Résolution d'entités pour les flux de données à l'aide de la technique d'embedding

Zhongwei MA [1], **Philippe ROOSE** [1], **Jiefu SONG** [2]

1. *Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA Anglet, France*
   zhongwei.ma@univ-pau.fr,philippe.roose@univ-pau.fr
2. *Institut de Recherche en Informatique de Toulouse - Université Toulouse Capitole 31000, Toulouse, France*
   jiefu.song@ut-capitole.fr

RÉSUMÉ. *La plupart des systèmes de traitement de flux de données recueillent des données provenant de différentes sources en temps réel et les consomment immédiatement. Cependant, de nombreuses analyses décisionnelles nécessitent des données en temps réel et des données historiques (par exemple, un dataset local ou des enregistrements antérieurs) en même temps pour comprendre la situation actuelle et avoir une vue globale. La résolution d'entités permet de déterminer si deux enregistrements différents font référence à la même entité en l'absence d'un identifiant unifié dans le cas multi-sources. Il est donc essentiel d'intégrer les données en temps réel aux données stockées en appliquant la résolution d'entités et de garantir l'accessibilité et la facilité d'utilisation des données multi-sources. Les méthodes existantes pour la résolution d'entités sont souvent incapables de prendre en charge ces dernières de manière continue tout en fournissant des méthodes générales et efficaces pour l'analyse de données complexes telles que le texte. Étant donné la bonne capacité de l'embedding dans la capture des informations sémantiques et syntaxiques, nous visons à appliquer cette technique au traitement de flux de données dans le but d'intégrer les données entrants en temps réel aux données historiques. En outre, la plupart des approches privilégient aujourd'hui la précision et la scalabilité tout en ignorant l'augmentation de la consommation d'énergie que nécessite l'amélioration de cette précision. Nous proposons donc ici une approche d'embedding de graphe dynamique adaptée au traitement des données en temps réel pour effectuer la résolution d'entités dans des tables relationnelles tout en évaluant la consommation d'énergie de ce traitement.*

ABSTRACT.

MOTS-CLÉS : *flux de données, résolution d'entité, embedding*

KEYWORDS: *entity resolution, data stream, embedding*

1. Introduction

La résolution d'entités (Entity Resolution, ci-après dénommé ER), également appelée couplage d'enregistrements (record linkage) ou l'alignement de données (data matching), vise à trouver différentes descriptions se rapportant à la même entité dans le monde réel, au sein de sources de données ou entre celles-ci, lorsqu'il n'existe pas d'identifiant unique de l'entité, et ceci afin de garantir la qualité et la cohérence des datasets intégrés. (Christophides *et al*., 2020). En supposant que nous disposions d'enregistrements de ventes en temps réel sur différentes plateformes de distributeurs, si ces enregistrements sont intégrés aux enregistrements historiques de notre base de données, nous pouvons obtenir des informations plus approfondies sur les produits populaires et les tendances d'achat des consommateurs. Il s'agit d'une étape cruciale lorsque nous procédons à l'intégration de données et que nous voulons offrir une vue uniforme de sources de données autonomes et hétérogènes, ce qui facilite le traitement ultérieur (Maharana *et al*., 2022).

Dans un contexte de big data, les données, caractérisées par leur nature continue et temps réel, sont de plus en plus utilisées dans le monde numérique (Bahri *et al*., 2021). Ces données sont générées à partir d'un large éventail de sources, notamment les plateformes de réseaux sociaux, les objets de l'IoT, etc. Les exigences en matière d'analyse des flux de données sont de deux aspects : le traitement en temps réel pour soutenir la prise de décision immédiate avec une faible latence, et le stockage des données pour des requêtes ultérieures ou des analyses plus complexes et plus coûteuses en temps. Par conséquent, les systèmes modernes de traitement des flux de données doivent intégrer à la fois le traitement en flux et le traitement par lots (Isah *et al*., 2019) dans un cadre unifié, comme l'architecture Lambda introduite à l'origine par (Marz, Warren, 2015). Pour améliorer l'efficacité de ces systèmes, il est essentiel d'intégrer les données de flux aux données historiques par ER.

Le premier défi tient à l'incomplétude des datasets. En raison de la nature illimitée des flux, un cadre dynamique est nécessaire pour exécuter l'ER de façon incrémentale. L'absence d'un dataset complet complique la tâche, car il devient difficile d'identifier la meilleure correspondance dans un volume de données fini.

Historiquement, l'ER a été définie comme une tâche hors ligne réalisée lors de l'intégration des données afin d'améliorer la qualité des bases de données (Christophides *et al*., 2020). De nombreuses approches ont ainsi été développées pour traiter ce problème en mode batch à l'aide de différentes méthodes. Cependant, ces approches orientées batch, conçues pour des bases de données statiques, ne sont pas adaptées aux flux de données en continu. Jusqu'à présent, deux principales approches ont été adaptées aux données incrémentales. La première repose sur des méthodes basées sur des règles (Gazzarri, Herschel, 2023) Toutefois, ces méthodes s'appuient souvent sur des règles complexes et sont limitées à des domaines spécifiques, ce qui restreint leur généralisabilité. La seconde approche repose sur l'apprentissage automatique, en particulier les techniques de clustering (Do Nascimento *et al*., 2018 ; Saeedi *et al*., 2020).

Néanmoins, cette approche rencontre des difficultés à traiter efficacement les données textuelles non structurées, ce qui limite son efficacité dans des environnements réels.

Le deuxième défi porte sur la consommation. La grande quantité de données et leurs traitements entraînent une consommation significative de ressources.

