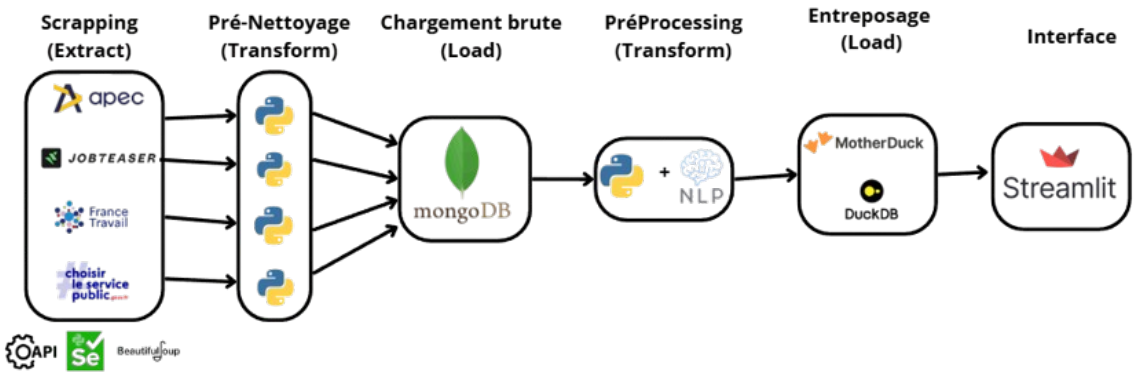

FIGURE 1 – Architecture globale du pipeline RUCHE, de l'extraction des données sources jusqu'à l'interface Streamlit

2.1 Le choix de l'alimentation d'une base de données NoSQL sous MongoDB

Pour rappel, les consignes initiales du projet prévoient que les données relatives au marché de l'emploi doivent être stockées et structurées dans un data warehouse hébergé sur un SGBD libre. Bien que des solutions technologiques existent pour satisfaire à ces

critères, la plupart, si ce n'est la totalité d'entre elles, imposent une limite maximale quant au volume de données stocké.

Cette limite peut s'avérer être un frein important dans les débuts du projet, attendu qu'il peut être plus qu'hasardeux de prédire le volume effectivement mobilisé par des données scrapées de sources diverses. En outre, la structuration d'un data warehouse, bien qu'elle se doive de répondre aux besoins métiers, est également conditionnée par le type et l'architecture des données qui l'alimentent.

Il est donc apparu dans ce contexte nécessaire de passer par une étape intermédiaire de stockage des données brutes, afin de pouvoir conserver une certaine souplesse dans les méthodes employées dans les phases initiales du projet et donner l'opportunité d'observer les données avant de se conformer à une structure posant le risque de ne pas correspondre à la réalité des données.

La caractéristique fondamentale d'une base de données NoSQL réside dans la flexibilité de son schéma : son adaptabilité aux différentes structures possibles d'un même set de données résultant des méthodes de scraping et à l'hétérogénéité des structures des différents documents. Ce format s'est donc imposé afin de faire cohabiter initialement les documents issus des procédures de scraping, sans avoir à définir de formalisme rigide.

Le choix de l'hébergement de cette base de données initiale s'est porté sur MongoDB en raison de ses fonctionnalités. En effet, la plateforme MongoDB est particulièrement adaptée au stockage de documents en format JSON, attendu que le stockage est réalisé en BSON (Binary JSON). Ce choix technologique permet de stocker directement les productions des procédures de scraping sans aucune transformation nécessaire préalable. Par ailleurs, le JSON permettant d'obtenir des listes ou bien des objets dans des objets, MongoDB gère nativement ces hiérarchies. Cette fonctionnalité permet de conserver la structure des éléments scrapés sans risque de perte d'information.

2.2 Quatre procédures de scraping

2.2.1 France Travail

Le site francetravail.fr propose une API officielle d'extraction des annonces de sa base de données. L'extraction de données à grande échelle à partir d'API publiques présente des défis structurels : latence réseau, limitations de débit (rate limiting) et hétérogénéité des données. Ce projet expose la conception d'un système de scraping "production-grade" capable d'automatiser la collecte d'offres d'emploi liées à l'Intelligence Artificielle, en optimisant le compromis entre performance et conformité aux politiques d'utilisation de l'API.

Le système est conçu selon une architecture de pipeline à trois niveaux, garantissant une séparation stricte entre l'authentification, la découverte et l'extraction.

Pipeline de traitement :

— Étage 1 (Authentification) : Implémentation du protocole OAuth2 pour la gestion des jetons d'accès.
— Étage 2 (Moteur de Recherche) : Pagination récursive et dédoublonnage par hachage avec une complexité temporelle de $O(1)$.
— Étage 3 (Scraper Parallèle) : Unité d'extraction granulaire gérant le filtrage par expressions régulières (RegEx).

Implémentation Technique

Pour pallier l'inefficacité du traitement séquentiel, nous avons implémenté un ThreadPoolExecutor. Le choix des fils d'exécution (threads) plutôt que des processus se justifie par la nature "I/O-bound" de la tâche, minimisant ainsi l'empreinte mémoire. La conformité avec l'API est assurée par un limiteur de débit basé sur l'algorithme de la fenêtre glissante (Sliding Window). Le débit maximum est défini par l'équation $T_{\max} = \min(N_{\text{workers}} \times \frac{1}{L_{\text{api}}}, \text{RateLimit})$, où L_{api} représente la latence réseau moyenne.

Le stockage s'appuie sur des dataclasses Python, assurant un typage fort et une sérialisation efficace vers le format JSON. Le filtrage sémantique utilise des frontières de mots pour garantir une précision accrue lors de l'identification des compétences clés (Data, IA, Machine Learning).

Analyse des Performances et Résultats

L'approche parallèle a démontré une amélioration substantielle du débit de données par rapport à l'approche séquentielle de référence. Pour 100 requêtes, le temps d'exécution est passé de 32,5 secondes en mode séquentiel à 11,1 secondes en mode parallèle avec 8 workers, représentant un gain de performance de 192%. Le débit a augmenté de 3,1 offres par seconde à 9,0 offres par seconde, soit un facteur multiplicatif de 2,9. Le taux de succès est resté constant à plus de 95% dans les deux configurations. Le filtrage post-extraction a permis d'affiner les résultats avec une précision de 63,6%, éliminant les faux positifs générés par le moteur de recherche natif de l'API.

2.2.2 APEC

Le scraper APEC adopte une architecture modulaire reposant sur Selenium WebDriver et BeautifulSoup, permettant une extraction structurée des offres d'emploi publiées sur la plateforme apec.fr. La stratégie d'acquisition se décompose en trois phases distinctes et séquentielles.

Dans un premier temps, le système procède à la collecte exhaustive des URLs d'offres en itérant sur les pages de résultats de recherche, avec application d'un filtre temporel strict limitant la collecte aux annonces publiées durant les trente derniers jours. Cette phase exploite les paramètres d'URL natifs de l'APEC, notamment le filtre anciennetePublication=101852, pour garantir la pertinence temporelle des données. Le système détecte automatiquement les pages vides consécutives et interrompt la pagination lors-

qu'un seuil configurable est atteint, optimisant ainsi le temps d'exécution. Pour chaque mot-clé fourni par l'utilisateur, une URL de recherche APEC est construite dynamiquement. Les pages de résultats sont chargées via Selenium, puis analysées afin d'identifier les liens vers les pages d'offres individuelles. Les URLs sont normalisées (suppression des paramètres superflus) et stockées dans une structure de type set, garantissant une déduplication efficace en temps constant. Un critère d'arrêt basé sur le nombre de pages consécutives sans nouvelles offres permet d'éviter les parcours inutiles.

