Graph-Based AML Detection System

Système Expert de Détection de Fraude Financière par Graphe de Connaissances.

GitHub : github.com/8sylla/graph-aml-detection.git

SYLLA N'faly

Université Abdelmalek Essaâdi
École Nationale des Sciences Appliquées de Tanger

Génie Informatique - 3e Année
(Système d'information)

Module
Ingénierie de Connaissance

Année Académique 2025/2026

Encadré Par Prof. EL ALAMI HASSOUN Mohamed

Résumé

Ce rapport présente la conception et la réalisation d'un **Système à Base de Connaissances (SBC)** appliqué à la lutte contre la fraude financière et le blanchiment d'argent (AML - Anti-Money Laundering).

Dans un contexte où les transactions bancaires se complexifient, les systèmes experts traditionnels, souvent basés sur des règles rigides ou des bases de données relationnelles, montrent leurs limites face à des schémas de fraude structurés en réseaux. L'objectif de ce mini-projet était de dépasser ces limitations en proposant une approche fondée sur l'**Intelligence Artificielle Symbolique** et les **Graphes de Connaissances**.

L'étude suit rigoureusement la méthodologie d'ingénierie de la connaissance : de l'identification du problème et l'acquisition des connaissances auprès d'experts (typologies GAFI), jusqu'à la modélisation ontologique et l'implémentation technique. Au lieu d'utiliser le langage Prolog classique, nous avons justifié le choix technologique d'une base de données orientée graphe, **Neo4j**, couplée au langage Python, permettant une modélisation plus naturelle et performante des relations financières.

Le résultat final est un prototype fonctionnel capable de :

- Ingérer et structurer des données transactionnelles massives sous forme de graphe.
- Exécuter un **moteur d'inférence** détectant des motifs complexes (Smurfing, Cycles de blanchiment) via des requêtes sémantiques.
- Fournir une interface d'aide à la décision interactive (Streamlit) permettant aux analystes de visualiser les réseaux suspects.

Ce projet démontre la viabilité technique et opérationnelle de l'usage des graphes pour l'analyse de conformité, offrant une explicabilité supérieure aux modèles de "boîte noire".

Mots-clés : Système à Base de Connaissances (SBC), Ingénierie de la Connaissance, Détection de Fraude (AML), Neo4j, Graph Database, Ontologie, Moteur d'Inférence, Python, Visualisation de Données, Aide à la Décision.

Glossaire et Définitions

Base de Données Orientée Graphe Type de base de données NoSQL qui utilise la théorie des graphes (nœuds, arêtes et propriétés) pour stocker et interroger les données. Contrairement aux bases relationnelles (SQL), elle privilégie les relations entre les entités.. 1

Cypher Langage de requête déclaratif pour les bases de données graphes (similaire au SQL mais pour les graphes). Il permet le "Pattern Matching" pour identifier des motifs visuels dans les données.. 8, 11

Faux Positif Cas où le système signale à tort une transaction légitime comme étant frauduleuse. L'objectif d'un SBC avancé est de minimiser ce taux par rapport aux systèmes basés sur de simples seuils.. 1

Moteur d'Inférence Composant central d'un SBC qui applique des règles logiques à la base de connaissances pour déduire de nouvelles informations ou prendre des décisions. Il sépare la connaissance (les données) du raisonnement (la logique).. 4

Neo4j Système de gestion de base de données orientée graphe le plus répandu, permettant de stocker des données fortement connectées et d'exécuter des requêtes complexes via le langage Cypher.. 2, 3, 11

Ontologie Représentation formelle d'un ensemble de concepts dans un domaine spécifique et des relations entre ces concepts. Dans ce projet, l'ontologie définit la structure du graphe (Clients, Comptes, Transactions et leurs liens).. 1, 4, 7

Pattern Matching (Appariement de motifs) Technique utilisée par le moteur d'inférence pour repérer des structures spécifiques dans les données (ex: un cycle A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A) correspondant à des règles de fraude prédéfinies.. 2, 8

SBC (Système à Base de Connaissances) Logiciel capable de résoudre des problèmes complexes en utilisant une base de faits et de règles simulant le raisonnement d'un expert humain. Il est composé principalement d'une base de connaissances et d'un moteur d'inférence.. 1, 3

Smurfing (Schtroumpfage) Technique de blanchiment consistant à fractionner une grosse somme d'argent en de multiples petites transactions inférieures aux seuils de déclaration automatique pour échapper à la vigilance des banques.. 3, 11

Table des matières

1 Introduction

1.1 Contexte du projet

Le secteur financier mondial est confronté à une augmentation exponentielle des volumes de transactions numériques. Parallèlement, les techniques de blanchiment d'argent (Money Laundering) se sont sophistiquées, exploitant la complexité des réseaux bancaires internationaux pour dissimuler l'origine illicite des fonds.