Pour surmonter les limitations de l'analyse textuelle et la complexité des algorithmes spécifiques à un domaine, nous nous tournons vers les techniques d'embedding, qui projettent les données textuelles dans un espace vectoriel de faible dimension afin de capturer les relations et l'information sémantique des mots (Wang *et al*., 2020). Jusqu'à présent, cette technique est principalement utilisée pour le traitement en batch (Li *et al*., 2020). Récemment, d'importants travaux ont été menés sur les embeddings incrémentaux (Barros *et al*., 2021), jetant ainsi les bases de l'ER dans un contexte de flux de données sans s'appuyer sur le traitement en batch.

Dans ce travail, nous introduisons un cadre pour l'intégration de données en flux continu basé sur l'embedding dynamique de graphes. Ce cadre englobe l'ensemble du processus, depuis l'entraînement préalable des modèles d'embedding sur des datasets locaux, jusqu'à l'intégration des données en flux, en impliquant progressivement les nouvelles données et en déterminant si elles appartiennent à une entité existante dans l'dataset d'origine. La construction des embeddings permet de capturer efficacement les informations sémantiques et syntaxiques (Wang *et al*., 2020). De plus, afin de relever les défis énergétiques, nous proposons une approche originale permettant de mesurer la consommation énergétique lors de l'exécution du code d'embedding pour l'intégration des données. Nous proposons les contributions suivantes :

1) Modèle d'embedding dynamique de graphes. Basé sur des modèles d'embedding capturant les relations et les informations sémantiques des données, nous mettons à jour le graphe et ses embeddings de manière incrémentale. Plus précisément, à l'arrivée d'un nouvel enregistrement, nous ajoutons un nœud correspondant au graphe, établissons des arêtes entre ce nœud et la structure existante, et générons des parcours aléatoires évolutifs à partir du nouveau nœud afin de quantifier la similarité. Cette approche évite l'entraînement répétitif, réduisant ainsi les coûts de calcul tout en permettant un ER efficace dans un contexte de flux de données.

2) Évaluation du modèle en termes de performance et d'efficacité. Nous évaluons la méthode proposée en analysant sa consommation énergétique ainsi que ses performances. Cette analyse permet une compréhension quantitative des ressources consommées dans les tâches d'ER basées sur l'embedding.

L'article est structuré comme suit. La section 2 présente les méthodes courantes d'ER et en analyse les caractéristiques. La section 3 introduit la conception de notre approche. Dans la section 4, nous décrivons en détail l'implémentation du protocole et analysons les résultats obtenus. Enfin, nous concluons dans la dernière section en évoquant les perspectives de travaux futurs.

2. État de l'art

Nous passons en revue la littérature liée à cet article selon deux axes. Dans **la Section 2.1**, nous résumons les méthodes adaptées aux flux de données afin de présenter l'état actuel de l'ER incrémental. **La Section 2.2** se concentre sur les techniques d'embedding et illustre comment elles peuvent être appliquées pour réaliser l'ER.

2.1. Résolution d'entités pour les flux

De nombreuses études mentionnées dans la littérature (Binette, Steorts, 2022) ont exploré diverses approches pour l'ER afin de relever le défi des données incrémentales. Nous les catégorisons en trois grands groupes : les approches basées sur les requêtes, les approches basées sur les règles et les approches basées sur l'apprentissage.

Les approches basées sur les requêtes (Simonini *et al*., 2022) stockent toutes les données entrantes et effectuent l'ER au moment de l'exécution de la requête. Cependant, ces méthodes à la demande peuvent nécessiter plusieurs minutes de traitement pour fournir un résultat. Par ailleurs, le stockage de doublons dans les enregistrements augmente également la consommation des ressources, entraînant une inefficacité.

Les approches basées sur les règles (Gazzarri, Herschel, 2021) sont parmi les plus couramment utilisées pour l'ER. Elles reposent sur des règles déterministes et interprétables permettant de comparer les attributs des enregistrements. Toutefois, la mise en œuvre de ces approches peut être très complexe et nécessite des algorithmes sophistiqués adaptés aux flux de données.

Les approches basées sur l'apprentissage sont devenues de plus en plus courantes grâce aux avancées des techniques d'apprentissage automatique *(machine learning)*. La plupart de ces méthodes fonctionnent en mode batch (Papadakis *et al*., 2023) et visent à améliorer la précision et l'efficacité grâce à des techniques telles que la correspondance de graphes et les réseaux neuronaux. Cependant, l'entraînement des modèles d'apprentissage automatique présente des défis importants, en particulier en présence de données avec bruits ou de ressources annotées limitées, et ce d'autant plus dans un contexte de flux de données où le dataset est incomplet. Les méthodes incrémentales adoptent souvent des techniques non supervisées comme le clustering (Saeedi *et al*., 2020) Plutôt que de lier individuellement les enregistrements, l'objectif du clustering est de regrouper les enregistrements correspondant aux mêmes entités, souvent latentes. Néanmoins, les performances des méthodes de clustering restent limitées lorsqu'il s'agit de traiter des données de grande dimension, telles que le texte, qui joue un rôle crucial dans l'ER. De plus, les données avec bruits, comme les fautes d'orthographe ou les valeurs manquantes, peuvent générer du bruit et des faux positifs qui diminuent la précision, en particulier dans le cas des flux de données (Christophides *et al*., 2020).

2.2. Embeddings et résolution d'entités

L'embedding de mots est une technique clé en apprentissage automatique, visant à projeter des données de haute dimension dans un espace vectoriel où les mots sont représentés par des vecteurs de longueur fixe. Cette représentation permet de capturer les relations sémantiques entre mots similaires (Almeida, Xexéo, 2023).