Une fois l'ensemble des URLs collectées et dédoublonnées, la deuxième phase consiste en un scraping parallèle des pages détail, orchestré par un ThreadPoolExecutor permettant de traiter simultanément jusqu'à quinze offres. Chaque thread instancie son propre driver Chrome en mode headless, charge la page cible, extrait le contenu HTML via BeautifulSoup, puis isole le corps principal de l'annonce en supprimant les balises parasites. Selenium est utilisé pour charger la page complète dans un navigateur Chrome en mode headless, configuré pour limiter la consommation de ressources (désactivation des images, logs réduits, user-agent explicite). Une fois le rendu terminé, le HTML final est récupéré et analysé avec BeautifulSoup. L'extraction se concentre sur la balise principale, correspondant au contenu principal de l'offre. Les éléments non pertinents (scripts, styles, navigation) sont supprimés, puis le contenu est converti en texte brut.

Le système implémente un mécanisme de suivi de progression thread-safe avec affichage en temps réel des métriques de performance, incluant le taux de réussite, la vitesse de traitement et l'estimation du temps restant. Malgré l'optimisation par parallélisation, le scraping via Selenium demeure relativement lent, avec un débit observé d'environ 5,6 pages par minute pour un pool de threads.

Vingt-trois champs structurés ont été extraits via des expressions régulières, notamment le type de contrat, la localisation, les compétences techniques et comportementales, ainsi que les descriptions de poste et profils recherchés. Les motifs regex étaient spécifiques aux conventions typographiques de l'APEC, exploitant la structure prévisible des sections identifiées par des mots-clés comme "Descriptif du poste" ou "Compétences attendues". Cette approche, volontairement déterministe, permet un traitement rapide et reproductible, adapté à un volume important d'offres. Les résultats sont sérialisés au format JSON et stockés localement, avec génération automatique d'un nom de fichier intégrant un horodatage et les mots-clés de recherche utilisés, assurant ainsi la traçabilité complète du processus d'acquisition.

2.2.3 JobTeaser

Le scraper JobTeaser met en œuvre une stratégie d'extraction différenciée, recourant à undetected-chromedriver pour contourner les mécanismes anti-bot, notamment les protections Cloudflare déployées sur la plateforme jobteaser.com. L'architecture du système distingue rigoureusement deux niveaux d'extraction : la collecte des métadonnées depuis

les pages de liste et l'extraction approfondie depuis les pages détail.

La navigation s'effectue avec des paramètres géographiques ciblant spécifiquement la France, encodés dans l'URL de recherche sous forme de coordonnées GPS et d'identifiants de localisation. Le système itère sur les pages de résultats jusqu'à atteindre soit la limite de pages configurée, soit l'épuisement des résultats disponibles. Pour chaque carte d'offre présente sur la page liste, le scraper extrait un premier ensemble de données structurées incluant l'identifiant unique, le titre, l'entreprise, la localisation et le type de contrat, en s'appuyant sur des sélecteurs CSS spécifiques aux attributs data-testid utilisés par l'application React de JobTeaser.

Le système implémente un filtrage multi-critères avant d'engager la navigation vers les pages détail : il exclut les doublons par comparaison d'identifiants, applique un filtre temporel basé sur l'analyse de balises HTML time pour ne retenir que les offres respectant la limite d'ancienneté définie, et vérifie la pertinence du titre par correspondance avec un vocabulaire contrôlé de termes liés à la data. Cette approche de filtrage précoce permet de minimiser le nombre de pages détail à charger, optimisant ainsi la performance globale du scraper.

Les URLs éligibles sont accumulées dans une file d'attente locale avant déclenchement séquentiel des requêtes de détail, garantissant le respect des contraintes de pagination imposées. Sur chaque page détail, le système attend explicitement le chargement complet du DOM React via des WebDriverWait ciblant les éléments de contenu, puis extrait l'intégralité des champs disponibles par interrogation des sélecteurs CSS normalisés. La description complète de l'offre est récupérée depuis un conteneur article identifié par son attribut testid, et l'ensemble des données est consolidé dans une structure dictionnaire incluant les champs bruts ainsi que l'horodatage de scraping.

Un module effectue un post-traitement sur les textes extraits, notamment des nettoyages de caractères spéciaux, des normalisations de format de date, et une détection de compétences techniques et comportementales par recherche de motifs regex dans un corpus textuel assemblé. Les résultats enrichis sont exportés au format JSON avec séparation des fichiers bruts et enrichis, permettant une analyse différentielle des données.

2.2.4 Choisir le Service Public

Afin de ne pas restreindre l'analyse du marché de l'emploi sur les questions data et IA aux seuls employeurs de droit privé, une collecte des offres d'emploi proposées par le secteur public s'est imposée. Une première recherche a été initialement effectuée sur un ensemble de sites d'emploi public : emploipublic.fr, emploi-territorial.fr et emploi.fhf.fr. Afin de faciliter la prise en compte des différents formats de chaque portail par un seul et même programme de scraping, le recours à une méthode d'extraction sémantique assistée par modèles de langage (LLM) s'est imposé afin de bénéficier d'une souplesse d'adaptation du modèle à la structure.

Cependant, le portail choisirleservicepublic.gouv.fr s'est révélé par la suite une source plus fiable d'offres et il a donc été choisi comme source unique de scraping dédiée au secteur public. Le modèle ayant déjà été développé à ce stade, le recours au LLM a été maintenu, dans un souci d'optimisation. Le système mis en œuvre à partir du portail choisirleservicepublic.gouv.fr constitue un pipeline automatisé de collecte, de structuration et de stockage d'offres d'emploi du Service Public français. Il combine des techniques de web scraping à grande échelle, de traitement parallèle, et de LLM afin de produire des données normalisées, exploitables dans un contexte analytique ou de recherche en intelligence artificielle.

Le scraper ChoisirLeServicePublic se distingue par une approche hybride combinant web scraping classique et extraction assistée par intelligence artificielle, représentant une rupture méthodologique avec les stratégies conventionnelles de parsing HTML. Contrairement aux scrapers précédents, ce système n'utilise pas Selenium mais s'appuie exclusivement sur la bibliothèque requests pour les requêtes HTTP et BeautifulSoup pour le parsing initial, privilégiant ainsi la légèreté et la rapidité d'exécution.

La collecte s'effectue en deux temps : une phase de recensement des URLs d'offres par pagination séquentielle des pages de résultats filtrées par mot-clé, suivie d'une phase d'extraction parallélisée des contenus détaillés. Le système construit dynamiquement les URLs de recherche en intégrant les mots-clés ciblés dans la structure de chemin du site, puis détecte automatiquement le nombre total de pages à parcourir par analyse de la pagination HTML. Pour chaque page de résultats, les liens vers les fiches détaillées sont extraits et ajoutés à une liste globale, avec application optionnelle d'une limite maximale d'offres par mot-clé.

L'originalité du dispositif réside dans l'utilisation d'un grand modèle de langage via l'API Mistral, pour transformer le contenu HTML brut en données structurées JSON. Chaque page détail fait l'objet d'un prétraitement visant à isoler le contenu sémantiquement pertinent en supprimant les éléments de navigation, scripts et styles, puis en limitant intelligemment la taille du texte transmis à l'API par extraction des sections centrales. Le prompt fourni au LLM intègre un exemple de démonstration selon la méthodologie du few-shot learning, spécifiant explicitement le schéma de sortie attendu et les règles de formatage strictes. Ce prompt contraint le modèle à retourner exclusivement un objet JSON sans préambule ni balisage markdown, contenant l'ensemble des champs normalisés incluant l'identifiant, le titre, la collectivité employeur, le département, le type de contrat, le salaire et les compétences requises.