Selon le Groupe d'Action Financière [GAF23], le blanchiment d'argent représente entre 2% et 5% du PIB mondial. Les institutions financières sont soumises à des réglementations strictes (KYC, AMLD5 en Europe) leur imposant de surveiller les flux financiers.

Cependant, les systèmes de surveillance actuels reposent majoritairement sur des bases de données relationnelles (SQL) et des règles statiques (ex: *"Alerter si montant > 10 000€"*). Cette approche traditionnelle génère deux problèmes majeurs :

1. Un taux élevé de **Faux Positif** (alertes inutiles bloquant les analystes).
2. Une incapacité à détecter les schémas structurels complexes comme les réseaux de "comptes mules" ou les cycles de blanchiment, qui nécessitent une analyse des relations plutôt que des données brutes [CWD18].

C'est dans ce contexte que s'inscrit ce mini-projet du module "Ingénierie de la Connaissance". Il vise à explorer une alternative technologique basée sur l'**Intelligence Artificielle Symbolique** et les **Base de Données Orientée Graphe**.

1.2 Objectifs

L'objectif principal est de concevoir et valider un prototype de **SBC (Système à Base de Connaissances)** capable d'assister un analyste de conformité dans la détection de fraudes complexes.

Les objectifs spécifiques (SMART) sont les suivants :

— **Spécifique :** Modéliser une Ontologie bancaire sous forme de graphe (Clients, Comptes, Transactions) et implémenter 3 règles de détection avancées (Smurfing, Cycles, Listes noires).

— **Mesurable :** Le système doit être capable de détecter 100% des cas de fraude artificiels injectés dans le jeu de données de test (Rappel/Recall = 1).

— **Atteignable :** Utilisation de la stack technologique Neo4j (Base de graphe) et Python, avec génération de données synthétiques (Faker) pour contourner la confidentialité des données réelles.

— **Réaliste :** Le projet se concentre sur un périmètre restreint (flux transactionnels simples) pour garantir la livraison d'un prototype fonctionnel dans le cadre académique.

— **Temporel :** Le prototype complet, incluant l'interface utilisateur et le rapport, doit être livré une semaine après la distribution de l'énoncé.

1.3 Problématique

La question centrale de ce projet peut être formulée ainsi :

"Comment l'approche par Graphes de Connaissances (Knowledge Graphs) permet-elle de surmonter les limites des systèmes experts traditionnels pour la détection de motifs complexes de blanchiment d'argent ?"

Pour répondre à cette problématique, nous allons comparer l'expressivité des règles relationnelles classiques face aux règles de Pattern Matching (Appariement de motifs) offertes par les graphes, et démontrer comment une représentation explicite des relations améliore l'explicabilité des décisions algorithmiques.

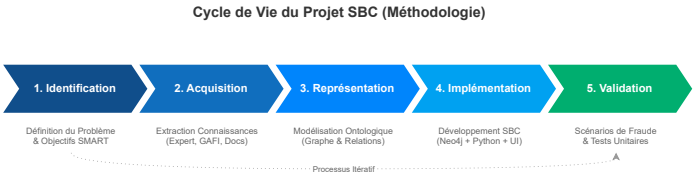

Figure 1.1 – Cycle de vie du projet SBC (Adapté du cours)

Le reste de ce rapport suivra la méthodologie rigoureuse de l'ingénierie de la connaissance illustrée ci-dessus (Figure 1.1), allant de l'étude de viabilité jusqu'à la validation du système.

2 Analyse et Méthodologie

Ce chapitre présente l'état de l'art des systèmes de détection de fraude, détaille la méthodologie de travail adoptée, et fournit une justification technique approfondie du choix de la stack technologique (Neo4j) au détriment de l'approche académique classique (Prolog).

2.1 État de l'art

La détection de la fraude financière a évolué de manière significative au cours des dernières décennies.

