

M1 CNS PARCOURS SYSTÈMES AUTONOMIQUES  
TRAVAIL D'ÉTUDE ET DE RECHERCHE

---

# Alignment des LLMs avec les graphes de connaissances pour la génération d'explications adaptées

---



*Présenté par :*

MOKHTARI DHIA ELHAK  
LAKHMI BILLAL HICHEM  
OUHENIA ANIS

*Encadrant :*

HAMIDI MASSINISSA

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Objectif du Travail d'étude et de recherche . . . . .	3
<b>2</b>	<b>État de l'art</b>	<b>3</b>
2.1	Les grands modèles de langage (LLMs) dans l'éducation . . . . .	3
2.2	Représentations mentales en sciences cognitives et éducatives . . . . .	4
2.3	Graphes de connaissances . . . . .	4
2.4	Techniques d'alignement des LLMs . . . . .	5
<b>3</b>	<b>Méthodologie</b>	<b>6</b>
3.1	Construction du graphe de connaissances . . . . .	6
3.2	Modélisation des connaissances de l'apprenant . . . . .	7
3.3	Alignement du LLM pour la génération d'explications adaptées . . . . .	8
<b>4</b>	<b>Implémentation</b>	<b>8</b>
4.1	Technologies et outils . . . . .	8
4.2	Architecture du système . . . . .	9
4.3	Fonctionnalités implémentées . . . . .	10
<b>5</b>	<b>Expérimentations et résultats</b>	<b>11</b>
5.1	Protocole expérimental simulé . . . . .	11
5.2	Résultats observés et comparatifs . . . . .	11
5.3	Analyse qualitative . . . . .	12
<b>6</b>	<b>Discussion et limitations</b>	<b>13</b>
6.1	Points forts . . . . .	13
6.2	Limitations identifiées . . . . .	13
<b>7</b>	<b>Conclusion et perspectives</b>	<b>14</b>
7.1	Synthèse des contributions . . . . .	14
7.2	Perspectives futures . . . . .	14

# 1 Introduction

Les grands modèles de langage (LLMs\*) révolutionnent actuellement de nombreux domaines, dont celui de l'éducation. Leur capacité à générer du contenu cohérent et pertinent ouvre de nouvelles perspectives pour la création de ressources pédagogiques personnalisées. Cependant, malgré leurs performances impressionnantes, ces modèles présentent encore des lacunes significatives dans leur capacité à s'adapter aux différences individuelles des apprenants. L'un des défis majeurs dans l'utilisation des LLMs pour l'éducation réside dans leur incapacité native à prendre en compte la représentation mentale unique de chaque apprenant. Cette représentation, forgée par les expériences personnelles et les connaissances préalablement acquises, conditionne fortement la façon dont un nouvel apprentissage sera assimilé. De plus, les LLMs ne sont pas intrinsèquement conçus pour identifier et s'appuyer sur les prérequis spécifiques à chaque apprenant, ce qui limite considérablement la pertinence des contenus générés.

## 1.1 Objectif du Travail d'étude et de recherche

Ce travail vise à explorer et développer une approche permettant d'aligner les LLMs avec les représentations mentales des apprenants et leurs prérequis, grâce à l'utilisation de graphes de connaissances\*. L'objectif est de créer un système capable de générer des explications pédagogiques adaptées au niveau de compréhension individuel de chaque apprenant, améliorant ainsi l'efficacité du processus d'apprentissage.

# 2 État de l'art

## 2.1 Les grands modèles de langage (LLMs) dans l'éducation

Les LLMs comme GPT-4, Claude ou Mistral ont démontré des capacités impressionnantes dans la génération de texte cohérent et contextuel. Dans le domaine éducatif, ces modèles sont déjà utilisés pour diverses applications :

- Génération automatisée de questions et quiz
- Création de contenus pédagogiques adaptés à différents niveaux
- Production d'explications et d'exemples pour illustrer des concepts complexes
- Assistance à la rédaction et correction de travaux d'étudiants

Cependant, comme le soulignent [1] et [2], ces applications présentent encore des limitations importantes quant à la personnalisation des contenus. Les LLMs génèrent généralement des réponses basées sur des patterns statistiques sans véritable compréhension du niveau de connaissance spécifique de l'apprenant.



FIGURE 1 – Liste des LLMS les plus utilisés en 2024 (Source : Tech Radar)

## 2.2 Représentations mentales en sciences cognitives et éducatives

Les sciences cognitives ont largement étudié la façon dont les connaissances sont organisées dans l'esprit humain. Plusieurs théories et modèles sont particulièrement pertinents pour notre étude :

- **Les cartes conceptuelles\*** : Développées par Novak et Gowin [3], elles représentent graphiquement les relations entre les concepts et aident à visualiser la structure de connaissance d'un individu.
- **La théorie des schémas\*** : Proposée initialement par Bartlett [4] puis développée par d'autres chercheurs, elle suggère que les connaissances sont organisées en structures cognitives appelées schémas, qui facilitent la compréhension et l'assimilation de nouvelles informations.
- **Les modèles mentaux\*** : Johnson-Laird [5] décrit comment les individus construisent des représentations internes des phénomènes qu'ils observent, influençant leur raisonnement et leur apprentissage.

Ces théories mettent en évidence l'importance d'identifier et d'exploiter les structures de connaissances préexistantes chez l'apprenant pour faciliter l'acquisition de nouvelles connaissances.

## 2.3 Graphes de connaissances

Les graphes de connaissances constituent une approche prometteuse pour modéliser formellement les structures de connaissances. Ils permettent de représenter les concepts et leurs relations sous forme de graphes orientés [6]. Dans le contexte éducatif, plusieurs travaux ont exploré l'utilisation des graphes de connaissances :

- **Knowledge Forest\*** [7] : Utilise des graphes pour représenter les structures de connaissances et guider la génération de contenu pédagogique.

- **Intelligent Tutoring Systems\*** : Des systèmes comme AutoTutor [8] intègrent des représentations de connaissances pour adapter leurs interactions aux besoins spécifiques des apprenants.
- **Curriculum Learning\*** : L’organisation de l’apprentissage selon une progression logique basée sur les prérequis conceptuels [9].

Ces approches démontrent le potentiel des graphes de connaissances pour structurer et personnaliser l’apprentissage, mais leur intégration avec les capacités des LLMs reste un domaine relativement inexploré.

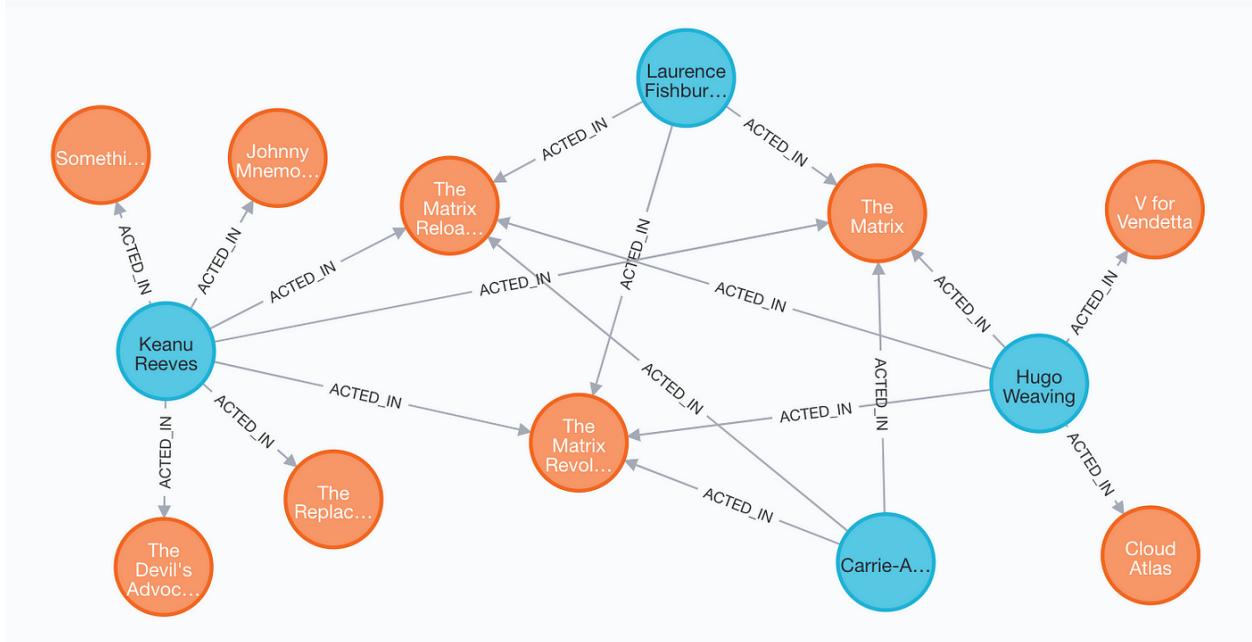


FIGURE 2 – Exemple d’un graphe de connaissances

## 2.4 Techniques d’alignement des LLMs

L’alignement des LLMs vise à faire correspondre leur comportement avec des objectifs ou contraintes spécifiques. Plusieurs techniques ont été développées récemment :