Les approches basées sur la prédiction, par exemple, word2vec (Mikolov *et al*., 2013), entraînent des réseaux neuronaux récurrents en exploitant les données locales (par exemple, le contexte d'un mot) afin soit de prédire un mot à partir de son contexte, soit d'inférer le contexte à partir d'un mot donné. Ce processus garantit que les mots ayant des significations sémantiques similaires sont associés à des représentations vectorielles similaires. D'un autre côté, les méthodes basées sur le comptage (par exemple, Glove (Pennington *et al*., 2014)) construisent une matrice de cooccurrence de mots à partir d'un corpus, capturant des informations statistiques globales telles que le nombre total d'occurrences des mots et leurs fréquences. Voici quelques applications clés des techniques d'embedding dans ce domaine.

DeepER (Ebraheem *et al*., 2018) est l'un des premiers à utiliser des embeddings de mots (comme GloVe). Il applique des réseaux LSTM (Long Short-Term Memory) pour apprendre les relations entre les attributs des tuples en étiquetant les données et en les convertissant en une représentation vectorielle de dimensions fixes. Ensuite, des caractéristiques de similarité sont calculées et introduites dans un classificateur binaire afin de déterminer les correspondances entre entités. DeepMatch (Mudgal *et al*., 2018) adopte une approche similaire mais offre davantage de choix pour l'embedding des attributs et la représentation des similarités. Ils démontrent des performances compétitives sur des datasets contenant une certaine proportion de données avec bruits.

Ditto (Li *et al*., 2020) utilise des modèles pré-entraînés basés sur des "transformers" pour extraire des caractéristiques et ajuste le modèle en tant que classificateur binaire pour l'appariement d'entités. Il gère mieux les données avec bruits grâce à un processus de sérialisation structuré et à des techniques d'augmentation de données. (Brunner, Stockinger, 2020) teste également des modèles basés sur les transformers et explore les mécanismes d'attention pour la tâche d'ER. Cependant, l'utilisation de grands modèles de langage engendre des coûts énergétiques énormes en raison d'une exploitation de ressources significative.

Plus récemment, plusieurs frameworks ont proposé de représenter les données ou les paires d'enregistrements sur la base des embeddings, tels que multiEM (Zeng *et al*., 2024), Unicorn (Tu *et al*., 2023), et FlexER (Genossar *et al*., 2023). Ces frameworks emploient des techniques avancées, notamment les réseaux de neurones graphiques, afin d'identifier les similarités entre enregistrements et d'améliorer les capacités de généralisation. Néanmoins, ils encodent les données sous forme de séquences linéaires fixes basées sur des règles prédéfinies, ce qui limite leur capacité à interpréter les tables relationnelles, telles que les relations entre colonnes. Afin d'obtenir une interprétation plus complète, Embdi (Cappuzzo *et al*., 2020) introduit des représentations graphiques dans les tâches d'ER pour enrichir l'information capturée par les embeddings.

Malgré ces avancées, toutes les méthodes mentionnées fonctionnent en mode batch, ce qui limite leur application aux tâches nécessitant un traitement en temps réel. Pour traiter des flux de données en continu, il est nécessaire de disposer d'une méthode permettant d'intégrer progressivement les nouvelles données sans devoir relancer manuellement le programme à chaque fois.

3. Proposition

3.1. Problématique

Pour simuler l'intégration des données en flux dans un dataset local, nous supposons l'existence de deux sources de données : l'une contient l'ensemble des enregistrements issus d'une source statique, où chaque enregistrement correspond à une entité distincte, et l'autre est constituée des données entrantes en streaming. L'objectif de l'ER incrémental est d'associer un enregistrement entrant à l'entité la plus similaire dans un ensemble de références en évolution. Cet ensemble de références comprend à la fois le jeu de données statiques et les enregistrements de flux précédemment observés, et il se met à jour dynamiquement au fur et à mesure que de nouvelles données arrivent. Pour préciser davantage notre cadre, nous introduisons les principaux symboles dans la table 1 et les algorithmes utilisés dans les approches incrémentales.

TABLEAU 1. *Définitions des symboles*

Symbole	Explication
D	Un dataset statique, qui peut être vide
$\mathcal{D}$	Un dataset dynamique
ΔD	L'incrément d'un dataset incrémental pendant une période
R_i	Un enregistrement arbitraire
$\mathcal{C}_t$	L'ensemble des données traitées avant le temps t, servant d'entrée pour le prochain cycle de traitement
M_{t,R_i}	La meilleure correspondance pour l'enregistrement R_i au temps t
$\mathcal{M}_t$	La liste des correspondances pour tous les enregistrements au temps t
$sim(\cdot,\cdot)$	Une fonction de similarité pour comparer les enregistrements

L'ER dans les flux de données présente deux principaux défis. Le premier est lié à l'incomplétude de datasets dynamiques. À un instant t, nous n'avons qu'une vue partielle de $\mathcal{D}$, ce qui empêche d'attendre que l'ensemble des données soit disponible avant d'exécuter le processus. Plus précisément, nous pouvons représenter le dataset $\mathcal{D}$ à un instant donné comme suit :

$$\mathcal{D}_n = \bigsqcup_{i=1}^{n} \Delta D_i$$

où $\mathcal{D}_n$ représente le dataset formé en combinant les incréments $\Delta D_1, \ldots, \Delta D_n$, chaque incrément de données ΔD_i correspondant à de nouvelles données arrivant dans un intervalle de temps spécifique $[t_{i-1}, t_i]$. Nous supposons que ces intervalles sont séquentiels et non chevauchants, c'est-à-dire que chaque nouvel intervalle temporel suit *(meet)* immédiatement le précédent : $[t_{i-1}, t_i]$ meet $[t_i, t_{i+1}]$.

Ainsi, le dataset $\mathcal{D}_n$ évolue au fil du temps par l'ajout séquentiel de ces incréments.