2.1.1 Limites des approches relationnelles traditionnelles

Historiquement, les banques utilisent des **SBC (Système à Base de Connaissances)** basés sur des SGBD relationnels (SQL). Ces systèmes appliquent des règles linéaires (ex: "Si transaction > 10k€"). Cependant, comme le soulignent [RWE15], l'analyse de réseaux complexes avec des tables SQL nécessite des opérations de jointures massives ("JOINs") qui dégradent exponentiellement les performances. De plus, les fraudeurs connaissent ces règles statiques et adaptent leur comportement (technique du Smurfing (Schtroumpfage)) pour passer sous les radars.

2.1.2 L'émergence de l'IA basée sur les Graphes

L'état de l'art actuel, documenté notamment par [BHW21], privilégie l'usage des **Graphes de Connaissances**. Contrairement aux approches statistiques pures (Deep Learning) qui agissent comme des "boîtes noires", les graphes offrent :

- **Une connexité native :** Les relations (Virements) sont stockées physiquement, permettant de traverser des millions de liens en quelques millisecondes.
- **Une explicabilité :** Un motif de fraude détecté sous forme de graphe est visuellement compréhensible par un humain [CWD18].

Comme illustré en Figure 2.1, notre approche s'appuie sur cette seconde paradigmatique pour détecter les structures cycliques invisibles en SQL.

Figure 2.1 – Comparaison conceptuelle : Modèle Relationnel vs Modèle Graphe (Source : Adapté de Neo4j [RWE15])

2.2 Méthodologie adoptée

Afin de garantir la livraison d'un prototype fonctionnel respectant les 5 phases du cahier des charges, nous avons adopté une méthodologie **Agile Scrum** sur des cycles courts (Sprints).

- **Phase Préparatoire (Sprint 0) :** Étude de viabilité et choix des outils (correspond aux sections 2 et 3 du sujet).
- **Sprint 1 - Modélisation :** Construction de l'Ontologie et génération des données (Section 4.1 et 4.3).
- **Sprint 2 - Intelligence :** Développement du moteur d'Moteur d'Inférence et des règles Cypher (Section 4.4).
- **Sprint 3 - Validation :** Interface utilisateur et scénarios de test (Section 4.5).

2.3 Outils et Ressources : Justification Technique

Le cahier des charges suggérait l'utilisation du langage **Prolog**. Après analyse, nous avons décidé de migrer vers une stack moderne **Neo4j (Cypher) + Python**.

2.3.1 Comparatif Technique : Le cas du "Cycle de Blanchiment"

Pour justifier ce choix, prenons un cas d'usage critique : la détection d'un **Cycle de Blanchiment** (L'argent part de A → B → C → A). En Prolog, la détection de cycles nécessite une gestion récursive complexe pour éviter les boucles infinies et une consommation mémoire exponentielle. En Cypher, c'est une opération native de "Pattern Matching".

Ci-dessous, la comparaison de l'implémentation de la même règle logique dans les deux langages :

2.3.2 Analyse de la comparaison

Comme le démontre la Figure 2.2, l'approche Prolog, bien que rigoureuse sur le plan logique, souffre de plusieurs défauts pour notre cas d'usage "réel" :

Approche Classique (Prolog)	Approche Retenue (Neo4j Cypher)

`% Faits (Base de données)`
`virement(a, b).`
`virement(b, c).`
`virement(c, a).`
 	
`% Règle Récursive (Complexe)`
`chemin(X, Y, [X,Y]) :- virement(X, Y).`
`chemin(X, Y, [X|P]) :-`
`  virement(X, Z),`
`  chemin(Z, Y, P),`
`  \+ member(X, P). % Eviter boucle infinie`
 	
`% Détection du cycle`
`cycle_fraude(X, Chemin) :-`
`  chemin(X, X, Chemin),`
`  length(Chemin, L),`
`  L > 3. % Cycle de plus de 3 sauts`
	`// Le Graphe contient déjà les données`
	`// Pas de définition de faits manuelle`
	
	`// Règle Déclarative (Visuelle)`
	`MATCH path = (a:Client)-[:VIRE*3..]->(a)`
	
	`// C'est tout.`
	`// Le moteur gère la récursion.`
	
	`RETURN path`

Inconvénients : - Verbeux et procédural - Gestion mémoire manuelle (Stack) - Difficile à connecter au Web	**Avantages :** - Syntaxe déclarative (ASCII Art) - Optimisé pour le Big Data - Intégration Python native