- **Prompt Engineering\*** : Manipulation des instructions fournies au modèle pour orienter sa génération [10].
- **Fine-tuning\*** : Ajustement des paramètres du modèle sur des données spécifiques au domaine ou à la tâche [11].
- **RLHF\*** (Reinforcement Learning from Human Feedback) : Utilisation de feedback humain pour affiner le comportement du modèle [12].
- **Constitutional AI\*** : Approche visant à aligner les modèles sur des principes ou contraintes spécifiques [13].

Dans le contexte éducatif, des travaux comme ceux de [14] ont commencé à explorer comment aligner les LLMs avec des objectifs pédagogiques, mais l’intégration avec des représentations individuelles des connaissances reste un défi majeur.

## 3 Méthodologie

Notre approche méthodologique s'articule autour de trois axes principaux : la construction d'un graphe de connaissances représentant les concepts du domaine et leurs relations, l'évaluation des connaissances de l'apprenant, et l'alignement du LLM pour générer des explications adaptées.

### 3.1 Construction du graphe de connaissances

Le graphe de connaissances constitue la structure fondamentale représentant le domaine d'apprentissage et les relations entre les concepts. Sa construction repose sur une approche semi-automatique combinant extraction automatique et validation humaine. Notre méthodologie s'appuie sur l'algorithme ACE\* (AI-Assisted Construction of Educational Knowledge Graphs with Prerequisite Relations) [?], qui permet de générer des graphes orientés, pondérés et acycliques à partir de contenus pédagogiques textuels. Les étapes suivies sont les suivantes :

- **Extraction des concepts et des paires candidates** : Identification des concepts fondamentaux par analyse de curricula, manuels et ressources pédagogiques, suivie de la génération automatique de paires de concepts via ACE.
- **Évaluation sémantique des relations** : Chaque paire de concepts est évaluée selon deux scores :
  - **CSR\*** (Concept Semantic Relatedness) : Mesure la proximité sémantique des concepts dans le texte.
  - **CER\*** (Concept Embedding Relatedness) : Évalue leur proximité vectorielle dans un espace d'embedding\*.
- **Filtrage et pondération des relations** : Les paires obtenant les scores les plus élevés sont sélectionnées comme relations de type *PREREQUISITE\**, avec un poids compris entre 0 et 1 reflétant l'intensité du lien.
- **Enrichissement par LLM** : Utilisation d'un LLM pour suggérer des concepts ou relations supplémentaires, validés manuellement.
- **Validation et raffinement** : Vérification de la cohérence globale du graphe et de la pertinence des relations par des experts du domaine.

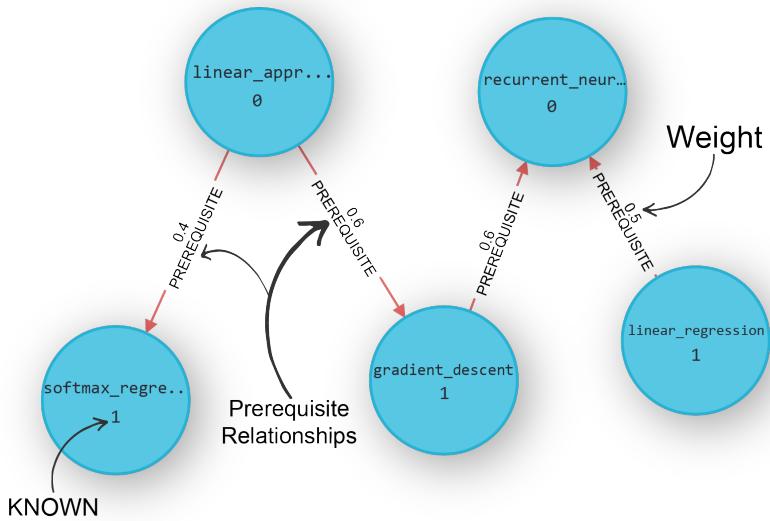


FIGURE 3 – Architecture du graphe de connaissances

### 3.2 Modélisation des connaissances de l'apprenant

Pour adapter les explications au niveau de chaque apprenant, nous devons évaluer ses connaissances actuelles, en tenant compte de la force des relations entre les concepts dans le graphe de connaissances ainsi que de l'état de maîtrise de chaque concept. Chaque concept dans le graphe est marqué avec un flag **KNOWN\***, qui peut avoir les valeurs suivantes :

- 0 : Le concept est inconnu.
- 1 : Le concept est maîtrisé.

En parallèle, les relations de prérequis entre les concepts sont associées à des poids compris entre 0 et 1, représentant la force de la dépendance entre les concepts. Ces poids permettent au système de prioriser les prérequis les plus importants dans le processus d'explication. Notre méthodologie comprend les étapes suivantes :

- **Évaluation diagnostique initiale** : Le système interroge le graphe de connaissances pour obtenir la liste des concepts inconnus (**KNOWN** = 0). Chaque concept est présenté à l'apprenant pour évaluer sa maîtrise. Si un concept est jugé maîtrisé (**KNOWN** = 1), il est mis à jour dans le graphe.
- **Annotation du graphe** : Le flag **KNOWN** est utilisé pour marquer le niveau de maîtrise de chaque concept (soit 0 pour inconnu, soit 1 pour maîtrisé). Lorsqu'un concept est maîtrisé, les concepts liés par des prérequis avec des poids élevés sont considérés comme plus urgents à aborder dans l'explication, tandis que ceux avec un poids faible peuvent être traités plus tard ou de manière optionnelle.
- **Mise à jour dynamique** : L'état de connaissance de l'apprenant est mis à jour à chaque interaction. Par exemple, un concept avec un flag **KNOWN** égal à 0 sera lié à des concepts ayant des prérequis avec des poids différents. Les prérequis avec un poids plus élevé seront prioritaires dans les explications, et le flag **KNOWN** des concepts sera

ajusté en fonction des performances de l'apprenant.

- **Inférence de connaissances** : Le système utilise des règles d'inférence pour estimer la maîtrise des concepts non directement évalués. L'inférence prend en compte la valeur **KNOWN** des concepts liés et le poids des relations pour estimer si un concept non évalué est effectivement connu. Un concept avec un poids de relation élevé pourra être traité plus tôt, même s'il n'a pas encore été directement évalué.

### 3.3 Alignement du LLM pour la génération d'explications adaptées

L'objectif central est d'adapter les explications générées par le LLM au niveau de connaissance spécifique de l'apprenant. Notre approche d'alignement comprend :

- **Construction de prompts contextualisés** : Intégration des informations du graphe de connaissances et du profil de l'apprenant dans les instructions fournies au LLM.
- **Génération multi-étapes** : Utilisation d'une approche séquentielle où le LLM planifie d'abord l'explication en fonction des prérequis identifiés, puis génère le contenu correspondant.
- **Post-traitement basé sur les règles** : Application de règles de vérification pour s'assurer que l'explication n'utilise pas de concepts inconnus sans les expliquer.
- **Fine-tuning ciblé** : Adaptation du modèle sur un corpus d'explications annotées selon différents niveaux de connaissances préalables.