Plutôt que de comparer l'ensemble des données en mode batch, ce qui est impossible étant donné que le flux de données est généré de manière continue sans fin, le processus doit fonctionner de manière incrémentale. Une approche raisonnable consiste à exécuter le processus d'ER à chaque arrivée d'un nouvel incrément ΔD_n, afin d'éviter une accumulation excessive de données. Le moment de ces mises à jour peut être déterminé par divers facteurs, tels qu'un nombre prédéfini de nouveaux enregistrements ou des intervalles de temps fixes.

L'idée principale de l'ER incrémental est de réutiliser les résultats des traitements précédents pour éviter les calculs redondants. Dans ce cadre, pour des données incrémentales ΔD_n à l'instant t, les données précédemment traitées sont stockées dans un ensemble de candidats, représenté par $\mathcal{C}_t = D \cup \mathcal{D}_{n-1}$. Ces données ont été transformées de leur état initial vers une forme prête pour la comparaison. À chaque mise à jour incrémentale, l'ensemble de candidats est continuellement enrichi avec les nouvelles données, tandis que les données précédemment traitées restent inchangées et sont uniquement utilisées pour la fonction de similarité. La Figure 1 illustre l'évolution des données impliquées dans ce processus incrémental au fil du temps. Ainsi, nous pouvons formuler le premier problème comme suit :

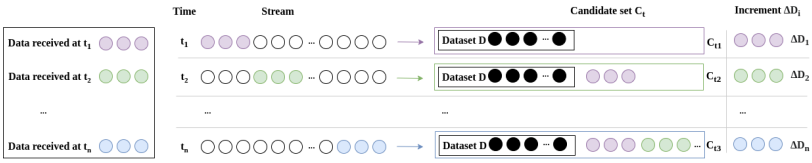

FIGURE 1. *Données impliquées dans le processus incrémental au fil du temps*

Définition du problème 1. À tout instant t, mettre à jour de manière incrémentale l'ensemble de candidats $\mathcal{C}_t$ et comparer les paires de données à l'aide de la fonction de similarité $sim(\Delta D_n, \mathcal{C}_t)$ afin de déterminer M_{t,R_i} pour chaque élément $R_i \in \Delta D_n$.

Le second problème est que la mise en correspondance n'est pas un processus statique dans le temps. Étant donné que les enregistrements candidats évoluent continuellement, les correspondances précédemment établies M_{t',R_i} (où $t' < t$) peuvent devoir être mises à jour lors de l'arrivée de nouveaux enregistrements. Cela nécessite un maintien continu de la cohérence des résultats de correspondance. Nous définissons l'ensemble des correspondances à l'instant t comme suit :

$$\mathcal{M}_t = \{(R_i, M_{t,R_i}) \mid i \leq t\}$$

Avec la croissance de $\mathcal{M}_t$, le stockage de toutes les correspondances historiques devient rapidement impraticable. Pour garantir l'efficacité, il est nécessaire de définir une stratégie permettant de déterminer quelles correspondances doivent être conservées ou supprimées. Nous résumons ce problème ainsi :

Définition du problème 2. Gérer et maintenir les résultats de correspondance dynamiques au fil du temps $\mathcal{M}_t$, de sorte qu'à tout instant t, la correspondance correcte apparaisse dans la liste dès qu'elle est identifiée, et que sa valeur de similarité reste relativement élevée par rapport à l'ensemble des enregistrements potentiellement similaires.

Nos pipelines répondent aux problèmes 1 et 2 via un processus d'ER incrémental qui met à jour efficacement les correspondances tout en réduisant l'espace de stockage. Toutes les informations de correspondance pertinentes sont stockées dans un index unique, assurant un accès uniforme aux requêtes. Les détails sont présentés dans la section suivante.

3.2. Proposition

Notre cadre d'ER incrémental [1] comprend la préparation des données, la construction d'embeddings incrémentaux et la génération des correspondances. Dans ce travail, nous réutilisons l'approche de construction d'un modèle d'embedding présentée dans (Cappuzzo *et al*., 2020), qui est adaptée à l'extension aux flux de données et représente des informations sémantiques riches dans un modèle linguistique léger. Cependant, au lieu de construire un modèle d'embedding basé sur deux datasets complets, nous introduisons un embedding incrémental qui met à jour le modèle au fil du temps afin d'améliorer la scalabilité des applications de traitement en flux. Pour illustrer comment ces trois parties s'intègrent et comment les données circulent entre elles, nous résumons ces étapes dans la section suivante.

3.2.1. Préparation des données

La préparation des données traite les incréments bruts ΔD_i provenant des sources de données qui génèrent des données en temps réel. L'extraction de métadonnées est effectuée à cette étape, permettant d'obtenir des descriptions concises et représentatives des entités du monde réel. Une table relationnelle est ensuite construite à partir des métadonnées extraites, fournissant une base flexible pour diverses tâches dans différents domaines. Dans ce travail, elle est spécifiquement utilisée pour l'ER. Cette étape de préparation des données transforme des données non structurées en un format structuré, réduisant efficacement le volume de données à comparer et standardisant les informations pour les traitements en aval.