Le traitement est parallélisé via un pool de threads afin d'améliorer le débit global. Un mécanisme de rate limiting thread-safe est implémenté pour réguler les appels aux modèles de langage, garantissant le respect des quotas imposés par les APIs externes et la stabilité du système sous forte charge. Le système implémente un mécanisme robuste de limitation du débit d'appels API par fenêtre glissante temporelle, garantissant le respect des quotas

imposés par le fournisseur tout en maximisant le parallélisme via un ThreadPoolExecutor configuré avec quatre workers.

Les documents HTML collectés font l'objet d'un prétraitement systématique incluant le nettoyage structurel, la suppression des éléments non informatifs et le contrôle de la taille des contenus. Cette étape vise à maximiser le signal sémantique utile tout en minimisant le bruit transmis aux modèles de traitement du langage.

Cette stratégie hybride, bien que plus coûteuse en ressources API, permet de s'affranchir de la complexité du parsing HTML spécifique au site et offre une meilleure résilience face aux évolutions de structure des pages, le LLM étant capable d'adapter son extraction à des variations mineures de mise en forme.

3 Entrepôt de données sous MotherDuck

3.1 Uniformisation des variables de chaque collection de la base NoSQL avec des fonctions MongoDB dédiées

L'harmonisation des données issues de sources hétérogènes constitue un enjeu fondamental dans la constitution d'un entrepôt analytique cohérent. Pour répondre à cette problématique, le projet a développé une bibliothèque réutilisable de fonctions MongoDB centralisée dans le module mongodb_utils.py, permettant d'abstraire les opérations courantes d'insertion, de mise à jour et de gestion des index pour l'ensemble des collections de la base NoSQL. Cette approche modulaire garantit l'uniformité des procédures d'importation tout en permettant des adaptations spécifiques selon les particularités structurelles de chaque source. Le système implémente une stratégie de clés uniques différenciée selon la plateforme d'origine : pour les offres APEC, le champ reference_apec est systématiquement copié vers un champ id normalisé, tandis que pour JobTeaser, France Travail et ChoisirLeServicePublic, l'identifiant natif est directement exploité. Les scripts main_mongo.py et reference_apec.py intègrent une phase préalable d'analyse structurelle du fichier JSON source, détectant automatiquement la présence de doublons par comparaison des identifiants et générant des statistiques de validation avant tout import effectif. Cette phase de contrôle qualité permet d'identifier en amont les documents dépourvus d'identifiant unique, lesquels sont systématiquement exclus du processus d'insertion pour préserver l'intégrité référentielle de la base. Les opérations d'écriture exploitent le mécanisme de bulk_write avec des opérations UpdateOne paramétrées en mode upsert, permettant soit l'insertion de nouveaux documents, soit la mise à jour de documents existants sur la base de leur identifiant. Des index uniques sont créés sur les champs id et, pour APEC, également sur reference_apec, garantissant l'unicité au niveau du système de gestion de base de données et permettant des requêtes performantes lors des phases ultérieures de transformation. L'ensemble de ce dispositif assure une traçabilité complète du processus d'alimentation, avec génération de métriques détaillées incluant le nombre de documents insérés, mis à jour, dupliqués ou rejetés, ainsi que des statistiques sur la taille moyenne des documents et le volume total stocké dans chaque collection de la base RUCHE_nosql.

3.2 Choix de technologie MotherDuck, issu de DuckDB

Le choix de MotherDuck comme système de gestion de l'entrepôt de données repose sur les performances exceptionnelles de son moteur sous-jacent DuckDB pour le traitement analytique de données vectorielles de haute dimensionnalité. DuckDB offre un support natif des types de données array, permettant le stockage et la manipulation efficace de vecteurs d'embeddings de très grandes dimensions. Cette capacité s'avère déterminante

pour l'implémentation d'un système de recherche sémantique d'offres d'emploi, où chaque annonce est représentée par un vecteur dense capturant son contenu sémantique enrichi. Le moteur de requête columnar de DuckDB, optimisé pour les opérations vectorisées, permet d'effectuer des calculs de similarité cosinus directement au niveau du serveur via la fonction array_cosine_similarity, évitant ainsi le transfert massif de données vers le client et réduisant drastiquement les temps de réponse pour les recherches par similarité. Cette architecture de compute pushdown garantit que les calculs intensifs sur les embeddings sont exécutés là où résident les données, exploitant pleinement les capacités de parallélisation du moteur. L'extension cloud de DuckDB via MotherDuck apporte une dimension collaborative et évolutive au projet, permettant un accès concurrent aux données depuis plusieurs environnements de développement sans nécessiter de réplication locale complète de la base. La synchronisation automatique entre les instances locales et le cloud assure la cohérence des données tout en préservant les performances d'exécution, les requêtes étant optimisées selon qu'elles s'exécutent sur le cache local ou nécessitent un accès distant. Cette approche hybride combine la rapidité d'exécution de DuckDB avec la flexibilité et la persistance d'une solution cloud, particulièrement adaptée aux workflows de data science nécessitant à la fois des performances analytiques élevées et une infrastructure de stockage scalable. La compatibilité SQL complète de DuckDB facilite par ailleurs l'intégration avec les outils d'analyse existants, tout en offrant des fonctionnalités avancées comme la lecture directe de fichiers Parquet, la gestion de transactions ACID, et l'exécution de requêtes complexes sur des volumes de données conséquents sans dégradation notable des performances.

3.3 Machine Learning permettant de filtrer les annonces non-data/IA

La problématique du filtrage des offres d'emploi relevant effectivement des domaines de la data science et de l'intelligence artificielle nécessite une approche plus sophistiquée qu'une simple recherche lexicale par mots-clés. Les termes "data", "IA" ou "analytics" apparaissent fréquemment dans des annonces de postes périphériques tels que les fonctions commerciales, de ressources humaines ou de santé, générant un taux significatif de faux positifs lorsque seules des expressions régulières sont employées. Pour répondre à cette limitation, le système déployé implémente une stratégie hybride combinant des règles expertes sous forme de patterns regex avec un modèle de classification supervisé basé sur l'algorithme de régression logistique et la représentation vectorielle TF-IDF des textes. Le processus débute par une phase de labellisation semi-automatique où quinze patterns de whitelist identifient les postes clairement liés à la data engineering, data science ou business intelligence, tandis que quinze patterns de blacklist excluent les métiers de santé, comptabilité, commercial ou gestion administrative. Cette double filtration génère un en-

semble d'entraînement supervisé composé des offres labellisées, répartis majoritairement entre des postes data et minoritairement avec des postes non-data, révélant un déséquilibre de classes traité par application de poids équilibrés lors de l'entraînement du modèle. Les offres restantes non détectées par les patterns regex, constituent l'ensemble de prédiction où le modèle doit démontrer sa capacité de généralisation. Le vectoriseur TF-IDF transforme le texte combiné de chaque offre, incluant titre, description et fonction, en une représentation numérique capturant l'importance relative de chaque terme dans le document par rapport au corpus global, avec extraction de bigrammes pour capturer les expressions composées comme "data scientist" ou "machine learning". Le modèle de régression logistique, entraîné sur ces vecteurs TF-IDF, atteint des performances remarquables avec un F1-Score de 0.978, une ROC-AUC de 0.996 et une précision de 99,6% sur la classe data, validant la robustesse de l'approche. L'analyse des coefficients du modèle révèle que le terme "commercial" possède le poids négatif le plus fort, permettant l'identification correcte de postes ambigus comme "Business Developer Grands Comptes – SaaS & IA" qui, malgré la présence du terme "IA" dans le titre, est classifié comme non-data avec une probabilité de 97,5%. Cette capacité de discrimination contextuelle, absente des approches purement lexicales, permet de récupérer 1766 offres data supplémentaires non détectées par les regex, représentant un gain de 67,6% par rapport à une stratégie exclusivement basée sur les patterns. Pour une compréhension approfondie de la méthodologie employée, des fondements mathématiques du TF-IDF et de la régression logistique, ainsi que de l'analyse détaillée des cas d'usage limites, le document technique notice_TFIDF_ML_filtre_data_nondata.md est annexé au présent rapport (voir Annexe A).

