

INALCO M2 Master ingénierie multilingue 2025

Ingénierie des connaissances : Contrôle n°2 : Correction

NOM
PRÉNOM

I) Révisions

1) Décrivez les étapes principales de la Data preparation et leur utilité. Qu'est ce qu'un connecteur ? À quoi cela sert-il ?

La data préparation désigne l'ensemble des opérations de nettoyage et de transformation des données avant qu'elles ne soient intégrées dans une base de données.

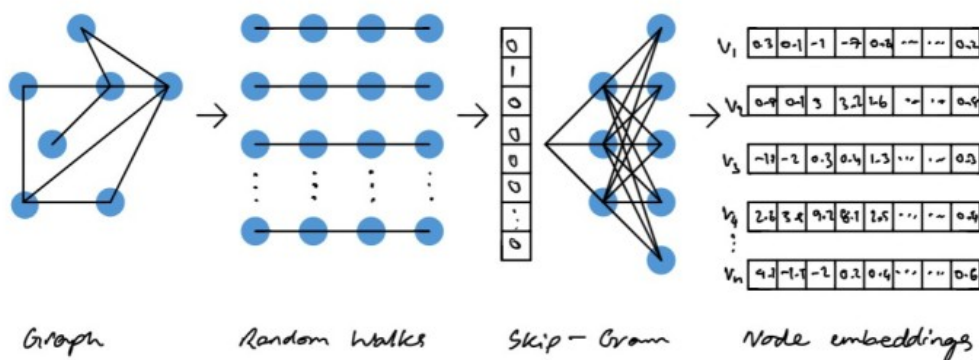
Les étapes principales sont :

- * l'acquisition des données
- * L'exploration (comprendre la distribution des données, détecter les outliers)
- * Leur nettoyage
- * leur transformation / enrichissement
- * Enfin l'ingestion dans la base de données, si possible en continu.

Un connecteur est un système qui permet de relier une source d'information à une base de données. Il permet l'ingestion facile de données en continu dans un système informatique. Le système (base de données, application) est informé dès que de nouvelles informations sont disponibles dans la source, et périodiquement ces nouvelles informations sont transmises, avec une fréquence dépendant du besoin de l'application.

Par exemple pour une application de newsletter, tous les jours les informations de différents sites de journaux sont transmises pour pouvoir être intégrées

2) Comment peut-on construire les embeddings des nœuds d'un graphe ?



Expliquez le fonctionnement de la méthode node2vec. Quelles sont ses limitations et comment peut-on les dépasser ?

La difficulté pour construire l'embedding d'un nœud d'un graphe est qu'il faut à la fois tenir compte du sémantisme du nœud, mais aussi de sa position dans le graphe (liens avec les autres nœuds), ce que les techniques d'embedding classiques peinent à faire.

La méthode node2vec permet de réaliser un encodage de ce positionnement du nœud en s'inspirant de l'idée de word2vec. Dans word2vec, le sens (l'embedding) d'un mot est construit à partir de ses voisins dans les phrases. L'astuce est de réaliser des « phrases de nœuds » et d'utiliser ces phrases pour entraîner le modèle d'embedding. Pour se faire :

- * on procède à des marches aléatoires dans le graphe (on sélectionne un nœud au hasard et passons à un de ses voisins jusqu'à obtenir un chemin de k nœuds. On construit ainsi des « phrases de nœud »

- * on se sert de ces phrases pour entraîner les embeddings à la façon word2vec avec la tâche de prédiction d'un nœud à partir de son contexte.

L'inconvénient de cette méthode est qu'elle définit le sens du nœud uniquement à partir de sa position dans le graphe et pas à partir des attributs de l'objet. Pour avoir des embeddings qui à la fois combinent des informations sur le sens et la position des nœuds, des méthodes à base de transformers ont été réalisées.

II) Nouveaux chapitres

3) La fonction mystère : Voici une fonction vraiment chouette, mais j'ai oublié son nom !

```
def my_little_algo(data, n = 10, max_iters=100):
    keys, vectors = zip(*data.items())
    vectors = np.array(vectors)
    centroids = vectors[random.sample(range(len(vectors)), n)]

    for _ in range(max_iters):
        clusters = {i: [] for i in range(n)}
        for i, vec in enumerate(vectors):
            cluster = np.argmin([np.linalg.norm(vec - c) for c in centroids])
            clusters[cluster].append(i)

        new_centroids = np.array([np.mean(vectors[clusters[i]], axis=0) if clusters[i] else centroids[i]
                                  for i in range(n)])

        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids

    return {keys[i]: cluster for cluster, indices in clusters.items() for i in indices}
```

Quel est cet algorithme ? Expliquez son fonctionnement global, expliquez quels sont ces paramètres puis expliquez ce qu'il fait ligne à ligne.