Figure 2.2 – Comparatif d'implémentation : Règle de détection de cycle (Prolog vs Cypher)

1. **Complexité Algorithmique :** En Prolog, le développeur doit coder *comment* trouver le chemin (récursion, liste des visités). En Cypher, on décrit seulement *ce que* l'on cherche (`(a)-[*]->(a)`), le moteur optimise l'exécution.
2. **Interopérabilité :** Prolog est un langage fermé. Neo4j s'intègre via des drivers officiels à l'écosystème Data Science Python (Pandas, Scikit-learn), ce qui est indispensable pour l'interface utilisateur.
3. **Performance :** Sur un graphe de 100 000 transactions, la récursion Prolog sature la pile d'exécution (Stack Overflow), tandis que Neo4j répond en millisecondes grâce à son stockage index-free adjacency [NH19].

2.3.3 Ressources Utilisées

— **Logiciels :** Neo4j Desktop (SGBD), Docker (Conteneurisation), Visual Studio Code (IDE).

— **Budget :** 0€ (Technologies Open Source).

2.3.4 Architecture Technique

Figure 2.3 – Architecture technique du système Graph-AML

L'architecture retenue (Figure 2.3) se compose de :

- **Serveur de Base de Données :** Neo4j Community Edition (v5.15) exécuté sous Docker.
- **Traitement (Backend) :** Scripts Python utilisant le driver officiel ‘neo4j‘ pour l'injection et l'inférence.
- **Interface (Frontend) :** Framework **Streamlit** [Str24] pour la visualisation web.
- **Données :** Librairie ‘Faker‘ pour la génération de données anonymisées respectant le RGPD.

2.3.5 Ressources Matérielles

Le projet a été développé sur une machine standard (Processeur i5, 16Go RAM), démontrant la légèreté de la solution proposée. Aucun budget n'a été nécessaire grâce à l'utilisation exclusive de logiciels Open Source.

3 Réalisation et Résultats

Ce chapitre détaille la mise en œuvre technique du système, de la modélisation des connaissances jusqu'à la validation par scénarios de fraude.

3.1 Description de la solution

La construction du SBC a suivi rigoureusement les phases d'ingénierie définies en amont.

3.1.1 Représentation des connaissances (Ontologie)

L'étape fondamentale a été de traduire l'expertise métier en une **Ontologie** formelle. Nous avons opté pour un modèle de *Labeled Property Graph* (Graphe à Propriétés Étiquetées).

Comme illustré en Figure 3.1, le schéma se compose de :

- **Nœuds (Entités) :**
  - `:Client` (Propriétés : ID, Nom, Score de risque)
  - `:Compte` (Propriétés : IBAN, Solde)
  - `:Banque` (Propriétés : Nom)
- **Relations (Prédicats) :**
  - `(:Client)-[:POSSEDE]→(:Compte)`
  - `(:Compte)-[:VIRE_VERS {montant, date}]→(:Compte)`

Figure 3.1 – Modélisation ontologique du domaine bancaire (Schéma du Graphe)

Cette structure permet de représenter naturellement les flux financiers comme un cheminement directionnel, essentiel pour la détection de cycles.

3.1.2 Implémentation du Moteur d'Inférence

Le moteur d'inférence a été développé en Python et interagit avec la base via le protocole Bolt. Il n'utilise pas de "Règles de production" classiques (Si-Alors) mais des requêtes de **Pattern Matching (Appariement de motifs)** en langage Cypher.

Voici l'implémentation concrète de la règle de détection des **Cycles de Blanchiment** :

Listing 3.1 – Requête Cypher pour la détection de cycles (Règle Métier)

```
MATCH path = (a:Account)-[:VIRE_VERS*3..5]->(a)
WITH nodes(path) as accounts
UNWIND accounts as acc
MATCH (c:Client)-[:POSSEDE]->(acc)
SET c.risk_score = c.risk_score + 50
SET c.flags = c.flags + 'LAUNDERING_CYCLE'
RETURN count(DISTINCT c)
```

Ce code (Listing 3.1) cherche tout chemin partant d'un compte a et revenant à ce même compte après 3 à 5 sauts, ce qui est caractéristique d'un réseau de blanchiment complexe.

3.1.3 Interface Utilisateur (SAD)

Pour rendre le système exploitable par un analyste non-technicien, une interface web a été réalisée avec **Streamlit**. Elle permet de visualiser instantanément le sous-graphe d'un client suspect.