3.4 Architecture de l'entrepôt de données

L'architecture de l'entrepôt de données adopte un modèle dimensionnel en étoile, paradigme éprouvé pour les systèmes d'aide à la décision et l'analyse multidimensionnelle. Cette structure se compose d'une table de faits centrale f_offre entourée de tables de dimensions dénormalisées permettant une navigation intuitive dans les différentes facettes des offres d'emploi.

La table de faits f_offre constitue le noyau du système et stocke les métriques et attributs granulaires de chaque offre, incluant les champs job_id comme clé primaire, source_platform et source_url pour la traçabilité, scraped_at pour l'horodatage d'acquisition, title, description, company_name et company_description pour le contenu textuel, nb_annees_experience et experience_required pour les prérequis professionnels, salaire, is_teletravail, hard_skills, soft_skills, langages, education_level, job_function et job_grade pour les caractéristiques détaillées du poste, ainsi que is_duplicate et similarity_score pour le marquage des doublons détectés par analyse sémantique. Les vecteurs

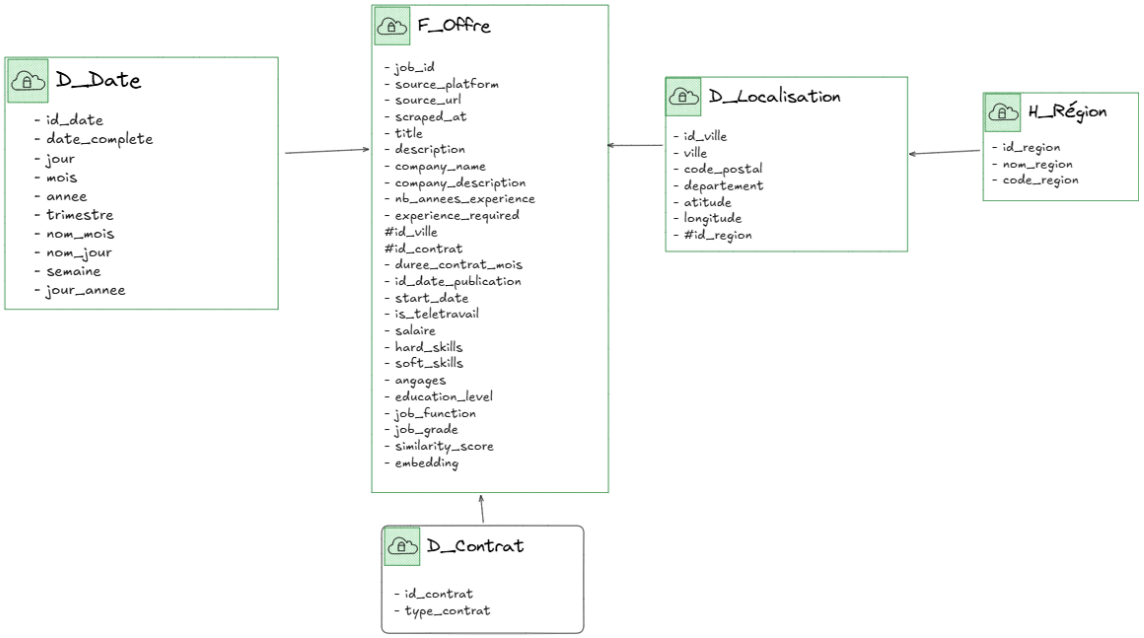

FIGURE 2 – Schéma en étoile de l'entrepôt MotherDuck avec la table de faits f_offre et les dimensions d_date, d_contrat, d_localisation et h_region

d'embeddings de 768 dimensions sont stockés dans le champ embedding de type FLOAT array, permettant les calculs de similarité cosinus directement en SQL.

La table de faits référence quatre tables de dimensions via des clés étrangères garantissant l'intégrité référentielle du modèle. La dimension d_contrat contient les attributs id_contrat comme clé primaire et type_contrat comme libellé normalisé, permettant de classifier les offres selon qu'il s'agit de CDI, CDD, stage, alternance, freelance ou intérim. La dimension d_date décompose temporellement les dates de publication selon les attributs id_date, date_complete, jour, mois, annee, trimestre, nom_mois, nom_jour, semaine et jour_annee, facilitant les analyses de saisonnalité, de tendances mensuelles ou de comparaisons interannuelles. La dimension d_localisation structure l'information géographique avec les champs id_ville, ville, code_postal, departement, latitude, longitude et id_region, permettant à la fois des analyses spatiales fines au niveau communal et des agrégations départementales ou régionales via la jointure avec la table de hiérarchie. La table de hiérarchie h_region complète le dispositif en fournissant la correspondance entre id_region, nom_region et code_region, permettant les agrégations territoriales de haut niveau selon les treize régions françaises.

Cette architecture en étoile optimise les performances des requêtes analytiques en minimisant le nombre de jointures nécessaires, les tables de dimensions étant directement accessibles depuis la table de faits centrale, tout en permettant une extension future du modèle par ajout de nouvelles dimensions sans refonte structurelle majeure. La dénormalisation assumée des tables de dimensions privilégie la simplicité des requêtes et la rapidité d'exécution au prix d'une redondance contrôlée des données de référence, compromis clas-

sique et justifié dans les architectures décisionnelles orientées lecture intensive.

3.5 Script d'alimentation et d'ETL

Le pipeline d'alimentation de l'entrepôt MotherDuck s'articule autour d'un script ETL exhaustif orchestrant la collecte, la transformation et le chargement des données depuis les quatre collections MongoDB brutes vers le schéma en étoile normalisé. La première phase d'extraction établit une connexion avec MongoDB Atlas et itère sur les collections apec_raw, francetravail_raw, servicepublic_raw et jobteaser_raw pour construire un DataFrame pandas unifié, chaque document étant enrichi d'un identifiant unique job_id et d'une référence à sa plateforme d'origine. Cette consolidation gère l'hétérogénéité structurelle des sources en mappant les champs variables vers un schéma cible harmonisé, par exemple en normalisant les multiples représentations possibles du titre de poste, de la description, du nom d'entreprise ou du type de contrat selon les conventions spécifiques de chaque plateforme.

La phase de transformation applique une série de normalisations critiques pour assurer la cohérence sémantique et analytique des données. Les types de contrats subissent une standardisation via la fonction normalize_contract_type qui regroupe les variantes lexicales en macro-catégories canoniques, extrayant également la durée contractuelle en mois pour les CDD via l'analyse par expressions régulières de patterns temporels. Les dates de publication sont harmonisées vers le format ISO 8601 par la fonction normalize_publication_date, gérant les multiples formats rencontrés tels que "JJ/MM/AAAA", "DD MMM YYYY" ou les références relatives comme "il y a X jours". Les noms d'entreprises font l'objet d'un nettoyage typographique supprimant les caractères spéciaux, les espaces multiples et normalisant la casse via normalize_company_name.

La localisation constitue un défi particulier résolu par un processus d'extraction et d'enrichissement en plusieurs étapes : extraction du nom de ville depuis les chaînes hétérogènes de localisation, déduction du code postal et du département, puis géocodage via l'API du référentiel géographique français pour obtenir les coordonnées GPS latitude/longitude, et enfin attribution de la région administrative via un mapping exhaustif département-région couvrant l'ensemble du territoire métropolitain et la Corse. Un module de détection de doublons basé sur la similarité cosinus des représentations TF-IDF compare chaque paire d'offres dont le score de similarité dépasse un seuil paramétrable de 0,9, marquant les documents redondants avec un flag is_duplicate et conservant le score de similarité pour permettre des analyses post-hoc de la qualité de dédoublonnage.