Il s'agit du fameux algorithme **k means** qui effectue un clustering géométrique d'une liste de vecteurs.

Il prend en entrée un dictionnaire mappant l'id d'un objet à son vecteur, ainsi que k, le nombre de clusters voulus à la fin, et renvoie un dictionnaire affectant l'id de chaque objet à son cluster.

Dans l'algorithme ;

- * on extrait la liste des vecteurs

- * on initialise aléatoirement les k clusters en choisissant un nœud au hasard pour être son « centroïde » (soit censé être le représentant géométrique du vecteur)

Puis la procédure de mise à jour des clusters est déclenchée jusqu'à ce que plus aucune modification ne soit faite.

- * On assigne chacun des vecteurs à un des k clusters en le plaçant dans le cluster dont il est à distance minimale du centroïde.
- * On recalcule la valeur du centroïde comme la moyenne de tous les vecteurs assignés au clustering
- * si les nouveaux centroïdes sont très proches des anciens, on arrête la procédure (sinon après max_iters itérations)

4) Voici une classe python qui sert à réaliser des clusterings hiérarchiques.

```
class HierarchicalClustering:
    def __init__(self, data: List[Any], similarity_func: Callable, termination_cond: Callable):
        self.data = data
        self.similarity_func = similarity_func
        self.termination_cond = termination_cond
        self.clusters = [[item] for item in data] # Start with each item in its own cluster

    def run(self):
        while not self.termination_cond(self.clusters):
            max_sim = float('-inf')
            pair_to_merge = (None, None)
            for i in range(len(self.clusters)):
                for j in range(i + 1, len(self.clusters)):
                    sim = self.similarity_func(self.clusters[i], self.clusters[j])
                    if sim > max_sim:
                        max_sim = sim
                        pair_to_merge = (i, j)
            if pair_to_merge == (None, None):
                i, j = pair_to_merge
                new_cluster = self.clusters[i] + self.clusters[j]
                del self.clusters[j]
                del self.clusters[i]
                self.clusters.append(new_cluster)
        return self.clusters
```

Expliquez l'importance de la condition de terminaison et de la mesure de similarité lorsqu'on fait un clustering hiérarchique.

Quel est le type des variables similarity_func et termination_cond ?

Implémentez une condition de terminaison pour que l'algorithme s'arrête quand on a obtenu k clusters.

Le principe des clustering hiérarchiques est au départ de considérer que chaque nœud est un cluster puis de fusionner les deux clusters les plus similaires jusqu'à obtenir un clustering satisfaisant les propriétés recherchées.

* Il est important de ce fait de définir une mesure de similarité **entre deux clusters et plus entre deux nœuds**. Il existe plusieurs façon de faire (considérer un point représentant du cluster, faire des moyennes...) . La définition de cette mesure de similarité va décider quels clusters vont être fusionnés à chaque étape de l'algorithme et donc fortement influencer le résultat.

* Il est également crucial de définir clairement quand l'algorithme est censé s'arrêter. En effet sans condition de terminaison, on fusionnera tous les nœuds jusqu'à obtenir un unique cluster. On peut par exemple poser une limite minimum à atteindre sur la mesure de similarité pour valider une fusion.

similarity_func et termination_cond sont des fonctions (objets de classe __callable_)

```
def get_k_clusters(clusters, k):
    return len(clusters) <= k
```

5) A quoi sert un système de recommandation ? Quelles sont les données que l'on peut utiliser pour construire un système de recommandation ? Présentez plusieurs approches possibles.

Un système de recommandation sert à conseiller à un utilisateur des contenus qui sont censés correspondre à ses attentes. Pour cela on a besoin de données :

- * sur l'utilisateur
- * sur les contenus qu'il a consommé
- * sur les contenus qu'on veut potentiellement lui proposer
- * potentiellement sur les contenus aimés par des utilisateurs qui ont le même comportement que lui.