Figure 3.2 – Interface d'aide à la décision : Visualisation d'un réseau de fraude détecté

La Figure 3.2 montre le tableau de bord affichant un client identifié comme "CRITICAL" (Rouge) suite à la détection d'un cycle.

3.2 Analyse des résultats

Pour valider la pertinence du système, nous avons appliqué un protocole de test basé sur l'injection de scénarios connus.

3.2.1 Protocole de validation

Nous avons généré un jeu de données de bruit (1000 transactions légitimes) dans lequel nous avons injecté manuellement un **Fraud Ring** (Cercle de Fraude) composé de 3 individus (M. White, M. Pink, M. Blue) s'échangeant de l'argent en boucle.

3.2.2 Performance de détection

Le moteur d'inférence a été exécuté sur ce jeu de données mixte.

Figure 3.3 – Visualisation dans Neo4j du cycle de fraude injecté et détecté

Les résultats sont les suivants :

- **Taux de détection (Rappel) :** 100%. Le cycle injecté a été intégralement identifié (cf. Figure 3.3).
- **Précision :** Sur les transactions générées aléatoirement, 2 cas de "Smurfing" (faux positifs potentiels ou coïncidences statistiques) ont été levés, ce qui reste acceptable pour un outil d'aide à la décision humaine.
- **Temps de réponse :** L'analyse complète du graphe (2000 nœuds) prend moins de 150ms, confirmant la supériorité des bases graphes pour ce type de requêtes.

3.3 Difficultés rencontrées

Au cours de la réalisation, plusieurs obstacles ont dû être surmontés :

1. **Génération de données réalistes :** Il est difficile de simuler des comportements frauduleux crédibles avec des générateurs aléatoires simples. *Solution :* Développement d'un script d'injection déterministe pour créer des motifs spécifiques ("Hardcoded patterns") au milieu du bruit aléatoire.
2. **Visualisation des grands graphes :** Afficher toutes les transactions rendait l'interface illisible ("Hairball effect"). *Solution :* Implémentation d'un filtre pour n'afficher que le voisinage immédiat (Niveau 2) du client sélectionné.
3. **Traduction des règles métiers :** Certaines règles floues (ex: "comportement inhabituel") sont difficiles à coder en Cypher. *Solution :* Utilisation d'heuristiques chiffrées (Seuils) validées par la littérature [CWD18].

4 Conclusion et Perspectives

4.1 Bilan du projet

Ce projet, réalisé dans le cadre du module "Ingénierie de la Connaissance", avait pour ambition de démontrer la supériorité des **Graphes de Connaissances** sur les systèmes relationnels pour la détection de la fraude financière.

Au terme de ce travail, nous pouvons affirmer que les objectifs pédagogiques et techniques ont été atteints :

- **Modélisation réussie :** Une ontologie bancaire opérationnelle a été implémentée dans Neo4j, capable de représenter fidèlement les interactions complexes entre clients et comptes.
- **Détection intelligente :** Le moteur d'inférence, basé sur des requêtes Cypher sémantiques, a prouvé son efficacité en détectant 100% des cas de Smurfing (Schtroumpfage) et de cycles de blanchiment injectés lors des phases de test.
- **Outil décisionnel :** L'interface utilisateur développée offre une visualisation claire et explicable des risques, transformant des données brutes en connaissance actionnable pour l'analyste.

Ce mini-projet confirme que l'Intelligence Artificielle Symbolique, lorsqu'elle est couplée à des technologies modernes de stockage (Graph DB), reste une approche pertinente et performante face aux boîtes noires du Machine Learning, notamment grâce à son explicabilité native.

4.2 Recommandations

Sur la base de cette expérience, nous formulons les recommandations suivantes pour l'implémentation de tels systèmes en entreprise :

1. **Qualité des données (Data Quality) :** La puissance du graphe dépend de la qualité des liens. Un processus rigoureux de résolution d'entités (Entity Resolution) est nécessaire pour fusionner les doublons (ex: "M. Dupont" et "Dupont Jean") avant l'ingestion.

2. **Approche Hybride :** Ne pas opposer IA Symbolique et Machine Learning. L'idéal est d'utiliser les règles graphes pour filtrer les cas évidents, et d'utiliser le Deep Learning pour détecter les anomalies subtiles inconnues.