## 4 Implémentation

Notre implémentation repose sur une architecture modulaire intégrant plusieurs composants techniques. Cette section détaille les technologies utilisées et les choix d'implémentation.

### 4.1 Technologies et outils

Pour la réalisation de notre système, nous avons sélectionné les technologies suivantes :

- **Base de données de graphe Neo4j\*** : Pour le stockage et la manipulation efficace du graphe de connaissances, permettant des requêtes complexes sur les relations entre concepts.
- **LLM local (Mistral)** : Déployé via Ollama\* pour permettre une génération de contenu flexible et contrôlable, tout en limitant les coûts et les problèmes de confidentialité.
- **Framework Python** : Utilisation de bibliothèques comme LangChain\* pour faciliter l'interaction avec le LLM et la construction de chaînes de traitement.
- **Interface utilisateur Chainlit\*** : Création d'une interface conversationnelle permettant à l'apprenant d'interagir naturellement avec le système.
- **Intégration PyPDF2 et OCR** : Pour l'extraction de contenu à partir de documents PDF, permettant d'analyser les ressources pédagogiques et d'enrichir le graphe de connaissances.



FIGURE 4 – Technologies utilisées

## 4.2 Architecture du système

Le système se compose de plusieurs modules qui gèrent les connaissances de l'apprenant, les prérequis et leur poids, tout en utilisant les flags `KNOWN` pour ajuster les interactions :

- **Module de gestion du graphe de connaissances** : Responsable de la création et mise à jour du graphe stocké dans Neo4j, en marquant chaque concept avec un flag `KNOWN` et en associant des prérequis avec des poids compris entre 0 et 1.
- **Module d'évaluation des connaissances** : Ce module génère des questions pour évaluer la maîtrise des concepts. En fonction des réponses de l'apprenant, le flag `KNOWN` est mis à jour et les prérequis avec des poids élevés sont priorisés dans les explications suivantes.
- **Module d'analyse de documents** : Après le téléchargement des PDF par l'apprenant, le système extrait les concepts mentionnés et les compare avec l'état du graphe de connaissances. Les flags `KNOWN` et les poids des prérequis guident la génération des explications.
- **Module d'alignement du LLM** : Le LLM utilise le profil actualisé de l'apprenant, incluant les flags `KNOWN` et les relations de prérequis pondérées, pour générer des explications contextualisées et adaptées au niveau de l'utilisateur.
- **Interface conversationnelle** : Permet à l'apprenant d'interagir avec le système. Elle présente des explications personnalisées et recueille des retours pour ajuster le flag `KNOWN` de chaque concept en fonction des interactions et des performances.

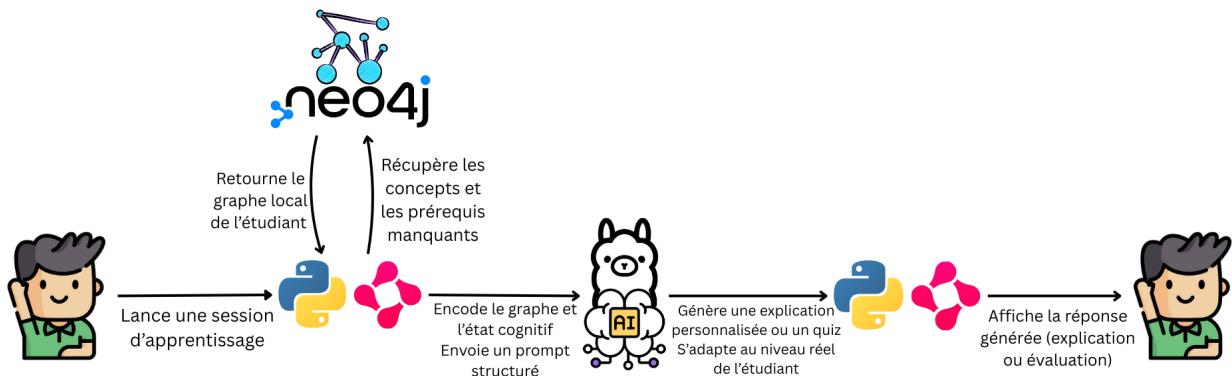


FIGURE 5 – Schéma global du système

### 4.3 Fonctionnalités implémentées

Notre système intègre plusieurs fonctionnalités concrètes permettant une adaptation dynamique des contenus générés à l'état cognitif de chaque apprenant :

- **Évaluation interactive des prérequis** : Le système interroge Neo4j pour identifier les concepts non encore maîtrisés ( $\text{KNOWN} = 0$ ). L'apprenant est invité à répondre à des questions de type "connaissez-vous ce concept?", permettant de mettre à jour son graphe personnel.
- **Génération d'explications conditionnelles** : Lorsqu'un concept est inconnu, le système récupère ses prérequis pondérés dans le graphe et demande au LLM, via un prompt, d'expliquer d'abord ces prérequis avant de traiter le concept cible. Cela garantit une progression adaptée au profil de l'utilisateur.
- **Encodage textuel du graphe dans le prompt** : Les relations entre concepts extraits de Neo4j sont encodées sous forme de texte puis intégrées dans le prompt envoyé au LLM. Cette approche, inspirée de l'article *Talk Like a Graph*\*, améliore la qualité du raisonnement du modèle.
- **Few-shot prompting\*** simplifié : Des exemples d'explications ou de quiz sont insérés dans certains prompts pour guider le LLM, améliorant ainsi la structuration et la clarté des contenus générés.
- **Génération automatique de quiz** : Après chaque explication, un quiz Vrai/Faux est automatiquement généré par le LLM au format JSON. La validation de ce quiz permet de mettre à jour le statut du concept ( $\text{KNOWN} = 1$ ) dans le graphe.
- **Analyse de documents PDF** : L'apprenant peut charger un fichier PDF. Le système en extrait le contenu (texte ou via OCR), l'indexe avec Chroma, puis utilise un retriever pour poser des questions sur le contenu ou détecter les concepts abordés.
- **Mise à jour du graphe basée sur les interactions** : Chaque réponse correcte à un quiz ou confirmation d'un concept connu déclenche une mise à jour dans Neo4j, modifiant dynamiquement le graphe de connaissances de l'apprenant.

## 5 Expérimentations et résultats

Bien que notre système soit encore en phase de développement, nous avons pu formuler des observations et comparer théoriquement les bénéfices attendus par rapport à un LLM standard. Cette section présente notre protocole expérimental envisagé, ainsi qu'une analyse comparative basée sur le fonctionnement interne du système et des cas simulés.

### 5.1 Protocole expérimental simulé

Pour évaluer notre approche, nous avons conçu un protocole expérimental simulé pouvant être réalisé avec un groupe d'apprenants dans un futur proche. Il s'articule autour des étapes suivantes :

- **Constitution du graphe de connaissances** : Construction d'un graphe orienté et pondéré dans un domaine ciblé (par exemple : machine learning), comprenant 35 concepts et plus de 200 relations de prérequis.
- **Modélisation des apprenants** : Chaque étudiant est associé à une vue personnalisée du graphe où chaque concept possède un attribut **KNOWN**, mis à jour dynamiquement après un quiz ou une évaluation interactive.
- **Comparaison des approches** :
  - **LLM standard** : Génération d'explications à partir d'un texte ou d'une question, sans tenir compte du niveau réel de l'étudiant.
  - **Notre système aligné** : Utilisation du graphe de prérequis pour guider dynamiquement le LLM, en intégrant uniquement les concepts maîtrisés et les prérequis nécessaires dans le prompt.
- **Éléments évalués** :
  - Cohérence des explications par rapport au niveau réel de l'étudiant.
  - Progressivité dans l'introduction des concepts.
  - Réduction des erreurs liées à des prérequis manquants.