L'intégration du filtre de machine learning s'effectue à ce stade via l'appel à la fonction filter_data_jobs_ml, laquelle applique le modèle de régression logistique entraîné pour exclure les offres non pertinentes du périmètre data/IA, réduisant le dataset de 32,7% en éliminant les faux positifs détectés par apprentissage automatique. La phase de char-

gement construit progressivement le schéma en étoile en générant d'abord les tables de dimensions avec attribution d'identifiants séquentiels uniques, puis en peuplant la table de faits en résolvant les clés étrangères vers les dimensions préalablement créées. Des validations de qualité sont exécutées pour comptabiliser les valeurs manquantes substituées par des entrées "UNKNOWN" dans les dimensions, garantissant que chaque ligne de la table de faits possède une jointure valide même en présence de données partielles. Le script se conclut par l'exécution de requêtes analytiques de vérification produisant des distributions par plateforme source, par type de contrat, par région géographique et par entreprise, validant la cohérence du chargement et la disponibilité immédiate des données pour les analyses exploratoires et la construction des visualisations de l'application Streamlit.

4 Application Streamlit

4.1 Rappel des contraintes imposées de l'application

L'architecture technique de l'application RUCHE impose des contraintes structurelles fondamentales dictées par les choix technologiques effectués en amont du développement. La première contrainte majeure réside dans le stockage des vecteurs d'embeddings directement au sein de l'entrepôt de données MotherDuck, sous forme de colonnes de type FLOAT array de dimension 768, correspondant à la sortie du modèle sentence-transformers paraphrase-multilingual-mpnet-base-v2. Cette architecture impose que l'ensemble des calculs de similarité sémantique soient exécutés côté serveur via des requêtes SQL natives exploitant la fonction array_cosine_similarity intégrée à DuckDB, garantissant une stratégie de compute pushdown optimale où les opérations vectorielles intensives sont déléguées au moteur de base de données plutôt qu'au client Streamlit.

Cette approche contraint l'application à formuler des requêtes SQL complexes combinant jointures multiples entre la table de faits f_offre et les tables de dimensions d_localisation, d_contrat, d_date et h_region, tout en intégrant dynamiquement les clauses de filtrage utilisateur et le calcul de distance vectorielle dans une unique instruction SQL. L'utilisation du schéma en étoile impose par ailleurs une normalisation stricte des filtres appliqués, chaque critère de recherche devant être traduit en prédicats SQL compatibles avec la structure dimensionnelle, par exemple en mappant les sélections utilisateur de régions vers des jointures sur la hiérarchie h_region ou les types de contrat vers des comparaisons sur d_contrat.type_contrat.

La contrainte de connexion à MotherDuck via un token d'authentification sécurisé nécessite une gestion rigoureuse des variables d'environnement et impose l'utilisation de décorateurs @st.cache_resource pour partager efficacement les connexions entre les différentes pages de l'application multi-pages Streamlit, évitant ainsi la multiplication coûteuse des établissements de connexion. L'ensemble de ces contraintes techniques converge vers un impératif de performance où chaque interaction utilisateur doit minimiser les transferts de données entre le cloud MotherDuck et le client Streamlit, privilégiant systématiquement les agrégations et calculs serveur au détriment d'opérations Python côté application, tout en maintenant des temps de réponse inférieurs à la seconde pour préserver une expérience utilisateur fluide malgré le traitement de plusieurs milliers d'offres vectorisées.

4.2 Architecture de l'application et moteur de recherche sémantique

L'application Streamlit adopte une architecture multi-pages structurée autour de sept modules fonctionnels distincts orchestrant l'ensemble des capacités analytiques et inter-

actives du système RUCHE.

Le module central 1_home_page.py implémente le moteur de recherche sémantique intelligent permettant aux utilisateurs de formuler des requêtes en langage naturel, le système encodant dynamiquement la requête textuelle en un vecteur de 768 dimensions via le modèle sentence-transformers, puis exécutant une recherche par similarité cosinus directement dans MotherDuck pour retourner les offres les plus pertinentes triées par score de correspondance sémantique décroissant. Cette recherche s'enrichit de multiples filtres combinatoires incluant la sélection géographique multi-régions via multiselect, des checkboxes pour les types de contrat, et un slider de score minimum de pertinence permettant d'ajuster finement le seuil de similarité acceptable, l'ensemble étant traduit en une requête SQL unique intégrant les clauses WHERE appropriées et le tri par array_cosine_similarity descendant avec limitation TOP-K configurable. L'interface présente les résultats sous forme de cartes stylisées affichant le titre cliquable renvoyant vers l'URL source, le nom de l'entreprise, la localisation enrichie ville-région-code postal, le type de contrat, le score de similarité arrondi en pourcentage, un aperçu tronqué de la description avec possibilité d'expansion, et les compétences techniques requises, le tout rendu en HTML personnalisé avec bordures dorées et effets de survol pour une expérience visuelle soignée.

Pour une compréhension approfondie des fondements mathématiques et de la méthodologie employée dans le moteur de recherche sémantique, incluant la stratégie d'enrichissement contextuel par jointures SQL, le choix du modèle multilingue MPNet, la formulation algébrique de la similarité cosinus, et l'analyse comparative des performances, le lecteur est invité à consulter le document technique détaillé notice_moteur_recherche_semantique.md annexé au présent rapport (voir Annexe B).

Le module 2_cartographie.py propose une visualisation géospatiale interactive exploitant la bibliothèque Folium avec plugin MarkerCluster pour regrouper dynamiquement les offres par densité géographique, chaque marqueur circulaire étant dimensionné logarithmiquement proportionnellement au nombre d'offres et coloré selon un gradient heatmap, les tooltips de survol révélant la ville et le nombre d'offres tandis que les popups de clic exposent jusqu'à trente annonces détaillées avec liens directs vers les sources.

Le module 3_visualisation.py construit un tableau de bord analytique présentant des métriques clés sous forme de KPIs, des graphiques Plotly Express illustrant les salaires moyens par intitulé de poste avec barres horizontales colorées, une visualisation scatter des salaires par région, un tableau des cinq compétences les plus demandées, et un diagramme sunburst révélant la répartition hiérarchique des compétences par métier standardisé.

Le module 4_add_offers.py fournit une interface de saisie manuelle permettant l'ajout contrôlé d'offres via un formulaire exhaustif collectant titre, description, entreprise, localisation, contrat, salaire, date de début, compétences techniques et comportementales, langues, avec validation des champs obligatoires, normalisation ETL des entrées via les fonctions réutilisables du pipeline, génération automatique d'identifiants UUID, résolu-

tion des clés étrangères vers les dimensions, détection préventive de doublons par calcul de similarité TF-IDF avec blocage au-delà d'un seuil de 0.9, et insertion transactionnelle dans la table de faits avec commit explicite.

Le module 5_Clustering.py applique une réduction dimensionnelle UMAP sur les embeddings pour projeter les vecteurs de 768 dimensions en espace tridimensionnel, suivie d'un clustering HDBSCAN identifiant automatiquement les groupes d'offres sémantiquement cohérentes, chaque cluster étant labellisé par extraction des trois compétences techniques les plus discriminantes calculées par ratio de fréquence locale sur fréquence globale, et visualisé via un scatter plot 3D Plotly interactif coloré par label de cluster.

Le module 6_Graphe_competences.py construit un graphe de co-occurrences exploitant NetworkX pour modéliser les relations entre compétences techniques, chaque paire de compétences apparaissant conjointement dans une offre générant une arête pondérée par le nombre de co-occurrences, le graphe étant filtré par seuil minimal configurable et disposé spatialement via un algorithme spring layout tridimensionnel avant rendu Plotly avec nœuds dimensionnés selon le degré de centralité.