Plusieurs types d'approches existent :

- * approches à base d'embeddings des contenus : on construit un embedding des contenus consommés par l'utilisateur pour lui proposer les contenus les plus similaires
- * Les méthodes basées sur un filtrage collaboratif (Collaborative Filtering ou CF). L'idée est de se baser sur les interactions globales des utilisateurs et on déduit ce qui va intéresser une personne à partir de ce qui a intéressé des personnes avec un profil similaire. On utilise pour cela des techniques d'extraction de feature et aussi du clustering (des profils utilisateurs).
- * Le mieux est bien sûr d'utiliser des solutions hybrides qui combinent le meilleur des deux mondes. La plupart du temps ces solutions hybrides reposent sur l'usage de graphes de connaissance. Représenter les données sous forme de graphe permet de voir plus de connexions et de retrouver des connexions entre les objets. Par exemple pour le cas d'un système de recommandation de vidéos les informations sur le film (genre, acteurs, réalisateur) permettent de mieux cerner les goûts de l'utilisateur et lui proposer du contenu proche thématiquement, ou qu'a apprécié un utilisateur avec les mêmes patterns.

6) Définissez ce qu'est le Graph RAG.

Quelles sont les différentes approches qui ont été explorées pour l'implémenter?

Le RAG (Retrieval Augmented Generation) consiste à extraire des informations d'une base de documents pour répondre à une question d'un utilisateur grâce à un LLM.

Le Graph RAG est une variante qui consiste à requêter des informations dans des graphes ou bases de connaissance (base structurée) et non dans des documents.

Le terme de « Graph RAG » a fini par désigner des techniques et des approches qui n'avaient rien à voir entre elles.

1) Extraction de nœuds et de relations

Cette approche consiste à récupérer les composants d'un graphe de connaissances, c'est-à-dire ses sommets (nœuds) et ses arêtes (relations) à partir d'une recherche vectorielle. ON construit un embedding des nœuds / relations et un embedding de la question de l'utilisateur et on récupère les éléments qui lui sont le plus similaires.

2) Regroupement de graphes et résumé de clusters

Cette technique implique de regrouper des nœuds similaires en clusters et de sélectionner les clusters les plus pertinents pour répondre à la requête. En résumant les clusters, le système réduit la complexité du graphe avant d'interagir avec le LLM.

3) Transformation de la requête en requête de graph database

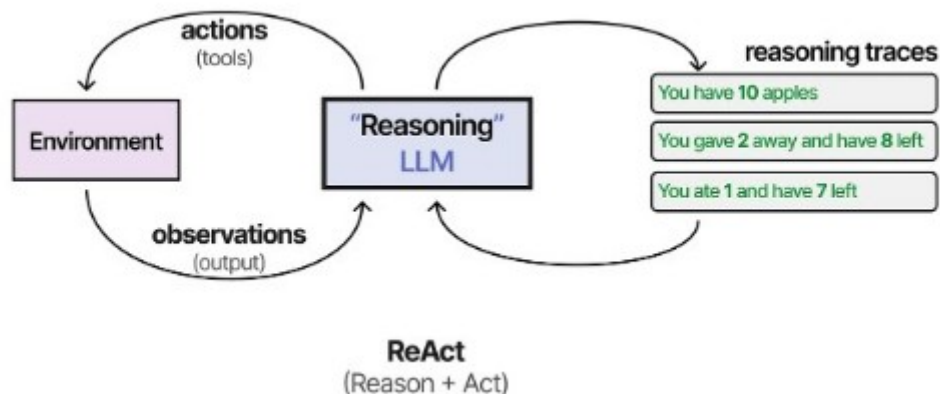
Inspirée par les techniques de text-to-SQL, cette approche convertit la requête en langage naturel de l'utilisateur en une requête de base de données de graphes (par exemple, en utilisant Cypher pour Neo4j). La requête de graphe est ensuite exécutée pour extraire le sous-graphe le plus pertinent que le LLM doit traiter.

7) Quels sont les composants nécessaires pour construire un agent IA ? Expliquez leur rôle. Présentez l'architecture ReAct à l'aide d'un schéma.

Un agent IA est un programme qui connecte un modèle d'IA générative (LLM, VLM...) à un environnement extérieur et lui permet de réaliser des tâches complexes en autonomie.