### 5.2 Résultats observés et comparatifs

Les différences de comportement entre un LLM standard et notre approche orientée graphe ont été observées sur plusieurs aspects clés :

Critère	LLM standard	Notre système (KG + LLM)
Prise en compte du niveau étudiant	Aucune	Oui (via KNOWN et pré-requis)
Structuration des explications	Variable, parfois hors de portée	Guidée par les pré-requis
Cohérence des chaînes logiques	Parfois incohérente	Contrôlée via le graphe orienté
Progressivité	Aléatoire	Ajustée dynamiquement
Personnalisation	Générique	Personnalisée par graphe individuel

TABLE 1 – Comparatif entre un LLM standard et notre système aligné

De plus, notre système est capable :

- D'identifier les prérequis manquants d'un concept demandé.
- De générer un quiz ciblé sur ces lacunes.
- D'expliquer d'abord les prérequis avant de traiter le concept cible.

Ces capacités ne sont pas possibles avec un LLM utilisé seul.

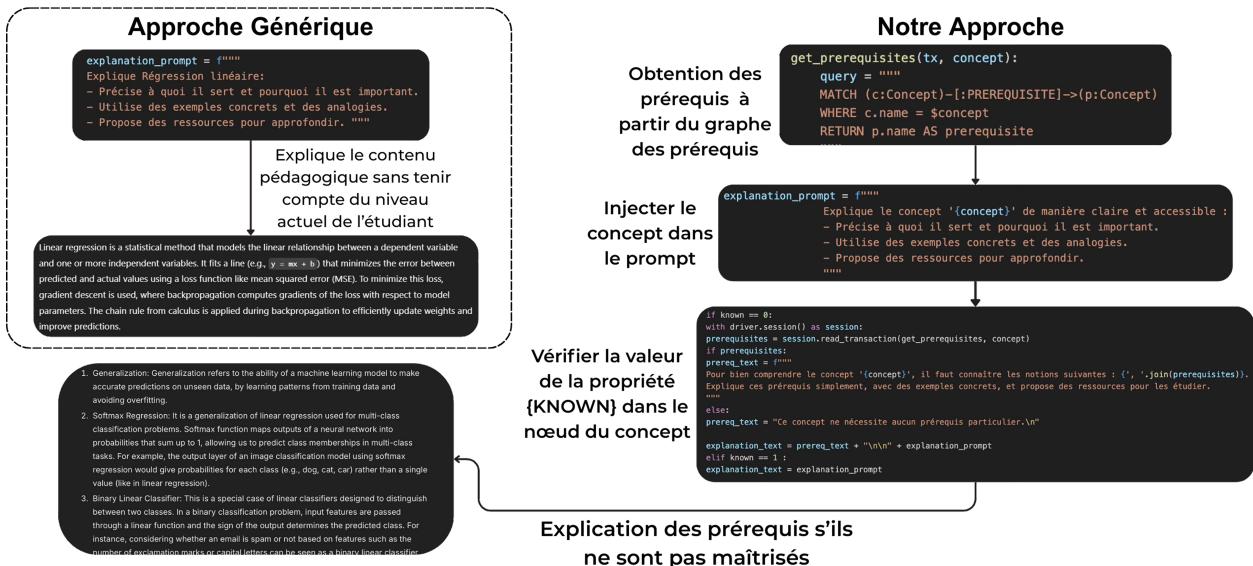


FIGURE 6 – Approche d'un LLM Standard VS Notre Approche

### 5.3 Analyse qualitative

Les retours d'utilisation sur des cas simulés et interactifs ont permis d'identifier plusieurs bénéfices notables :

- **Meilleure clarté des explications** : L'introduction préalable des prérequis réduit les confusions et facilite la compréhension des concepts complexes.

- **Sentiment de progression logique** : Les apprenants perçoivent une montée en compétence structurée et cohérente.
- **Pertinence des contenus générés** : Le LLM, guidé par le graphe, évite d'aborder des notions non maîtrisées par l'étudiant, ce qui limite les hallucinations et les erreurs de raisonnement.

Toutefois, plusieurs limites ont également été relevées :

- **Qualité des prompts sensible à l'encodage du graphe** : Une mauvaise structuration textuelle des relations peut dégrader la pertinence des explications.
- **Présence de redondances** : Si le modèle est trop fortement guidé, des répétitions peuvent apparaître dans les explications successives.
- **Calibration du score de confiance (KNOWN) perfectible** : Les seuils actuels nécessitent des ajustements pour mieux refléter la maîtrise réelle de l'apprenant.

## 6 Discussion et limitations

L'analyse des résultats obtenus nous permet d'identifier les forces et les limites de notre approche, ainsi que des pistes d'amélioration pour les travaux futurs.

### 6.1 Points forts

Notre système présente plusieurs avantages notables :

- **Personnalisation effective** : L'alignement du LLM avec le graphe de connaissances permet une adaptation précise des explications au profil individuel de l'apprenant.
- **Évolutivité du modèle apprenant** : La mise à jour dynamique du profil de connaissances permet au système de s'adapter à la progression de l'apprenant.
- **Transparence du processus** : L'utilisation explicite du graphe de connaissances rend le processus d'adaptation plus compréhensible et contrôlable que les approches boîtes noires.
- **Indépendance relative du domaine** : Bien que notre prototype ait été testé dans un domaine spécifique, l'architecture proposée est transposable à d'autres domaines d'apprentissage.

### 6.2 Limitations identifiées

Malgré ces avantages, notre approche présente plusieurs limitations qu'il convient de reconnaître :

- **Coût de construction du graphe** : La création initiale d'un graphe de connaissances de qualité reste un processus exigeant en temps et en expertise.
- **Précision de l'évaluation des connaissances** : L'évaluation automatique des connaissances de l'apprenant reste imparfaite, avec des risques de faux positifs et de faux négatifs.
- **Dépendance aux capacités du LLM** : La qualité des explications générées dépend fondamentalement des capacités du modèle de langage sous-jacent.
- **Gestion des styles d'apprentissage** : Notre approche actuelle se concentre principalement sur les connaissances préalables, sans prendre en compte d'autres facteurs

comme les styles d'apprentissage préférentiels.

- **Passage à l'échelle** : L'application de notre approche à des domaines très vastes ou interdisciplinaires pose des défis en termes de complétude et de cohérence du graphe de connaissances.

## 7 Conclusion et perspectives

### 7.1 Synthèse des contributions

Ce travail a exploré l'alignement des grands modèles de langage (LLMs) avec les représentations mentales des apprenants grâce à l'utilisation de graphes de connaissances. Nos principales contributions sont :

- Une méthodologie pour la construction et l'exploitation de graphes de connaissances représentant à la fois le domaine d'apprentissage et le profil de l'apprenant.
- Une approche d'alignement des LLMs permettant de générer des explications adaptées au niveau de connaissance spécifique de chaque apprenant.
- Un prototype fonctionnel démontrant la faisabilité et l'efficacité de cette approche dans un contexte éducatif.
- Des résultats expérimentaux suggérant une amélioration significative de l'efficacité de l'apprentissage grâce à la personnalisation des explications.

Ces contributions ouvrent la voie à des systèmes d'apprentissage plus intelligents et personnalisés, capables de s'adapter aux besoins spécifiques de chaque apprenant.

### 7.2 Perspectives futures

Plusieurs directions prometteuses pourraient être explorées pour étendre et améliorer ce travail :

- **Automatisation de la construction du graphe** : Développement de méthodes plus automatisées pour extraire des graphes de connaissances à partir de ressources pédagogiques existantes.
- **Intégration de dimensions affectives** : Prise en compte non seulement des connaissances, mais aussi des facteurs motivationnels et émotionnels dans l'adaptation des explications.
- **Approches multimodales** : Extension du système pour générer des explications combinant texte, visualisations et autres modalités adaptées aux préférences de l'apprenant.
- **Évaluation à long terme** : Études longitudinales pour évaluer l'impact de notre approche sur l'apprentissage à long terme et le transfert de connaissances.
- **Co-construction du graphe** : Développement d'approches permettant aux apprenants de contribuer à l'enrichissement du graphe de connaissances, reflétant leur propre compréhension du domaine.