3.2.2. Construction incrémentale des embeddings

À partir de la table relationnelle contenant les incréments de données, nous pouvons construire un modèle d'embedding incrémental. Le modèle utilisé dans ce processus est pré-entraîné sur un dataset local D, mais l'étape de pré-entraînement peut être omise si nous intégrons des flux de données à partir de zéro. Le pipeline d'entraînement est structuré comme suit : chaque enregistrement dans la table relationnelle se

1. https://github.com/Mzhongwei/er_embedding_streaming

Algorithme 1 : Algorithme d'Embedding Incrémental

Input : incrément de données ΔD_i, modèle d'embedding $\mathcal{E}_{i-1}$, graphe $\mathcal{G}_{i-1}$
Output : modèle d'embedding mis à jour $\mathcal{E}_i$, graphe mis à jour $\mathcal{G}_i$

1: Initialiser la liste des nœuds évolutifs $\mathcal{L}$
2: Initialiser le graphe $\mathcal{G}_i \leftarrow \mathcal{G}_{i-1}$
3: **foreach** $R \in \Delta D_i$ **do**
4: $\quad \mathcal{G}_i, \Delta N_1 \leftarrow$ ajouterNœudAGraphe$(\mathcal{G}_i, R)$
5: $\quad \mathcal{G}_i, \Delta N_2 \leftarrow$ ajouterLienAGraphe$(\mathcal{G}_i, R)$
6: $\quad \mathcal{L} \leftarrow$ ajouterNœudDynamiqueÀListe$(\mathcal{G}_i, \Delta N_1, \Delta N_2)$
7: Générer de nouvelles random walks pour $\mathcal{L}$: $\mathcal{W} =$ GenererRandomWalk$(\mathcal{G}, \mathcal{L})$;
8: Entraîner le modèle avec les random walks $\mathcal{W}$:
$\mathcal{E}_i =$ MettreAJourEmbedding$(\mathcal{E}_{i-1}, \mathcal{W})$
9: **return** $\mathcal{E}_i$, $\mathcal{G}_i$

voit attribuer un identifiant unique. Chaque valeur, y compris les numéros d'ID et les noms de colonnes, est traitée comme un nœud dans un graphe hétérogène non orienté. Un random walk est ensuite effectué sur ce graphe afin de générer des représentations contextuelles de ces valeurs et mots sous forme de phrases. Ces phrases sont ensuite projetées dans un espace vectoriel de dimension fixe, formant ainsi la représentation en embedding des valeurs (Cappuzzo *et al*., 2020).

Le pipeline incrémental suit la même structure mais intègre dynamiquement les nouvelles données. L'ensemble du processus est présenté dans l'algorithme 1. Lorsqu'un incrément de données ΔD_i arrive au temps t_i, nous utilisons le graphe $\mathcal{G}_{i-1}$ et le modèle d'embedding $\mathcal{E}_{i-1}$ à l'instant précédent t_{i-1} comme variables initiales. Le graphe est mis à jour en ajoutant de nouveaux nœuds contenant les valeurs extraites de chaque enregistrement R de l'incrément ΔD_i, si ces valeurs ne sont pas déjà présentes. Ensuite, de nouvelles arêtes sont insérées dans le graphe pour représenter les relations entre différents enregistrements et différentes valeurs au sein d'un même enregistrement (cf. lignes 3-5). À cette étape, tous les nœuds ayant évolué, y compris les nouveaux nœuds ΔN_1 ainsi que ceux dont les arêtes ont changé ΔN_2, sont enregistrés dans une liste de nœuds évolutifs $\mathcal{L}$ (cf. ligne 6). Un nouveau random walk $\mathcal{W}$ est ensuite effectué à partir de ces nœuds pour générer de nouvelles phrases (cf. ligne 7). Ces phrases sont utilisées pour affiner le modèle d'embedding par apprentissage incrémental, garantissant que les représentations restent à jour sans nécessiter un réentraînement complet depuis le début (cf. ligne 8). Le graphe mis à jour $\mathcal{G}_i$ et le modèle d'embedding $\mathcal{L}_i$ sont stockés pour l'itération suivante.

3.2.3. Construction de la liste de correspondances

La construction de la liste de correspondances prend en entrée les identifiants des enregistrements incrémentaux pour représenter ces enregistrements et le modèle d'embedding renouvelle à l'étape précédente. Comme illustré dans l'algorithme 2, pour chaque enregistrement représenté par un identifiant, nous pouvons calculer les degrés de similarité entre les vecteurs des numéros d'ID afin d'identifier les enregistrements

les plus similaires selon les représentations vectorielles du modèle d'embedding (cf. ligne 2). Ensuite, nous sélectionnons les k correspondances les plus similaires (cf. ligne 3). Ces relations, ainsi que leurs scores de similarité, sont stockées symétriquement dans une liste, dont seules les n meilleures correspondances sont conservées par enregistrement (cf. lignes 4-6).

Algorithme 2 : Construction de la Liste de Correspondances

Input : Identifiants des enregistrements incrémentaux $\mathcal{ID} = \{id_1, id_2, \ldots, id_n\}$, modèle d'embedding $\mathcal{E}_i$, ensemble de candidats $\mathcal{C}$, nombre de correspondances les plus similaires k après calcul et n dans la liste de correspondances, liste de correspondances précédente $\mathcal{M}_{i-1}$

Output : Liste de correspondances mise à jour $\mathcal{M}_i$

1: **foreach** $id \in \mathcal{ID}$ **do**
2: Exécuter la fonction de similarité : Sim($\mathcal{E}_i$, id)
3: Sélectionner les k identifiants les plus similaires avec leur score de similarité s : $\mathcal{N}_{id} = \{(j_1, s_1), (j_2, s_2), \ldots, (j_k, s_k) \mid j \in \mathcal{C}\}$
4: $\mathcal{M}_i[id] \leftarrow$ top-n éléments les plus similaires de $\mathcal{M}_{i-1}[id] \cup \mathcal{N}_{id}$;
5: **foreach** $(j, s) \in \mathcal{N}_{id}$ **do**
6: $\mathcal{M}_i[j] \leftarrow$ top-n éléments les plus similaires de $\mathcal{M}_{i-1}[j] \cup \{(id, s)\}$;
7: **return** $\mathcal{M}_i$

4. Expérimentation

4.1. Protocole

Dans cette section, nous présentons une évaluation expérimentale de notre proposition. L'objectif principal de nos expériences est d'évaluer les performances de notre architecture de traitement en temps réel et sa capacité à traiter des données incrémentales. Notre évaluation vise à répondre aux questions de recherche suivantes :

QR1. Quelle est l'efficacité et la performance de notre proposition ?