Enfin, le module 7_LLM.py expose un chatbot conversationnel alimenté par l'API Mistral permettant la structuration automatique d'offres d'emploi en format JSON normalisé, l'utilisateur soumettant une description textuelle brute que le modèle de langage parse selon un schéma strict défini dans le prompt système, générant un objet JSON conforme au schéma de la table de faits avec validation des types, gestion des valeurs manquantes, et possibilité de téléchargement direct du fichier structuré pour insertion ultérieure.

4.3 Choix des analyses et visualisations

Les choix analytiques et visuels opérés dans l'application RUCHE reflètent une volonté de conjuguer pertinence métier, exhaustivité des dimensions explorées, et accessibilité cognitive pour des utilisateurs non-techniques. L'analyse salariale constitue un axe central matérialisé par trois visualisations complémentaires répondant à des questionnements distincts mais convergents sur la valorisation financière des profils data et IA.

Le graphique en barres horizontales des salaires moyens par intitulé de poste standardisé permet d'identifier immédiatement les métiers les plus rémunérateurs, les intitulés ayant préalablement subi un clustering sémantique via embeddings UMAP-HDBSCAN pour regrouper les variantes lexicales désignant des rôles équivalents, évitant ainsi la fragmentation artificielle entre "Data Engineer", "Ingénieur Données" et "Data Engineering Lead" qui, bien que distincts syntaxiquement, relèvent de familles professionnelles comparables. Le scatter plot des salaires par région révèle les disparités géographiques de rémunération, chaque point étant dimensionné proportionnellement au salaire moyen pour accentuer visuellement les écarts, facilitant l'identification rapide des régions premium

comme l'Île-de-France ou des zones à opportunités intermédiaires. Ces deux représentations exploitent des agrégations SQL calculant les moyennes pondérées après parsing et normalisation des chaînes salariales hétérogènes via expressions régulières extractant les bornes numériques, convertissant les notations type "45k-50k€" en moyennes d'intervalles numériques exploitables.

L'analyse des compétences adopte une approche à deux niveaux combinant fréquence brute et contexte métier. Le tableau des cinq compétences techniques les plus demandées fournit une vision panoramique du marché en comptabilisant les occurrences globales après explosion des listes de compétences stockées sous forme de chaînes délimitées, permettant d'identifier rapidement les technologies incontournables comme Python, SQL ou Spark. Le diagramme sunburst enrichit cette analyse en introduisant la dimension métier, révélant quelles compétences sont spécifiquement associées à quels rôles standardisés, par exemple en montrant que TensorFlow apparaît massivement dans les offres Data Scientist mais marginalement dans les postes Data Analyst, information critique pour orienter les stratégies de montée en compétences. Cette visualisation exploite une double agrégation SQL avec GROUP BY imbriqués sur le couple titre standardisé-compétence, comptant les occurrences conjointes, puis filtrant sur les sept compétences globalement les plus fréquentes pour garantir la lisibilité du graphique circulaire hiérarchique.

Le module de clustering offre une exploration libre de l'espace sémantique des offres, permettant aux utilisateurs d'ajuster interactivement les hyperparamètres UMAP et HDBSCAN via sliders pour explorer différentes granularités de regroupement, chaque cluster étant automatiquement labellisé par calcul de scores TF-IDF inversés identifiant les termes surreprésentés localement relativement à leur distribution globale, générant des étiquettes synthétiques comme "Python/SQL/Pandas" ou "Cloud/AWS/DevOps" capturant l'essence thématique du groupe.

Le graphe de co-occurrences des compétences matérialise les synergies technologiques en révélant quelles compétences apparaissent fréquemment ensemble dans les offres, par exemple en montrant une arête forte entre Docker et Kubernetes ou entre SQL et Python, information précieuse pour anticiper les besoins de formations combinées. La construction de ce graphe exploite la fonction itertools.combinations pour générer toutes les paires de compétences au sein de chaque offre, incrémentant un compteur de co-occurrences filtré ensuite par seuil minimal pour ne conserver que les associations statistiquement significatives, le graphe résultant étant disposé en trois dimensions via spring layout de NetworkX et rendu interactivement en Plotly permettant rotation, zoom et inspection des nœuds.

Les métriques KPI affichées en en-tête des pages (nombre total d'offres, salaire moyen, salaire médian, nombre de villes, nombre de régions) fournissent des repères quantitatifs immédiats contextualisant les analyses détaillées, tandis que les top 4 régions et top 4 villes avec pourcentages relatifs orientent l'attention vers les bassins d'emploi dominants.

L'ensemble de ces choix visuels privilégie systématiquement les graphiques interactifs Plotly permettant survol, zoom et exploration plutôt que les rendus statiques, tout en maintenant une cohérence chromatique à travers l'application via palettes de couleurs coordonnées et une hiérarchie visuelle claire distinguant données primaires, secondaires et contextuelles.

4.4 Structuration et fonctionnalités de la partie cartographique

Le module cartographique constitue une pièce maîtresse de l'application RUCHE en matérialisant spatialement la distribution géographique des opportunités professionnelles data et IA sur le territoire français, exploitant les capacités de la bibliothèque Folium pour générer une carte interactive Leaflet embarquée dans l'interface Streamlit.

La stratégie de requêtage SQL sous-jacente repose sur une agrégation spatiale regroupant les offres par triplet ville-latitude-longitude via clause GROUP BY, calculant pour chaque localisation le nombre d'offres avec fonction d'agrégation COUNT ainsi que la collecte exhaustive des attributs individuels via fonctions ARRAY_AGG sur les champs job_id, title, company_name, type_contrat, salaire et source_url, permettant de reconstituer côté application la liste complète des offres attachées à chaque point géographique tout en minimisant le nombre de lignes retournées et donc la bande passante consommée. Cette requête intègre systématiquement des jointures LEFT JOIN vers les tables de dimensions d_localisation pour récupérer les coordonnées GPS et le département, h_region pour obtenir le nom de région, d_contrat pour le type de contrat, et d_date pour la date de publication, l'ensemble des filtres utilisateur étant traduits en clauses WHERE conditionnelles construites dynamiquement selon les sélections actives.

Le système de filtrage multi-critères combine sept checkboxes pour les types de contrat, un radio button pour treize fourchettes salariales prédéfinies, un second radio button pour cinq intervalles temporels de publication, deux multiselects pour sélection multiple de compétences techniques et de fonctions métier, chaque filtre générant une clause SQL additionnelle restreignant le périmètre des offres agrégées, par exemple en ajoutant AND c.type_contrat IN ('CDI', 'CDD') lorsque plusieurs types de contrat sont cochés ou AND d.date_complete >= CURRENT_DATE - INTERVAL '21 days' pour filtrer sur les trois dernières semaines. Les filtres sur compétences et fonctions exploitent les capacités de recherche full-text de DuckDB via l'opérateur LIKE pour détecter la présence de termes sélectionnés dans les champs textuels délimités, la requête générant dynamiquement des clauses du type AND (f.hard_skills LIKE '%Python%' OR f.hard_skills LIKE '%SQL%') pour implémenter une logique OU entre les compétences choisies.

L'affichage cartographique exploite le plugin MarkerCluster de Folium pour regrouper automatiquement les marqueurs proches selon le niveau de zoom, chaque cluster affichant un cercle numéroté indiquant le nombre d'offres regroupées, le code JavaScript personna-

lisé appliquant un gradient de taille et de couleur selon trois seuils définissant les clusters small, medium et large, le tout stylisé via classes CSS pour un rendu visuel cohérent. Les marqueurs individuels adoptent la forme CircleMarker avec rayon calculé logarithmiquement proportionnel au nombre d'offres agrégées pour éviter l'occultation mutuelle tout en conservant une échelle perceptible, la couleur suivant un gradient heatmap tricolore passant du marron pour les faibles densités au rouge puis au jaune pour les fortes concentrations, permettant d'identifier visuellement en un coup d'œil les métropoles attractives.