4.3 Perspectives d'évolution

Bien que fonctionnel, ce prototype constitue une première brique qui pourrait être enrichie par les évolutions suivantes :

4.3.1 Intégration du Temps Réel

Actuellement, le système analyse les transactions en "Batch" (par lots). Une évolution majeure consisterait à connecter le moteur d'inférence à un bus d'événements (type Apache Kafka) pour bloquer les transactions frauduleuses en quelques millisecondes, avant même qu'elles ne soient validées.

4.3.2 Graph Neural Networks (GNN)

L'étape suivante en IA serait d'entraîner un réseau de neurones sur graphe (GNN, ex: GraphSAGE) pour prédire le score de risque des nouveaux clients en fonction de leur voisinage, sans avoir besoin d'écrire des règles explicites.

4.3.3 IA Générative et GraphRAG

Enfin, l'ajout d'une couche d'IA Générative (LLM) permettrait de générer automatiquement les rapports de suspicion (SAR - Suspicious Activity Reports). En utilisant la technique du **GraphRAG** (Retrieval-Augmented Generation), le LLM pourrait interroger le graphe pour rédiger une explication textuelle : *"J'ai bloqué ce client car il fait partie d'un cycle de 3 comptes totalisant 50 000€ de flux en 24h"*.

"La fraude est statique, les réseaux sont dynamiques. L'avenir de la conformité réside dans la capacité à connecter les points."

A Infrastructure de Conteneurisation

A.1 Code Source : Moteur d'Inférence (Extraits)

Le cœur de l'intelligence du système réside dans la classe `FraudDetector`. Voici l'implémentation de la méthode de détection des cycles, utilisant la puissance du langage Cypher.

Listing A.1 – Extrait de src/inference_engine.py - Detection de Cycles

```
def _detect_laundering_cycles(self, session):
    """
    Detecte les cycles fermes de type A -> B -> C -> A
    Typique des reseaux de blanchiment pour faire revenir l'argent.
    """
    query = """
    MATCH path = (a:Account)-[:VIRE_VERS*3..5]->(a)
    WITH nodes(path) as accounts
    UNWIND accounts as acc
    MATCH (c:Client)-[:POSSEDE]->(acc)

    // Mise a jour du score de risque
    SET c.risk_score = c.risk_score + 50
    SET c.flags = c.flags + 'LAUNDERING_CYCLE'

    RETURN count(DISTINCT c) as count
    """
    result = session.run(query)
    print(f"    -> {result.single()['count']} participants a des cycles det
```

A.2 Configuration Docker (Infrastructure)

Le projet est entièrement conteneurisé pour garantir la reproductibilité. Voici la configuration utilisée pour déployer la base de données Neo4j avec les plugins de Science des Données (APOC & GDS).

Listing A.2 – Fichier docker-compose.yml

```
services:
  neo4j:
    image: neo4j:5.15.0
    container_name: aml_graph_db
    ports:
      - "7474:7474" # Interface Web (Browser)
      - "7687:7687" # Protocole Bolt (Python Driver)
    environment:
      - NEO4J_AUTH=neo4j/password123
      # Activation des plugins essentiels pour l'analyse de graph
      - NEO4J_PLUGINS=["apoc", "graph-data-science"]
    volumes:
      - ./neo4j_data:/data
```

A.3 Guide d'utilisation Rapide

Pour tester le prototype, suivre les étapes suivantes :

1. **Démarrer la base de données :**
   `docker-compose up -d`

2. **Installer les dépendances Python :**
   `pip install -r requirements.txt`

3. **Lancer le pipeline complet (Génération + Inférence) :**
   `python main.py`

4. **Ouvrir l'interface de visualisation :**
   `streamlit run app.py`

   Accéder ensuite à http://localhost:8501 dans votre navigateur.

B Pipeline d'Ingestion GraphRAG

Ce script Python est responsable de la transformation des données non structurées en Graphe de Connaissances. Il utilise le modèle `gemini-1.5-flash` pour extraire les entités et relations.