QR2. Comment la taille des données d'entraînement incrémentales affecte-t-elle les résultats expérimentaux de l'apprentissage incrémental ?

QR3. Quelle est la quantité d'énergie consommée par ce processus ?

Datasets. Nous utilisons un dataset open-source créé à partir de données réelles et dédié à la tâche d'ER. Il est composé de données provenant de deux sources et décrit une liste de produits d'Amazon [2].

Métriques d'évaluation. Nous utilisons la précision, le rappel et le score F1 en tant que principales métriques de performance. Étant donné qu'il est difficile de définir la fin du processus lors du traitement des flux, nous adoptons une approche globale

2. https://zenodo.org/records/7930461

de l'exactitude après le traitement de toutes les données de la source B. Nous trions la liste des correspondances par ordre décroissant de similarité et sélectionnons $x(x \leq listLength)$ enregistrements dont la similarité se classe parmi les k premiers pour observer le rappel. En particulier, si une similarité correspond à plusieurs résultats, nous les prenons tous. Ensuite, nous transformons ces listes en paires de correspondances similaires sous forme de groupes de deux, de sorte que nous obtenons $N_{predicted}$ paires après suppression des doublons. Nous comparons ensuite le résultat avec les N_{truth} paires de la vérité terrain, où les enregistrements correspondent à la même entité, afin d'identifier le nombre de paires correctement prédites N_{common}.

Pour mesurer la consommation énergétique du programme, nous évaluons séparément la puissance du CPU et de la RAM en watts, ainsi que la consommation énergétique en joules.

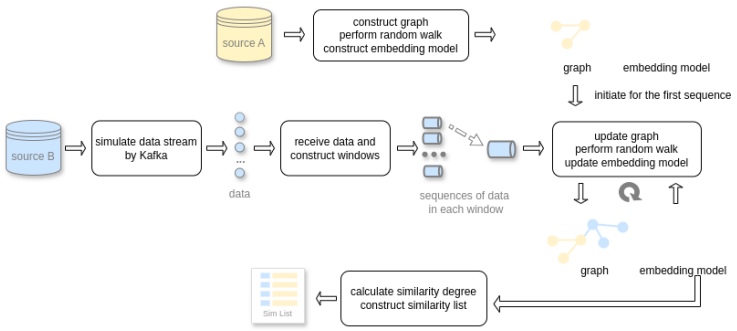

FIGURE 2. *Pipeline du processus d'implémentation*

Implémentation. Afin de simuler une application de traitement de données en flux, nous considérons les tuples de la source A comme un dataset local utilisé pour le pré-entraînement et envoyons les données de la source B vers un producteur Kafka de manière incrémentale, sous forme de flux de données à une fréquence fixe et régulière, sans concurrence. Pour chaque enregistrement de la source B, nous recherchons ses valeurs similaires dans la source A et construisons une séquence allant jusqu'à 10 éléments dans les deux directions (comme mentionné en Section 3.2) afin d'analyser dans quelle mesure le classement basé sur la similarité s'aligne avec les appariements corrects fournis par la vérité terrain. Dans cet article, nous utilisons directement les données relationnelles, et cet aspect ne sera donc pas approfondi davantage. L'ensemble du processus est illustré dans la Figure 2.

Sur le plan énergétique, nous utilisons Ecofloc (Alvarez-Valera *et al.*, 2024) pour mesurer la consommation énergétique. Ecofloc [3] est un WattMètre logiciel qui permet

3. https://github.com/labDomolandes/ecofloc

de mesurer l'énergie consommée par les processus en fonction de la charge qu'ils génèrent sur les principaux composants (CPU, GPU, RAM, NET, HD).

Les expériences ont été réalisées sur un ordinateur portable équipé d'un processeur 12×12th Gen Intel(R) Core(TM) i5-1245U avec 15,3 Go de RAM.

4.2. Analyse des résultats

4.2.1. Expériences sur l'efficacité

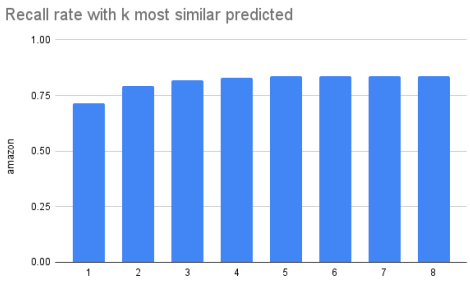

FIGURE 3. *Effet du nombre k de paires prédites les plus similaires sur le taux de rappel*

La Figure 3 illustre l'effet de l'ordre des valeurs les plus similaires sur le rappel, afin de démontrer l'efficacité de notre algorithme. Plus le rappel est élevé, plus le nombre de paires d'enregistrements correctement prédites est important. À mesure que la valeur de k augmente, nous observons que l'algorithme incrémental parvient à prédire avec succès la plupart des paires d'enregistrements similaires. De plus, la majorité des prédictions correctes sont concentrées parmi les enregistrements ayant des rangs de similarité relativement élevés.

L'ajustement de la valeur de k permet d'améliorer efficacement le rappel dans les tâches d'ER. En outre, au-delà d'un certain seuil, la longueur de la liste de similarité a un impact minimal sur la sensibilité des paires prédites. La plupart des paires ayant une faible similarité ne sont pas pertinentes, et privilégier les paires les plus similaires permet d'améliorer de manière significative à la fois l'efficacité et le rappel.