Chaque marqueur expose deux niveaux d'information interactive : un tooltip léger apparaissant au survol affichant ville, département, région et nombre d'offres dans une infobulle compacte, et un popup détaillé activé au clic présentant jusqu'à trente offres individuelles dans un conteneur scrollable avec en-tête sticky affichant les métadonnées de localisation, chaque offre apparaissant dans une carte miniature avec titre tronqué à cinquante caractères, entreprise limitée à trente caractères, type de contrat, salaire, et bouton vert cliquable redirigeant vers l'URL source dans un nouvel onglet, le popup indiquant explicitement le décompte des offres additionnelles si le total dépasse la limite d'affichage de trente.

Cette architecture permet de naviguer efficacement dans un dataset de plusieurs milliers d'offres géolocalisées sans surcharger l'interface, le clustering dynamique masquant la complexité sous-jacente aux niveaux de zoom élevés montrant la France entière tout en révélant progressivement le détail granulaire lors du zoom sur une région ou une ville spécifique. L'en-tête statistique de la page affiche trois métriques KPI calculées en temps réel sur le dataset filtré : nombre total d'offres sommé sur l'ensemble des villes, nombre de villes distinctes comptées après agrégation, et nombre de régions uniques, complétées par deux listes top 4 présentant les régions et villes les plus prolifiques en termes d'offres avec pourcentages relatifs facilitant la comparaison quantitative. L'ensemble du module bénéficie d'un système de cache Streamlit avec décorateur @st.cache_data et TTL de dix minutes permettant de mémoriser les résultats de requêtes identiques et d'éviter les appels répétés à MotherDuck lors des interactions utilisateur successives, tout en garantissant un rafraîchissement régulier pour intégrer les nouvelles offres insérées dans l'entrepôt.

5 Conclusions

5.1 Conclusion technique

Le projet RUCHE démontre la viabilité et l'efficacité d'une architecture modulaire intégrant des technologies hétérogènes pour la constitution d'un système d'analyse du marché de l'emploi orienté data science et intelligence artificielle. Les choix technologiques opérés se sont révélés judicieux tant du point de vue des performances que de la maintenabilité du système.

L'adoption de MongoDB comme base NoSQL intermédiaire a permis de gérer efficacement l'hétérogénéité structurelle des quatre sources de données, offrant la flexibilité nécessaire aux phases exploratoires du projet tout en préservant l'intégrité des structures natives de chaque plateforme. La transition vers MotherDuck pour l'entrepôt analytique final a validé l'hypothèse de performance pour le traitement de données vectorielles haute dimensionnalité, avec des temps de réponse inférieurs à une seconde pour les recherches par similarité cosinus sur un corpus de plusieurs milliers d'offres vectorisées en 768 dimensions.

La stratégie de parallélisation adoptée dans les scrapers a généré des gains de performance substantiels, avec des facteurs multiplicatifs compris entre 2,9 et 3,5 selon les sources, tout en maintenant des taux de succès supérieurs à 95%. L'approche innovante exploitant un LLM pour l'extraction structurée du scraper ChoisirLeServicePublic, bien que plus coûteuse en ressources API, a démontré une résilience supérieure face aux évolutions de structure HTML et a permis de traiter des sources multiples avec un code unique, réduisant la dette technique du projet.

Le pipeline ETL développé s'est montré robuste face aux données manquantes et aux formats hétérogènes, avec un taux de rétention de 67,3% après application du filtre machine learning, éliminant 32,7% de faux positifs tout en récupérant 1766 offres authentiquement liées à la data qui n'auraient pas été détectées par une approche purement lexicale. Les performances du modèle de classification TF-IDF couplé à une régression logistique (F1-Score : 0.978, ROC-AUC : 0.996) valident la pertinence de cette approche hybride combinant règles expertes et apprentissage supervisé.

L'architecture compute pushdown implémentée dans l'application Streamlit, déléguant les calculs vectoriels au moteur DuckDB, a permis de maintenir une expérience utilisateur fluide malgré la complexité des requêtes SQL multi-tables intégrant jointures dimensionnelles et calculs de similarité sémantique. Le système de cache à plusieurs niveaux (connexions, requêtes, modèle d'embeddings) a optimisé l'utilisation des ressources tout en garantissant la fraîcheur des données avec des TTL paramétrables.

Les limites identifiées concernent principalement la dépendance aux APIs externes (Mistral pour l'extraction, France Travail pour l'authentification OAuth2), la sensibilité

aux évolutions de structure HTML des sites scrapés nécessitant une maintenance régulière des sélecteurs CSS, et l'absence de mécanisme d'actualisation temps réel limitant l'utilisation à des analyses par lots différés. Des perspectives d'amélioration incluent l'implémentation d'index vectoriels HNSW pour accélérer les recherches de similarité à grande échelle, l'intégration de mécanismes de validation croisée pour les extractions LLM, et l'évolution vers une architecture événementielle permettant des mises à jour incrémentales continues.

5.2 Conclusion fonctionnelle

Sur le plan fonctionnel, le système RUCHE constitue un outil d'analyse multidimensionnelle du marché de l'emploi data et IA offrant des capacités inédites de découverte et d'exploration pour des utilisateurs aux profils variés.

La couverture géographique du corpus s'étend à l'ensemble du territoire métropolitain français, avec une représentation de plusieurs centaines de villes et des treize régions administratives. Les statistiques agrégées révèlent une forte concentration des opportunités en Île-de-France, représentant environ 40% du total des offres, suivie par les régions Auvergne-Rhône-Alpes et Provence-Alpes-Côte d'Azur. Au niveau urbain, Paris domine largement avec près de 30% des offres nationales, suivie par Lyon, Toulouse et Nantes. Cette cartographie spatiale permet aux utilisateurs d'identifier rapidement les bassins d'emploi attractifs et d'orienter leurs stratégies de mobilité professionnelle.

L'analyse salariale dévoile une valorisation différenciée selon les métiers, avec des rémunérations moyennes comprises entre 35k€ et 70k€ selon l'expérience et la spécialisation. Les postes de Data Scientist Senior et Machine Learning Engineer se positionnent dans les tranches supérieures, tandis que les profils juniors et les stages affichent des valorisations plus modestes mais cohérentes avec le marché. Les disparités régionales sont significatives, l'Île-de-France affichant des salaires supérieurs de 20 à 30% par rapport aux autres régions, prime de localisation justifiée par le coût de la vie mais également révélatrice de la concentration des acteurs technologiques de pointe.

L'analyse des compétences techniques met en évidence la prédominance du triptyque Python-SQL-Machine Learning, présent dans plus de 60% des offres analysées. Les technologies cloud (AWS, Azure, GCP) apparaissent dans environ 30% des annonces, tandis que les outils de visualisation (Power BI, Tableau) concernent principalement les profils orientés business intelligence. Le graphe de co-occurrences révèle des synergies technologiques fortes entre Docker et Kubernetes, entre Spark et Hadoop, et entre TensorFlow et PyTorch, informations précieuses pour planifier des parcours de formation cohérents.

Le moteur de recherche sémantique constitue l'innovation fonctionnelle majeure du système, permettant aux utilisateurs de formuler des requêtes en langage naturel du type "stage machine learning Lyon" ou "data engineer télétravail" sans se limiter à des correspondances lexicales strictes. Le système comprend les synonymes ("ingénieur données"

versus "data engineer"), les variantes orthographiques, et le contexte sémantique global de la requête, retournant des résultats pertinents même en absence de correspondance exacte des termes. Cette fonctionnalité démocratise l'accès à l'information pour des utilisateurs non familiers des terminologies techniques précises, tout en offrant une finesse de discrimination inatteignable par des moteurs de recherche traditionnels.