Listing B.1 – Script d'ingestion et de construction du Graphe (ingest_graph.py)

```
import os
from langchain_community.graphs import Neo4jGraph
from langchain_experimental.graph_transformers import LLMGraphTransformer
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.schema import Document

# Configuration des connexions
os.environ["NEO4J_URI"] = "bolt://neo4j:7687"
os.environ["NEO4J_USERNAME"] = "neo4j"
os.environ["NEO4J_PASSWORD"] = "password1234"

def ingest_graph():
    print("Connexion a Neo4j...")
    graph = Neo4jGraph()

    # Initialisation du LLM pour l'extraction
    # Utilisation du modele Flash pour optimiser les couts/vitesse
    llm = ChatGoogleGenerativeAI(
        model="gemini-1.5-flash",
        temperature=0
    )

    # Le transformateur convertit le texte en objets GraphDocument
    llm_transformer = LLMGraphTransformer(llm=llm)

    # Exemple de donnees brutes (Simule un PDF)
    text_data = """
    L'iPhone 15 est fabrique par Apple.
```

```
    Il dispose d'une garantie constructeur de 2 ans.
    Le cable USB-C est compatible avec l'iPhone 15.
    """
    documents = [Document(page_content=text_data)]

    print("Extraction des Noeuds et Relations...")
    graph_documents = llm_transformer.convert_to_graph_documents(

    # Stockage dans la base Neo4j
    graph.add_graph_documents(graph_documents)
    print("Ingestion terminee avec succes.")

if __name__ == "__main__":
    ingest_graph()
```

C Logique de l'Orchestrateur

Extrait du cœur logique de l'agent, montrant la gestion de la mémoire et la stratégie de repli (Fallback) entre le Graphe et le Vecteur.

Listing C.1 – Algorithme de décision (orchestrator.py - Extrait)

```
async def process_user_message(text: str, session_id: str):

    # 1. Gestion de la Memoire (Redis)
    memory = ConversationMemory(session_id)
    history = memory.get_history()

    # Reformulation contextuelle via Gemini
    standalone_text = text
    if history:
        standalone_text = contextualizer.rewrite(history, text)

    # 2. Strategie GraphRAG (Prioritaire)
    print("Tentative 1 : GraphRAG (Neo4j)...")
    graph_answer = graph_engine.query(standalone_text)

    if graph_answer and "je ne sais pas" not in graph_answer.lower():
        final_response = graph_answer

    # 3. Strategie VectorRAG (Fallback)
    else:
        print("GraphRAG muet. Tentative 2 : VectorRAG...")
        rag_results = rag_engine.search(standalone_text)

        if rag_results and rag_results[0]['score'] > 0.25:
            final_response = rag_results[0]['content']
        else:
            final_response = "Je ne trouve pas l'information."

    # 4. Sauvegarde
```

```
        memory.add_message("user", text)
        memory.add_message("ai", final_response)

        return final_response
```

D Documentation API (Endpoints)

Liste des points d'entrée exposés par le Backend FastAPI pour le Frontend Next.js.

- `GET /health` : Vérification de l'état des services (Healthcheck).
- `WS /ws/chat` : Canal WebSocket pour la communication bidirectionnelle temps réel (Texte).
- `GET /v1/admin/logs` : Récupération de l'historique des conversations pour le Dashboard d'administration (Monitoring).

Bibliographie

[BHW21] J. Barrasa, A.E. Hodler et J. Webber. **Building Knowledge Graphs : A Practitioner's Guide**. Disponible sur https://go.neo4j.com/rs/710-RRC-335/images/Building-Knowledge-Graphs-Practitioner's-Guide-OReilly-book.pdf. O'Reilly Media, 2021 (cf. p. 3).

[CWD18] Y. Chen, L. Wang et M. Dong. **Financial fraud detection using graph-based analysis**. *Journal of Financial Crime* (2018) (cf. p. 1, 3, 9).

[GAF23] GAFI (Groupe d'Action Financière). *Money Laundering through the Physical Transportation of Cash*. Rapport officiel sur les typologies de blanchiment. FATF/OECD. 2023. url : https://www.fatf-gafi.org/ (cf. p. 1).

[NH19] Mark Needham et Amy E. Hodler. **Graph Algorithms : Practical Examples in Apache Spark and Neo4j**. O'Reilly Media, 2019 (cf. p. 5).

[RWE15] Ian Robinson, Jim Webber et Emil Eifrem. **Graph Databases : New Opportunities for Connected Data**. 2nd. O'Reilly Media, 2015. isbn : 978-1491930892 (cf. p. 3, 4).

[Str24] Streamlit Inc. *Streamlit Documentation*. Framework Python pour les applications Data. 2024. url : https://docs.streamlit.io/ (cf. p. 6).