Un agent IA comporte généralement :

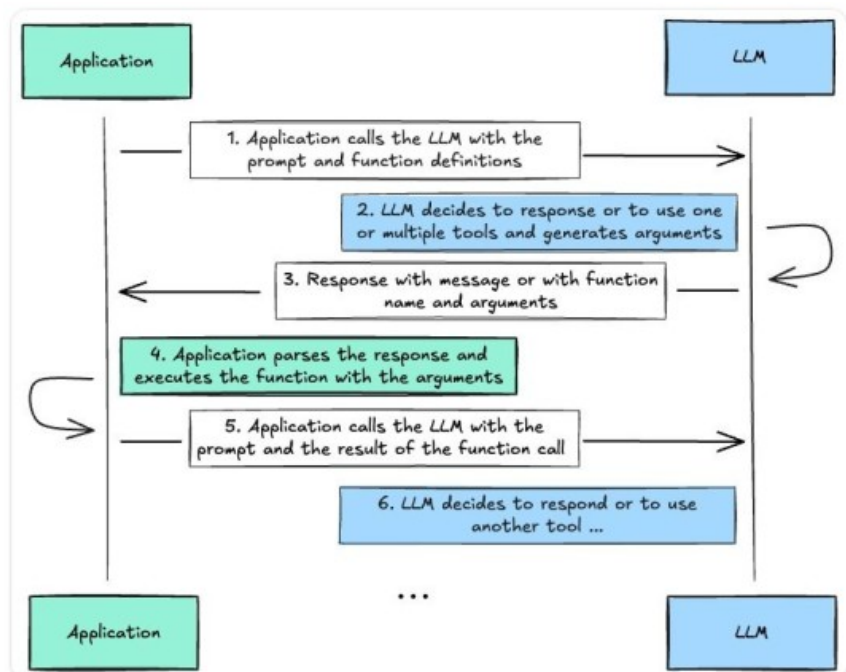
- * un LLM pour générer les réponses, contenus, gérer la planification
- * des outils que le LLM peut utiliser pour interagir avec son environnement. Ces outils sont exécutés par l'application et la réponse de ces outils est renvoyée comme contexte pour une génération de LLM.
- * un système de gestion de mémoire qui va aider à remplir la fenêtre de contexte du LLM avant chaque nouvelle génération.



Dans un agent de type ReACT, le LLM planifie sa prochaine action à partir d'informations recueillies sur son environnement (mémoire + outils de collecte de données), l'effectue, modifiant potentiellement l'environnement, et ainsi de suite jusqu'à ce que sa mission initiale soit accomplie.

8) Qu'est ce que le *function calling* (ou *tool calling*) ?

Expliquez les étapes de ce mécanisme en précisant quelle programme l'exécute (LLM, application...)



Le function calling est un trait de comportement des LLMs les plus récents qui quand on leur fournit dans leur prompt système une liste d'outils dans un format attendu leur permet, dans un format structuré, de signifier qu'il considèrent qu'il faut utiliser un de ces outils pour répondre à la requête de l'utilisateur. Ce trait de comportement est obtenu par un entraînement spécifique lors de la phase d'alignement du modèle. On nomme function calling également l'utilisation de ce mécanisme dans un programme.

Le mécanisme est une sorte de va et vient entre le LLM et le programme qui l'appelle :

1) L'utilisateur envoie une requête.

Si le modèle considère qu'il peut répondre sans utiliser d'outil, il renvoie une réponse texte comme d'habitude. (pas de tool call)

si il considère qu'un outil doit être utilisé, le LLM renvoie un objet spécifique mentionnant le nom de l'outil et les arguments.

2) L'application appelant le LLM parse sa réponse pour savoir si un outil doit être utilisé. Si oui elle exécute l'outil avec les arguments donnés par le LLM

3) La réponse de l'outil est renvoyée au LLM dans la suite de la conversation pour lui permettre de répondre à la question initiale posée en 1. Le LLM peut avec cette nouvelle information soit enfin répondre soit à nouveau appeler un outil.

Quand un outil est appelé, il y a donc deux générations réalisées par le LLM.

III) Culture scientifique et lecture d'articles.

9) Expliquez la stratégie nommée « Graph RAG » présentée dans le papier « From Local to Global: A Graph RAG Approach to Query-Focused Summarization ». Quelles sont les limites de cette approche ?