4.2.2. Effet de la taille des incréments sur les résultats

Dans l'apprentissage incrémental, la taille des nouvelles données ajoutées à chaque étape influence le processus d'entraînement du modèle. Des mises à jour trop fréquentes ou l'introduction d'un volume de données trop important en une seule fois peuvent déstabiliser le modèle. Afin d'analyser l'effet de la taille des données incrémentales sur la qualité des embeddings et sur la performance d'ER, nous avons fait

varier la proportion des données incrémentales. La taille des incréments est exprimée en pourcentage des données pré-entraînées. Par exemple, un incrément de 5% signifie que la quantité de données traitée à chaque étape incrémentale correspond à 5% du volume initial des données pré-entraînées.

TABLEAU 2. *Effect of increment size on results*

Taille de l'incrément	Temps d'exécution (s)	k	Précision (%)	Rappel (%)	Score F1 (%)
5.0%	326.50	1	34.7	59.3	43.5
		2	24.33	66.13	35.53
		3	19.93	67.33	31.00
10%	326.50	1	32.50	70.23	44.70
		2	22.33	78.67	34.67
		3	18.50	81.00	29.93
20%	332.00	1	34.40	71.33	46.40
		2	23.53	79.33	36.67
		3	20.00	81.80	32.00
30%	336.00	1	36.73	72.33	48.83
		2	25.87	79.67	38.90
		3	21.67	81.33	34.00
40%	338.00	1	34.50	70.00	46.33
		2	24.00	78.47	36.60
		3	20.00	81.23	32.40

Le Tableau 2 illustre l'évolution de la précision et du rappel en fonction de la taille de l'incrément. Le temps d'exécution et le rappel augmentent à mesure que la taille de l'incrément croît. On en déduit que, pour l'ER, bien que des mises à jour fréquentes du corpus textuel permettent un traitement plus rapide, elles ont un impact négatif sur les performances du modèle, réduisant sa capacité à identifier correctement les enregistrements similaires. En revanche, une taille d'incrément variant entre 20% et 40% n'a que peu d'effet sur les résultats.

Un rappel élevé indique que la majorité des appariements corrects sont identifiés, ce qui est crucial dans un contexte où d'importants volumes de données en flux continu arrivent rapidement. La réduction du nombre de paires potentielles constitue une base pour les étapes ultérieures qui visent à améliorer progressivement la précision, par exemple en passant d'une approche de traitement en flux à une méthode de traitement par lots plus efficace pour augmenter la précision.

4.2.3. Expériences sur la consommation d'énergie

Dans cette section, nous explorons l'étude de la consommation d'énergie dans les tâches d'apprentissage automatique ainsi que dans les tâches d'ERs.

TABLEAU 3. *Consommation d'énergie en fonction de la taille des incréments*

Incrément (% du pré-entraînement)	Temps d'exécution (s)	Consommation puissance CPU (W)	Consommation énergie CPU (J)	Consommation puissance RAM (W)	Consommation énergie RAM (J)
5%	326.50	1.23	401.76	2.18	711.28
10%	326.50	1.24	405.35	1.89	615.91
20%	332.00	0.97	323.48	2.03	673.21
30%	336.50	1.01	340.55	2.06	694.48
40%	338.00	1.05	354.11	2.08	704.64

Le tableau présente le temps d'exécution, la puissance et la consommation d'énergie calculée lors de l'exécution du programme pour différentes tailles de données incrémentales. À mesure que la quantité de données traitées en une seule session d'entraînement augmente, la consommation d'énergie tend d'abord à diminuer puis à augmenter. En effet, entraîner une grande quantité de données en une seule fois ou mettre à jour trop fréquemment consomme plus d'énergie que d'entraîner à plusieurs reprises une quantité modérée de données.

Bien que la chaleur dégagée par l'ordinateur lors de l'exécution du programme puisse influencer la consommation d'énergie (augmentation des besoins en refroidissement), les mesures effectuées offrent néanmoins une première estimation de l'impact énergétique de cette tâche. Il nous offre une orientation pour nos futures recherches sur l'optimisation de la consommation d'énergie.

De manière générale, il est évident que les embeddings de mots sont efficaces dans une approche incrémentale, permettant d'identifier la majorité des paires similaires correspondant à la même entité. De plus, certaines configurations, comme la taille des incréments, influencent les performances des embeddings de mots et, par conséquent, les résultats de la reconnaissance d'entités. Par ailleurs, ces configurations impactent également la consommation d'énergie durant l'exécution du code. Il est donc nécessaire d'explorer à la fois l'efficacité et la consommation énergétique afin de trouver un équilibre optimal entre les deux, permettant ainsi d'atteindre un objectif sans entraîner une consommation énergétique excessive.

5. Conclusion

Avec la demande croissante d'analyses de flux de données, il est essentiel d'intégrer efficacement les données en streaming avec les données stockées. Dans cette étude, nous avons abordé le défi d'ER dans l'intégration des données en adaptant les techniques d'embedding existantes, initialement conçues pour le traitement par lots, à un contexte de streaming. Nous avons proposé une solution dynamique basée sur un modèle d'embedding incrémental qui met continuellement à jour les représentations au fur et à mesure de l'arrivée des nouvelles données.

Plus précisément, nous avons construit un graphe hétérogène à partir de données relationnelles et l'avons mis à jour de manière incrémentale avec des séquences de données en continu. En exploitant ce graphe évolutif, nous avons effectué des random walks à partir des nouveaux nœuds ajoutés pour générer des phrases, qui ont ensuite été utilisées pour mettre à jour le modèle d'embedding. Notre approche a atteint un rappel allant jusqu'à 81.8%, démontrant ainsi son efficacité dans l'identification de correspondances similaires dans un contexte de streaming. Nous fournissons également des mesures quantitatives de la consommation d'énergie.