En conclusion, l'alignement des LLMs avec les graphes de connaissances représente une approche prometteuse pour personnaliser l'apprentissage à l'ère de l'intelligence artificielle. Notre travail constitue une première étape dans cette direction, avec de nombreuses opportunités d'amélioration et d'extension dans des travaux futurs.

## Références

- [1] Zhao, W., Eger, S., Gurevych, I. (2023). *A Survey on Large Language Models for Education*. arXiv preprint arXiv :2312.10530.
- [2] Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günemann, S., Hüllermeier, E., et al. (2023). *ChatGPT for good ? On opportunities and challenges of large language models for education*. Learning and Individual Differences, 103, 102274.
- [3] Novak, J. D., & Gowin, D. B. (1984). *Learning how to learn*. Cambridge University Press.
- [4] Bartlett, F. C. (1932). *Remembering : A study in experimental and social psychology*. Cambridge University Press.
- [5] Johnson-Laird, P. N. (1983). *Mental models : Towards a cognitive science of language, inference, and consciousness*. Harvard University Press.
- [6] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., et al. (2021). *Knowledge graphs*. ACM Computing Surveys, 54(4), 1-37.
- [7] Ye, P., Han, X., Lin, B., Liu, Q., Tong, Z., Wei, F. (2023). *Leveraging Knowledge Graphs for Zero-Shot LLM-Based Learning*. Proceedings of the AAAI Conference on Artificial Intelligence, 37(8), 11124-11132.
- [8] Graesser, A. C., Chipman, P., Haynes, B. C., & Olney, A. (2005). *AutoTutor : An intelligent tutoring system with mixed-initiative dialogue*. IEEE Transactions on Education, 48(4), 612-618.
- [9] Yu, W., Luo, M., Zhou, P., Si, C., Zhou, Y., Wang, X., Wu, J., Chen, X. (2020). *Curriculum Learning : A Survey*. Machine Learning, 1-40.
- [10] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D. (2022). *Chain of thought prompting elicits reasoning in large language models*. Advances in Neural Information Processing Systems, 35, 24824-24837.
- [11] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). *Training language models to follow instructions with human feedback*. Advances in Neural Information Processing Systems, 35, 27730-27744.
- [12] Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). *Deep reinforcement learning from human preferences*. Advances in Neural Information Processing Systems, 30.
- [13] Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. (2022). *Constitutional AI : Harmlessness from AI Feedback*. arXiv preprint arXiv :2212.08073.
- [14] Kumar, A., Kalwani, N., Li, J., Gupta, S., West, P. (2023). *Supporting Education with Large Language Models : Opportunities and Alignment Challenges*. arXiv preprint arXiv :2308.05591.

- [15] Chen, X., Liu, Q., Li, S., Du, Y., Yu, W., Wu, J. (2023). *Adaptive Curriculum Learning with Large Language Models for Educational Data Mining*. arXiv preprint arXiv :2310.04560.
- [16] Sun, H., Zhang, Y., Wang, S., Tan, M. (2025). *Educational Knowledge Graph Enhanced Large Language Models*. arXiv preprint arXiv :2503.13340.
- [17] Sharath, S., Pardos, Z., Chuang, I. (2024). *Embedding Knowledge Graphs in Educational Data Mining*. Journal of Educational Data Mining (JEDM), 16(1).
- [18] Lee, S., Ryu, J., Choi, J., Hwang, S. (2022). *Prompt Engineering for Educational Applications of Large Language Models*. arXiv preprint arXiv :2210.12228.

# Glossaire

**LLM\*** *Large Language Model* - Modèle de langage de grande taille, système d'IA entraîné sur d'immenses corpus de textes capables de générer du contenu linguistique cohérent et contextuel.

**Graphe de connaissances\*** Structure de données qui représente les connaissances sous forme de graphe orienté où les nœuds sont des concepts et les arêtes des relations sémantiques entre ces concepts.

**Prompt Engineering\*** Technique consistant à formuler et structurer précisément les instructions données à un LLM pour optimiser ses réponses selon des objectifs spécifiques.

**Fine-tuning\*** Processus d'ajustement des paramètres d'un modèle préentraîné sur un ensemble de données spécifiques à une tâche pour améliorer ses performances sur cette tâche.

**RLHF\*** *Reinforcement Learning from Human Feedback* - Apprentissage par renforcement à partir de feedback humain, méthode pour affiner le comportement d'un modèle en utilisant des évaluations humaines.

**Constitutional AI\*** Approche visant à encadrer le comportement des modèles d'IA par un ensemble de principes prédéfinis, similaires à une constitution.

**Cartes conceptuelles\*** Représentations graphiques organisées montrant les relations entre différents concepts, utilisées comme outils pédagogiques.

**Théorie des schémas\*** Théorie cognitive suggérant que la connaissance est organisée en unités structurées (schémas) qui facilitent la compréhension et l'assimilation de nouvelles informations.

**Modèles mentaux\*** Représentations internes simplifiées que les individus construisent pour comprendre et interagir avec le monde extérieur.

**Curriculum Learning\*** Stratégie d'apprentissage qui consiste à organiser l'acquisition des connaissances selon une progression de difficulté croissante.

**ACE\*** *AI-Assisted Construction of Educational Knowledge Graphs with Prerequisite Relations* - Algorithme d'assistance à la construction de graphes de connaissances éducatifs avec relations de prérequis.

**CSR\*** *Concept Semantic Relatedness* - Mesure quantifiant la proximité sémantique entre deux concepts dans un texte.

**CER\*** *Concept Embedding Relatedness* - Évaluation de la proximité de deux concepts dans un espace vectoriel d'embeddings.

**PREREQUISITE\*** Relation dans un graphe de connaissances indiquant qu'un concept doit être maîtrisé avant d'aborder un autre concept.

**KNOWN\*** Attribut binaire dans un graphe de connaissances indiquant si un concept est maîtrisé (1) ou non (0) par un apprenant.

**Few-shot prompting\*** Technique consistant à inclure quelques exemples dans les instructions données à un LLM pour guider sa génération.

**Intelligent Tutoring Systems\*** Systèmes informatiques conçus pour fournir un enseignement personnalisé sans intervention humaine directe.

**Knowledge Forest\*** Approche utilisant des graphes pour représenter et exploiter des structures de connaissances dans un contexte éducatif.

**Embedding\*** Représentation vectorielle d'un mot, phrase ou concept dans un espace multidimensionnel, capturant ses caractéristiques sémantiques.

**Chain of thought\*** Technique de prompting encourageant un LLM à décomposer son raisonnement en étapes intermédiaires explicites.

**Neo4j\*** Système de gestion de base de données orientée graphe permettant de stocker et interroger efficacement des données structurées en graphe.

**LangChain\*** Framework Python facilitant le développement d'applications basées sur des LLMs en permettant de créer des chaînes de traitement.

**Chainlit\*** Bibliothèque Python pour créer rapidement des interfaces conversationnelles basées sur des LLMs.

**Ollama\*** Outil permettant d'exécuter localement des modèles de langage open-source.

**Talk Like a Graph\*** Approche consistant à encoder textuellement les informations d'un graphe pour les intégrer dans un prompt de LLM.