Dans ce papier de Microsoft, les chercheurs combinent le RAG avec la tâche de Query Focused Summarization (faire un résumé de texte guidé par une question spécifique de l'utilisateur). Pour se faire, un important travail de preprocessing est nécessaire.

- * construction d'un graphe de connaissance par LLM

- * partition de ce graphe selon une hiérarchie de communautés à l'aide de l'algorithme de Louvain

- * génération d'un résumé pour chaque communauté

Quand l'utilisateur pose une question, approche map reduce est utilisée pour générer des réponses partielles à la question de l'utilisateur à partir des informations contenues dans les résumés de communauté. On construit la réponse finale à partir des résumés les plus intéressants (approche RAG documentaire classique)

Cette approche reste extrêmement coûteuse en appel de LLMs à la fois pour préparer la base de données et au moment de poser chaque question.

Les bénéfices de cette technique ne se font de plus sentir que sur des questions de sens très général.

10) Quels sont les différents paramètres qui peuvent jouer sur la compréhension d'un graphe par un LLM (4 points de réponse minimum attendus)

De nombreux facteurs peuvent influencer la compréhension d'un graphe par un LLM (cf article 3) :

- * le prompt utilisé pour présenter le graphe et la tâche à effectuer

- * la topologie du graphe (sa forme) et sa taille


- * la manière d'encoder le graphe pour le décrire (utiliser des lettres, des noms, une représentation informatique comme une liste d'adjacence...)

- * la nature du graphe (ce qu'il représente dans la réalité)

- * la taille du LLM et ses capacités de raisonnement

BONUS :

le retour de Jérémie : . Commentez cette publication à partir de vos connaissances.



Jérémie Ravenel ↑ Nouveaux posts
⚡ Building bridges @naas.ai Universal Data & AI Platform | Research ...
[Accéder à mon site web](#)
2 j • 🌐

Why are agents just the tip of the iceberg?

Because behind every intelligent agent lies a powerful agentic infrastructure, the invisible system that makes everything possible.

Everyone is talking about AI agents right now. But focusing only on the agent is like praising a great dish without acknowledging the kitchen, the chef, or the ingredients. The real innovation happens beneath the surface.

Here are the 7 core components of an agentic infrastructure:

1. Agents: the interface layer, executing tasks and interacting with users.
2. Integrations: the bridges connecting the agent to external systems, tools, and APIs.
3. Pipelines: the flow of data from raw sources to usable outputs.
4. Workflows: the logic and orchestration that guide how tasks unfold across tools and teams.
5. Ontologies: the brain of the system, ensuring shared understanding across humans and machines.
6. Analytics: the feedback loop that helps the system learn, optimize, and make informed decisions.
7. Apps: the productized outputs that package value into tangible, usable interfaces.

You don't build agents.
You build systems that enable agents to operate with intelligence, autonomy, and reliability.

That's where the magic really happens.
That's how AI stops being a gimmick and becomes infrastructure.

Dans ce post, Jérémie insiste sur le fait que malgré l'intérêt et les discussions sur l'architecture des agents IA et leurs capacités, il est également nécessaire de penser à l'infrastructure dans laquelle ces programmes vont être exécutés. Il emploie la métaphore du cuisinier (agent) faisant un plat qui ne peut rien faire si la cuisine est en mauvais état.

C'est pourquoi il ne faut pas parler seulement des

agents mais de toute l'« infrastructure agentique » qui comprend les agents, mais aussi les applications qui vont les utiliser, leur intégration avec d'autres sources de données / programme... Jérémie ne mentionne pas l'infrastructure matérielle qui est aussi un point critique .

On peut lui reprocher dans son énumération de mélanger des éléments qui n'ont rien à voir entre eux ou sont définis de façon floue :

pipeline : terme générique en informatique, redondant ici avec le problème de l'intégration.

Workflow : parle t on des workflows des équipes humaines qui vont utiliser des agents ? Il y a un débat sur la limite entre agents IA et workflows intégrant de l'IA générative. Dur à dire de quel type de workflow il parle.

Le rôle des ontologies n'est pas justifié ici.

Si la formulation de certains passages et la clarté du propos laissent parfois à désirer, ce post souligne bien le fait que les agents IA ne sont qu'un composant de systèmes plus vaste et que les autres composants de ce système méritent également des efforts de réflexion pour parvenir à leur bonne intégration.