Cependant, notre méthode se concentre principalement sur la détection de paires d'enregistrements potentiellement similaires. Comme prochaine étape, nous avons pour objectif d'affiner nos résultats en exploitant les listes de similarité pour identifier

des correspondances plus définitives. Par exemple, en améliorant la méthode de calcul de similarité plutôt que d'utiliser uniquement la similarité cosinus. De plus, nous nous concentrerons sur l'identification des étapes du processus d'ERs en streaming qui consomment le plus d'énergie, dans le but d'optimiser cette partie de l'algorithme et de développer une approche plus efficace et durable.

Bibliographie

Almeida F., Xexéo G. (2023). *Word embeddings: A survey.*

Alvarez-Valera H. H., Maurice A., Ravat F., Song J., Roose P., Valles-Parlangeau N. (2024). Energy measurement system for data lake: An initial approach. In N. T. Nguyen *et al*. (Eds.), *Intelligent information and database systems*, vol. 14795, p. 1527. Singapore, Springer Nature Singapore.

Bahri M., Bifet A., Gama J., Gomes H. M., Maniu S. (2021, mars). Data stream analysis: Foundations, major tasks and tools. *WIREs Data Mining and Knowledge Discovery*, vol. 11, n° 3, p. e1405.

Barros C. D. T., Mendonça M. R. F., Vieira A. B., Ziviani A. (2021, novembre). A survey on embedding dynamic graphs. , vol. 55, n° 1.

Binette O., Steorts R. C. (2022, mars). (almost) all of entity resolution. *Science Advances*, vol. 8, n° 12, p. eabi8021.

Brunner U., Stockinger K. (2020). Entity matching with transformer architectures - a step forward in data integration. In *Proceedings of the 23rd international conference on extending database technology (edbt)*.

Cappuzzo R., Papotti P., Thirumuruganathan S. (2020, juin). Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 acm sigmod international conference on management of data*, p. 13351349. Portland OR USA, ACM.

Christophides V., Efthymiou V., Palpanas T., Papadakis G., Stefanidis K. (2020). An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.*.

Do Nascimento D. C., Santos Pires C. E., Gomes Mestre D. (2018, mars). Heuristic-based approaches for speeding up incremental record linkage. *Journal of Systems and Software*, vol. 137, p. 335354.

Ebraheem M., Thirumuruganathan S., Joty S., Ouzzani M., Tang N. (2018, juillet). Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, vol. 11, n° 11, p. 14541467.

Gazzarri L., Herschel M. (2021, avril). End-to-end task based parallelization for entity resolution on dynamic data. In *2021 ieee 37th international conference on data engineering (icde)*, p. 12481259. Chania, Greece, IEEE.

Gazzarri L., Herschel M. (2023). Progressive entity resolution over incremental data. In *Proceedings of the 26th international conference on extending database technology (edbt)*. OpenProceedings.org.

Genossar B., Shraga R., Gal A. (2023, mai). Flexer: Flexible entity resolution for multiple intents. *Proc. ACM Manag. Data*, vol. 1, n° 1.

Isah H., Abughofa T., Mahfuz S., Ajerla D., Zulkernine F., Khan S. (2019). A survey of distributed data stream processing frameworks. *IEEE Access*, vol. 7, p. 154300-154316.

Li Y., Li J., Suhara Y., Doan A., Tan W.-C. (2020, septembre). Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, vol. 14, n° 1, p. 5060.

Maharana K., Mondal S., Nemade B. (2022). A review: Data pre-processing and data augmentation techniques. *Global Transitions Proceedings*, vol. 3, n° 1, p. 91-99. Consulté sur https://www.sciencedirect.com/science/article/pii/S2666285X22000565 (International Conference on Intelligent Engineering Approach(ICIEA-2022))

Marz N., Warren J. (2015). *Big data: Principles and best practices of scalable realtime data systems* (1st éd.). USA, Manning Publications Co.

Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. (2013). Distributed representations of words and phrases and their compositionality. In, p. 31113119. Red Hook, NY, USA, Curran Associates Inc.

Mudgal S., Li H., Rekatsinas T., Doan A., Park Y., Krishnan G. *et al*. (2018, mai). Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 international conference on management of data*, p. 1934. Houston TX USA, ACM.

Papadakis G., Efthymiou V., Thanos E., Hassanzadeh O., Christen P. (2023, novembre). An analysis of one-to-one matching algorithms for entity resolution. *The VLDB Journal*, vol. 32, n° 6, p. 13691400.

Pennington J., Socher R., Manning C. D. (2014). Glove: Global vectors for word representation. In *Empirical methods in natural language processing (emnlp)*, p. 1532–1543.

Saeedi A., Peukert E., Rahm E. (2020). Incremental multi-source entity resolution for knowledge graph completion. In *The semantic web*, vol. 12123, p. 393408. Cham, Springer International Publishing.

Simonini G., Zecchini L., Bergamaschi S., Naumann F. (2022, mars). Entity resolution on-demand. *Proceedings of the VLDB Endowment*, vol. 15, n° 7, p. 15061518.

Tu J., Fan J., Tang N., Wang P., Li G., Du X. *et al*. (2023, mai). Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data*, vol. 1, n° 1, p. 126.

Wang S., Zhou W., Jiang C. (2020, mars). A survey of word embeddings based on deep learning. *Computing*, vol. 102, n° 3, p. 717740.

Zeng X., Wang P., Mao Y., Chen L., Liu X., Gao Y. (2024). Multiem: Efficient and effective unsupervised multi-table entity matching. In *2024 ieee 40th international conference on data engineering (icde)*, p. 3421-3434.