## Annexes

## Diagramme de séquence

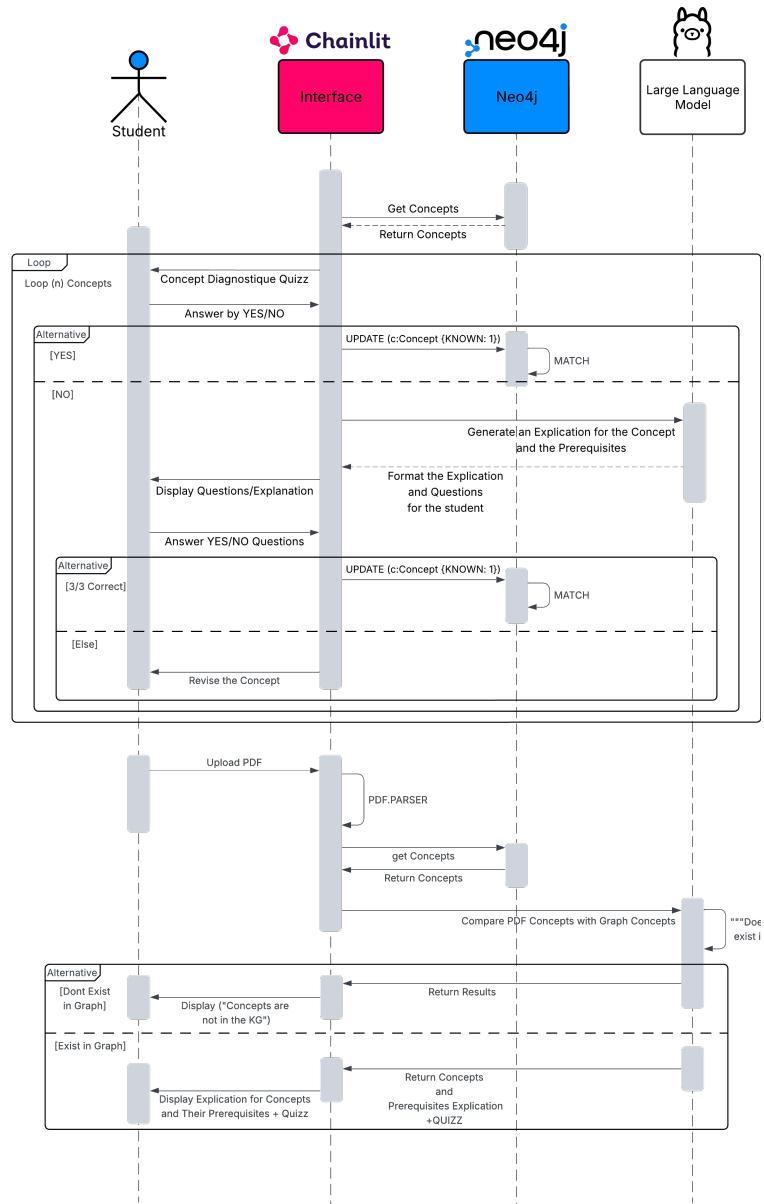


FIGURE 7 – Diagramme de séquence de notre système

## Code Python du système

```
1 import random
2 from neo4j import GraphDatabase
3 from langchain_community.llms import Ollama
4 import ollama
5 import PyPDF2
6 import chainlit as cl
7 from langchain_ollama import OllamaEmbeddings
8 from langchain.text_splitter import RecursiveCharacterTextSplitter
9 from langchain_community.vectorstores import Chroma
10 from langchain.chains import ConversationalRetrievalChain
11 from langchain_community.chat_message_histories import ChatMessageHistory
12 from langchain.memory import ConversationBufferMemory
13
14 from dotenv import load_dotenv
15 import os
16 import random
17 import pytesseract
18 from pdf2image import convert_from_path
19 from PIL import Image
20 import json
21
22 # Connexion à Neo4j
23 uri = "bolt://localhost:7687"
24 username = "neo4j"
25 password = "12345678"
26
27 try:
28     driver = GraphDatabase.driver(uri, auth=(username, password))
29     with driver.session() as session:
30
31         result = session.run("RETURN 1 AS test")
32         for record in result:
33             print("Connexion réussie:", record["test"])
34     except Exception as e:
35         print("Erreur de connexion à Neo4j:", e)
36
37 # Initialisation du modèle LLM
38 load_dotenv()
39
40 from langchain_ollama import OllamaLLM
41 llm = OllamaLLM(model="mistral")
42
43 def extract_text_from_pdf(file_path):
44     """
45     Extrait le texte du fichier PDF. Si le PDF contient uniquement des images,
→ l'OCR est utilisé pour extraire le texte des images.
```

```

46     Si le PDF contient à la fois du texte et des images, le texte est extrait et
47     ↪ l'OCR est appliqué aux pages sans texte.
48
49     :param file_path: Chemin vers le fichier PDF.
50     :return: Le texte extrait du PDF.
51     """
52
53     pdf = PyPDF2.PdfReader(file_path)
54     pdf_text = ""
55     images = convert_from_path(file_path)
56     ocr_text = ""
57
58     for page_number, page in enumerate(pdf.pages):
59         # Tenter d'extraire le texte
60         text = page.extract_text()
61         if text:
62             pdf_text += text
63         else:
64             # Si aucun texte n'est extrait, appliquer l'OCR sur l'image de cette
65             ↪ page
66             print(f"Pas de texte trouvé sur la page {page_number + 1}, tentative
67             ↪ d'OCR.")
68             img = images[page_number]
69             ocr_text += pytesseract.image_to_string(img)
70
71     full_text = pdf_text + "\n" + ocr_text
72     # Si le texte extrait est vide, retournez une indication
73     if not full_text.strip():
74         return None # Indique que le PDF ne contient pas de texte
75     return full_text
76
77
78 @cl.action_callback("generate_examples")
79 async def generate_examples(action):
80     chain = cl.user_session.get("chain")
81     random_seed = random.randint(1, 100)
82     text = cl.user_session.get("full_pdf_text")
83     prompt = (f"Using seed {random_seed}, generate exactly 5 creative examples that
84     ↪ illustrate key concepts from the text.\n\n{text}")
85     res = await chain.acall({"question": prompt},
86     ↪ callbacks=[cl.AsyncLangchainCallbackHandler()])
87     await cl.Message(content=res["answer"]).send()
88
89 @cl.action_callback("generate_quiz")
90 async def generate_quiz(action):
91     chain = cl.user_session.get("chain")
92     random_seed = random.randint(1, 100)
93     text = cl.user_session.get("full_pdf_text")

```

```

88     prompt = (f"Using seed {random_seed}, generate exactly 5 quiz questions. Each
89     ↵  should have one correct answer, "
90         f"three incorrect alternatives, and an explanation for the correct
91         ↵  choice.\n\n{text}")
92     res = await chain.acall({"question": prompt},
93     ↵  callbacks=[cl.AsyncLangchainCallbackHandler()])
94     await cl.Message(content=res["answer"]).send()
95
96 @cl.action_callback("generate_questions")
97 async def generate_questions(action):
98     chain = cl.user_session.get("chain")
99     themes = ["Factual Details", "Interpretative Insights", "Critical Evaluations"]
100    random_theme = random.choice(themes)
101    random_seed = random.randint(1, 100)
102    text = cl.user_session.get("full_pdf_text")
103    prompt = (f"Using seed {random_seed}, generate 5 unique questions focusing on
104    ↵  '{random_theme}'.\n\n{text}")
105    res = await chain.acall({"question": prompt},
106    ↵  callbacks=[cl.AsyncLangchainCallbackHandler()])
107    await cl.Message(content=res["answer"]).send()
108
109 @cl.action_callback("generate_explanation")
110 async def generate_explanation(action):
111     chain = cl.user_session.get("chain")
112     text = cl.user_session.get("full_pdf_text")
113     prompt = ("Provide a detailed and pedagogical explanation of a key concept from
114     ↵  the text.\n\n" + text)
115     res = await chain.acall({"question": prompt},
116     ↵  callbacks=[cl.AsyncLangchainCallbackHandler()])
117     answer = res["answer"]
118     sources = "\n\nSources:\n" + "\n".join([f"- {doc.metadata['source']}" for doc
119     ↵  in res["source_documents"]])
120     await cl.Message(content=answer + sources).send()
121
122 @cl.on_message
123 async def main(message: cl.Message):
124     chain = cl.user_session.get("chain")
125     cb = cl.AsyncLangchainCallbackHandler()
126
127     res = await chain.acall({"question": message.content}, callbacks=[cb])
128     answer = res["answer"]
129     sources = res["source_documents"]
130
131     sources_content = "\n\nSources:\n" + "\n".join([f"- {doc.metadata['source']}"
132     ↵  for doc in sources]) if sources else ""
133     await cl.Message(content=answer + sources_content).send()