La composante cartographique interactive enrichit considérablement l'expérience utilisateur en matérialisant visuellement la distribution spatiale des opportunités. Le clustering dynamique des marqueurs facilite la navigation multi-échelle, du panorama national au détail urbain, tandis que les popups détaillés fournissent un accès direct aux annonces sources sans rupture de flux. Les filtres combinatoires permettent d'affiner progressivement les requêtes selon des critères multiples (contrat, salaire, compétences, date), produisant des sous-ensembles pertinents pour des analyses ciblées.

Les fonctionnalités analytiques avancées (clustering UMAP-HDBSCAN, graphe de cooccurrences, diagrammes sunburst) s'adressent à des utilisateurs plus techniques désireux d'explorer les structures latentes du marché, d'identifier des niches émergentes, ou d'anticiper les évolutions de demandes de compétences. Le module d'ajout manuel d'offres, couplé au chatbot LLM de structuration, ouvre des perspectives de contribution collaborative et d'enrichissement continu du corpus.

Au-delà de l'outil technique, le projet RUCHE constitue une contribution méthodologique démontrant la faisabilité d'architectures hybrides intégrant web scraping, NLP, machine learning et visualisation interactive pour produire des systèmes d'aide à la décision dans le domaine des ressources humaines et de l'orientation professionnelle. Les applications potentielles s'étendent à la veille concurrentielle pour les recruteurs, à l'orientation stratégique des cursus de formation pour les établissements d'enseignement supérieur, et à l'analyse macroéconomique des dynamiques du marché du travail pour les acteurs institutionnels.

Annexes

Annexe A : Notice technique du filtrage Machine Learning

Le document `notice_TFIDF_ML_filtre_data_nondata.md` détaille la méthodologie complète du système de classification des offres d'emploi par approche hybride regex-ML. Il couvre la phase de labellisation semi-automatique via patterns whitelist et blacklist, l'architecture du modèle TF-IDF couplé à une régression logistique, les métriques de performance obtenues (F1-Score, ROC-AUC, précision par classe), l'analyse approfondie d'un cas d'usage fil rouge ("Business Developer Grands Comptes – SaaS & IA"), et les justifications mathématiques des choix méthodologiques effectués.

Annexe B : Notice technique du moteur de recherche sémantique

Le document `notice_moteur_recherche_semantique.md` expose l'architecture complète du système de recherche vectorielle implémenté dans l'application RUCHE. Il détaille la stratégie d'enrichissement contextuel par jointures SQL sur le schéma en étoile, le choix du modèle multilingue MPNet et ses caractéristiques techniques (768 dimensions, entraînement cross-lingue), la formulation mathématique de la similarité cosinus et ses propriétés géométriques, l'implémentation du compute pushdown via la fonction array_cosine_similarity de DuckDB, ainsi qu'une analyse comparative des performances et des exemples de requêtes annotées illustrant le fonctionnement du système sur des cas concrets.

Références

[1] Reimers N, Gurevych I. Sentence-BERT : Sentence Embeddings using Siamese BERT-Networks. In : Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong : Association for Computational Linguistics ; 2019. p. 3982-92.

[2] Reimers N, Gurevych I. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In : Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Online : Association for Computational Linguistics ; 2020. p. 4512-25.

[3] Devlin J, Chang MW, Lee K, Toutanova K. BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. In : Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis : Association for Computational Linguistics ; 2019. p. 4171-86.

[4] Salton G, McGill MJ. Introduction to Modern Information Retrieval. New York : McGraw-Hill ; 1983.

[5] Sparck Jones K. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation. 1972 ;28(1) :11-21.

[6] Cox DR. The regression analysis of binary sequences. Journal of the Royal Statistical Society : Series B (Methodological). 1958 ;20(2) :215-32.

[7] Hosmer DW, Lemeshow S, Sturdivant RX. Applied Logistic Regression. 3rd ed. Hoboken : Wiley ; 2013.

[8] Joachims T. Text categorization with support vector machines : Learning with many relevant features. In : Nédellec C, Rouveirol C, editors. Machine Learning : ECML-98. Berlin : Springer ; 1998. p. 137-42.

[9] Sebastiani F. Machine learning in automated text categorization. ACM Computing Surveys. 2002 ;34(1) :1-47.

[10] He H, Garcia EA. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering. 2009 ;21(9) :1263-84.

[11] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE : Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research. 2002 ;16 :321-57.

[12] Raasveldt M, Mühleisen H. DuckDB : an Embeddable Analytical Database. In : Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19). Amsterdam : ACM ; 2019. p. 1981-4.

[13] Kimball R, Ross M. The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling. 3rd ed. Indianapolis : Wiley ; 2013.

[14] Inmon WH. Building the Data Warehouse. 4th ed. Indianapolis : Wiley ; 2005.

[15] Johnson J, Douze M, Jégou H. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data. 2021 ;7(3) :535-47.

[16] Mitchell T. Machine Learning. New York : McGraw-Hill ; 1997.

[17] Manning CD, Raghavan P, Schütze H. Introduction to Information Retrieval. Cambridge : Cambridge University Press ; 2008.

[18] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn : Machine Learning in Python. Journal of Machine Learning Research. 2011 ;12 :2825-30.

[19] McKinney W. Data Structures for Statistical Computing in Python. In : van der Walt S, Millman J, editors. Proceedings of the 9th Python in Science Conference. Austin ; 2010. p. 56-61.

[20] Hunter JD. Matplotlib : A 2D graphics environment. Computing in Science & Engineering. 2007 ;9(3) :90-5.

[21] Sievert C. Interactive Web-Based Data Visualization with R, plotly, and shiny. Boca Raton : Chapman and Hall/CRC ; 2020.

[22] Agafonkin V. Leaflet : an open-source JavaScript library for mobile-friendly interactive maps [Internet]. 2022 [cited 2026 Jan 11]. Available from : https://leafletjs.com/

[23] Python Software Foundation. Python Language Reference, version 3.11 [Internet]. Wilmington : Python Software Foundation ; 2023 [cited 2026 Jan 11]. Available from : https://www.python.org/

[24] MongoDB Inc. MongoDB Documentation [Internet]. New York : MongoDB Inc. ; 2023 [cited 2026 Jan 11]. Available from : https://www.mongodb.com/docs/

[25] MotherDuck. MotherDuck Documentation [Internet]. Seattle : MotherDuck ; 2023 [cited 2026 Jan 11]. Available from : https://motherduck.com/docs/

[26] Streamlit Inc. Streamlit Documentation [Internet]. San Francisco : Streamlit Inc. ; 2023 [cited 2026 Jan 11]. Available from : https://docs.streamlit.io/

[27] Beautiful Soup Documentation [Internet]. [cited 2026 Jan 11]. Available from : https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[28] Selenium Project. Selenium Documentation [Internet]. [cited 2026 Jan 11]. Available from : https://www.selenium.dev/documentation/

[29] Mistral AI. Mistral API Documentation [Internet]. Paris : Mistral AI ; 2024 [cited 2026 Jan 11]. Available from : https://docs.mistral.ai/

[30] Hugging Face. Transformers : State-of-the-art Machine Learning for PyTorch, TensorFlow, and JAX [Internet]. New York : Hugging Face ; 2023 [cited 2026 Jan 11]. Available from : https://huggingface.co/docs/transformers/

Groupe n°6 - Master 2 SISE - Promotion 2025-2026

Responsable pédagogique : M. Ricco RAKOTOMALALA

Projet NLP et Text Mining