```

```

126
127 # -----
128 # Fonctions base de données
129 #
130 def get_random_concepts(tx):
131     query = """
132         MATCH (c:Concept)
133         WHERE c.name IS NOT NULL AND c.known = 0
134         RETURN c.name AS concept
135         ORDER BY rand()
136         LIMIT 5
137     """
138
139     result = tx.run(query)
140     return [record["concept"] for record in result]
141
142 def get_prerequisites(tx, concept):
143     query = """
144         MATCH (c:Concept)-[:PREREQUISITE]->(p:Concept)
145         WHERE c.name = $concept
146         RETURN p.name AS prerequisite
147     """
148
149     result = tx.run(query, concept=concept)
150     return [record["prerequisite"] for record in result]
151
152 def get_all_concepts(tx):
153     query = """
154         MATCH (c:Concept)
155         WHERE c.name IS NOT NULL
156         RETURN c.name AS concept
157         ORDER BY rand()
158     """
159
160     result = tx.run(query)
161     return [record["concept"] for record in result]
162
163 # -----
164 # Chat start
165 # -----
166 @cl.on_chat_start
167
168 async def on_chat_start():
169     with driver.session() as session:
170         concepts = session.read_transaction(get_random_concepts)
171
172         if not concepts:
173             await cl.Message("Aucun concept trouvé dans la base de données.").send()
174             await handle_pdf_upload()
175             return

```

```

173     cl.user_session.set("concepts", concepts)
174     cl.user_session.set("current_index", 0)
175
176     await ask_concept_question()
177
178
179
180     # -----
181     # Étape 1 : Demande initiale
182     # -----
183
184     async def ask_concept_question():
185         concepts = cl.user_session.get("concepts")
186         index = cl.user_session.get("current_index")
187
188         if index >= len(concepts):
189             await cl.Message(" Tu as terminé tous les concepts ! Place maintenant à
190             ↳ l'analyse du PDF.").send()
191             await handle_pdf_upload()
192             return
193
194         concept = concepts[index]
195         await cl.Message(
196             content=f"Connais-tu le concept suivant : **{concept}** ? (Oui/Non)",
197             actions=[
198                 cl.Action(name="yes", label="Oui", payload={"known": True}),
199                 cl.Action(name="no", label="Non", payload={"known": False})
200             ]
201         ).send()
202
203     # -----
204     # Si connu
205     # -----
206     @cl.action_callback("yes")
207     async def handle_known_concept(action):
208         # Récupérer le concept actuel
209         concepts = cl.user_session.get("concepts")
210         index = cl.user_session.get("current_index")
211         concept = concepts[index]
212
213         # Mise à jour de la propriété 'known' à 1 dans Neo4j
214         try:
215             with driver.session() as session:
216                 session.run("""
217                     MATCH (c:Concept)
218                     WHERE c.name = $concept
219                     SET c.known = 1

```

```

219     """", concept=concept)
220     await cl.Message("Parfait ! Passons au suivant.").send()
221 except Exception as e:
222     await cl.Message(f" Erreur lors de la mise à jour du concept : {e}").send()
223
224     # Passer au concept suivant
225     cl.user_session.set("current_index", cl.user_session.get("current_index") + 1)
226
227     await ask_concept_question()
228
229     # -----
230     # Si inconnu → Explication + Quiz
231     #
232     @cl.action_callback("no")
233     async def handle_unknown_concept(action):
234         concepts = cl.user_session.get("concepts")
235         index = cl.user_session.get("current_index")
236         concept = concepts[index]
237
238         with driver.session() as session:
239             prerequisites = session.read_transaction(get_prerequisites, concept)
240
241         prereq_text = f"""
242                         Pour bien comprendre le concept '{concept}', il est
243                         ↳ important de se concentrer uniquement sur les prérequis
244                         ↳ les plus pertinents et directement liés à ce concept.
245                         ↳ Parmi les notions suivantes : {',
246                         ↳ '.join(prerequisites)}', identifie uniquement celles qui
247                         ↳ sont étroitement liées à '{concept}'.
248                         Explique uniquement ces prérequis de manière simple, avec
249                         ↳ des exemples concrets, et propose des ressources utiles
250                         ↳ pour les étudier."""
251
252         concept_text = f"""
253             Explique ensuite le concept '{concept}' de manière claire et accessible :
254             - Décris son but, son utilité et dans quel contexte il est important.
255             - Utilise des exemples concrets et des analogies pour faciliter la
256                 ↳ compréhension.
257             - Suggère quelques ressources (articles, tutoriels, vidéos, etc.) pour aller
258                 ↳ plus loin et mieux maîtriser ce concept.
259             """
260
261
262
263     quiz_prompt = (
264         f"Génère 3 questions Vrai/Faux avec les bonnes réponses, au format JSON
265         ↳ comme ceci :\n"
266         f"[{{'question': '...', 'answer': 'Vrai'}}], ...] sur le concept
267         ↳ '{concept}'."

```

```

256     )
257
258     try:
259         explanation = await cl.make_async(llm.invoke)(prereq_text + "\n\n" +
260             concept_text)
261         quiz_json = await cl.make_async(llm.invoke)(quiz_prompt)
262
263         # Vérification du format du JSON
264         try:
265             quiz_data = json.loads(quiz_json.strip()) # Utilisation de
266             # json.loads() pour éviter les problèmes de sécurité
267             if not isinstance(quiz_data, list) or not all(isinstance(q, dict) and
268                 'question' in q and 'answer' in q for q in quiz_data):
269                 raise ValueError("Le format du quiz généré est incorrect.")
270             except (json.JSONDecodeError, ValueError) as e:
271                 await cl.Message(f" Erreur dans le format du quiz généré :
272                     {str(e)}").send()
273                 cl.user_session.set("current_index",
274                     cl.user_session.get("current_index") + 1)
275                 await ask_concept_question()
276
277             return
278
279         except Exception as e:
280             await cl.Message(f" Erreur pendant la génération : {e}").send()
281             cl.user_session.set("current_index", cl.user_session.get("current_index") +
282                 1)
283             await ask_concept_question()
284             return
285
286         # Afficher explication
287         await cl.Message(f" **Explication de
288             {concept}**\n\n{explanation.strip()}").send()
289
290         # Sauvegarde du quiz et gestion de la session
291         cl.user_session.set("current_quiz", quiz_data)
292         cl.user_session.set("quiz_index", 0)
293         cl.user_session.set("quiz_score", 0)
294
295         await send_quiz_question()
296
297         # -----
298         # Envoyer une question du quiz
299         # -----
300         async def send_quiz_question():
301             quiz = cl.user_session.get("current_quiz")
302             index = cl.user_session.get("quiz_index")

```

```

296     if index >= len(quiz):
297         score = cl.user_session.get("quiz_score")
298         total = len(quiz)
299
300         result_msg = f" Tu as bien compris ! Score : {score}/{total}" if score ==
301             ↪ total else f" Tu as eu {score}/{total}. Continue de réviser !"
302         await cl.Message(result_msg).send()
303
304         # Si toutes les questions sont répondues correctement, mettre à jour le
305             ↪ concept comme "connu"
306         if score == total:
307             concepts = cl.user_session.get("concepts")
308             current_index = cl.user_session.get("current_index")
309             if current_index < len(concepts):
310                 concept = concepts[current_index]
311                 try:
312                     with driver.session() as session:
313                         session.run("""
314                             MATCH (c:Concept)
315                             WHERE c.name = $concept
316                             SET c.known = 1
317                             """, concept=concept)
318                         await cl.Message(f" Le concept '{concept}' est maintenant
319                             ↪ marqué comme connu.").send()
320             except Exception as e:
321                 await cl.Message(f" Erreur lors de la mise à jour du concept :
322                     ↪ {e}").send()
323
324         # Passer au concept suivant
325         cl.user_session.set("current_index", cl.user_session.get("current_index") +
326             ↪ 1)
327
328         # Vérifier si tous les concepts ont été vus
329         concepts = cl.user_session.get("concepts")
330         current_index = cl.user_session.get("current_index")
331
332         if current_index >= len(concepts):
333             await cl.Message(" Tu as terminé tous les concepts ! Place maintenant à
334                 ↪ l'analyse du PDF.").send()
335             await handle_pdf_upload() # Lance l'analyse du PDF
336         else:
337             await ask_concept_question()
338
339         return
340
341         question = quiz[index]["question"]
342         await cl.Message(
343             content=f" **Question {index+1} :** {question}",

```

```

337     actions=[  
338         cl.Action(name="true", label="Vrai", payload={"response": "Vrai"}),  
339         cl.Action(name="false", label="Faux", payload={"response": "Faux"})  
340     ]  
341 ).send()  
342  
343 # -----  
344 # Réponse au quiz  
345 # -----  
346 @cl.action_callback("true")  
347 async def answer_true(action):  
348     await handle_quiz_response("Vrai")  
349  
350 @cl.action_callback("false")  
351 async def answer_false(action):  
352     await handle_quiz_response("Faux")  
353  
354 async def handle_quiz_response(user_answer):  
355     quiz = cl.user_session.get("current_quiz")  
356     index = cl.user_session.get("quiz_index")  
357     correct = quiz[index]["answer"]  
358  
359     if user_answer.lower() == correct.lower():  
360         await cl.Message(" Bonne réponse !").send()  
361         cl.user_session.set("quiz_score", cl.user_session.get("quiz_score") + 1)  
362     else:  
363         await cl.Message(f" Mauvaise réponse. La bonne réponse était  
364             **{correct}**.").send()  
365  
366     cl.user_session.set("quiz_index", index + 1)  
367     await send_quiz_question()  
368  
369 # -----  
370 # Lancer l'analyse du PDF après la fin des quiz  
371 # -----  
372 async def handle_pdf_upload():  
373     files = None  
374  
375     # Demande d'upload de fichier  
376     while files is None:  
377         files = await cl.AskFileMessage(  
378             content=" Veuillez uploader un fichier PDF pour commencer !",  
379             accept=["application/pdf"],  
380             max_size_mb=100,  
381             timeout=180  
382         ).send()

```

```

383     file = files[0]
384     pdf_text = extract_text_from_pdf(file.path)
385
386     if not pdf_text:
387         await cl.Message(content=" Le PDF téléchargé ne contient pas de texte
388         ↳ exploitable. Veuillez essayer avec un autre fichier.").send()
389     return
390
391     # Découpage du texte
392     text_splitter = RecursiveCharacterTextSplitter(chunk_size=1200,
393         ↳ chunk_overlap=50)
394     texts = text_splitter.split_text(pdf_text)
395     metadatas = [{"source": f"i-{pl}" for i in range(len(texts))}]
396
397     # Embedding & indexation
398     embeddings = OllamaEmbeddings(model="nomic-embed-text")
399     docsearch = await cl.make_async(Chroma.from_texts)(texts, embeddings,
400         ↳ metadatas=metadatas)
401
402     # Mémoire de conversation
403     memory = ConversationBufferMemory(
404         memory_key="chat_history",
405         output_key="answer",
406         chat_memory=ChatMessageHistory(),
407         return_messages=True,
408     )
409
410     # Création de la chaîne de QA
411     chain = ConversationalRetrievalChain.from_llm(
412         llm=llm,
413         chain_type="stuff",
414         retriever=docsearch.as_retriever(),
415         memory=memory,
416         return_source_documents=True
417     )
418     cl.user_session.set("chain", chain)
419
420     # Message d'attente pour l'analyse du PDF
421     wait_message = await cl.Message(" Analyse du PDF en cours...").send()
422
423     # Résumé du contenu
424     summary_prompt = f"""
425     Fais un résumé structuré et concis du texte suivant, en mettant en valeur les
426     ↳ concepts clés et idées principales :
427     {pdf_text}
428     """
429
430     summary = await cl.make_async(llm.invoke)(summary_prompt)

```

```

426
427     # Récupération de tous les concepts
428     with driver.session() as session:
429         concepts = session.read_transaction(get_all_concepts)
430         cl.user_session.set("concepts", concepts)
431
432     matched_concepts = set()
433
434     # Détection des concepts mentionnés dans le résumé
435     for concept in concepts:
436         prompt = f"""
437             Analyse ce résumé et détermine s'il mentionne ou traite du concept
438             ↳ \'{concept}\'.
439             Réponds uniquement par 'Oui' ou 'Non'.
440
441             Résumé :
442             {summary}
443             """
444
445         try:
446             response = await cl.make_async(llm.invoke)(prompt)
447             if "oui" in response.strip().lower():
448                 matched_concepts.add(concept)
449
450                 with driver.session() as session:
451                     result = session.run("MATCH (c:Concept) WHERE c.name = $concept
452                     ↳ RETURN c.known AS known", concept=concept)
453                     known = result.single()["known"]
454
455                     explanation_prompt = f"""
456                         Explique le concept '{concept}' de manière claire et accessible :
457                         - Précise à quoi il sert et pourquoi il est important.
458                         - Utilise des exemples concrets et des analogies.
459                         - Propose des ressources pour approfondir.
460                         """
461
462
463                     if known == 0:
464                         with driver.session() as session:
465                             prerequisites = session.read_transaction(get_prerequisites,
466                             ↳ concept)
467
468                         if prerequisites:
469                             prereq_text = f"""
470                                 Pour bien comprendre le concept '{concept}', il est
471                                 ↳ important de se concentrer uniquement sur les prérequis
472                                 ↳ les plus pertinents et directement liés à ce concept.
473                                 ↳ Parmi les notions suivantes : {',
474                                 ↳ '.join(prerequisites)}}, identifie uniquement celles qui
475                                 ↳ sont étroitement liées à '{concept}'.

```

```

466             Explique uniquement ces prérequis de manière simple, avec
467             ↳ des exemples concrets, et propose des ressources utiles
468             ↳ pour les étudier."""
469
470     else:
471         prereq_text = "Ce concept ne nécessite aucun prérequis
472             ↳ particulier.\n"
473
474     explanation_text = prereq_text + "\n\n" + explanation_prompt
475 else:
476     explanation_text = explanation_prompt
477
478     explanation = await cl.make_async(llm.invoke)(explanation_text)
479     await cl.Message(f" **Explication de
480             ↳ {concept}**\n\n{explanation.strip()}").send()
481
482 except Exception as e:
483     print(f" Erreur lors de la vérification du concept '{concept}' : {e}")
484
485     await wait_message.remove()
486
487 if matched_concepts:
488     cl.user_session.set("matched_concepts", matched_concepts)
489
490     # Génération du quiz
491     quiz_prompt = f"""
492     Génère 5 questions de quiz à choix multiples (QCM), chacune portant sur un
493         ↳ des concepts suivants : {', '.join(matched_concepts)}.
494     Pour chaque question :
495         - Propose une bonne réponse et trois distracteurs plausibles.
496         - N'indique PAS la bonne réponse.
497         - Utilise un format clair comme :
498             **Question 1 :** Quel est le rôle de XYZ ?
499             A. Réponse plausible
500             B. Réponse plausible
501             C. Réponse plausible
502             D. Réponse plausible
503             Adapte la langue du quiz à celle des concepts si besoin.
504             """
505
506     quiz_output = await cl.make_async(llm.invoke)(quiz_prompt)
507     await cl.Message(f" **Quiz basé sur les concepts détectés
508             ↳ :**\n\n{quiz_output.strip()}").send()
509 else:
510     await cl.Message(" Aucun concept détecté dans ce document.").send()
511
512     cl.user_session.set("full_pdf_text", pdf_text)

```

```
507     await cl.Message(" Le PDF a été traité. Vous pouvez maintenant poser vos  
508     ↵   questions !").send()  
509  
510  
511  
512  
513  
514  
515
```