ROYAUME DU MAROC
HAUT COMMISSARIAT AU PLAN
INSTITUT NATIONALDE STATISTIQUE
ET D'ECONOMIE APPLIQUEE

INSEA

Rapport de Projet de Fin d'Année

Sous le thème de :

Automatisation de l'affectation des professeurs des classes préparatoires selon une grille de besoins.

Organisme d'accueil : TEMA CONCEPT

Réalisé par : **DYN Mohamed**

Encadré par : **M. Mrabet Abdelhaie**

Soutenue devant le jury composé par :

- M. **KADRANI Abdeslam**
- M. **BENMANSOUR Rachid**

Filière : DATA SCIENCE

Année Universitaire : 2024/2025

Dédicace

À mes chers parents,

Pour leur amour inconditionnel, leurs sacrifices et leur soutien indéfectible, qui ont été ma plus grande source de force et de motivation.

À mes chères sœurs et à mes frères,

Pour leur affection, leurs encouragements et les moments de joie partagés.

À ma fiancée,

Pour sa patience, sa compréhension et sa présence rassurante tout au long de ce parcours.

À mes amis et à mes proches,

Pour leur amitié sincère, leurs conseils et leur bonne humeur qui m'ont accompagné dans les moments difficiles comme dans les instants de bonheur.

À tous ceux qui, de près ou de loin,

Ont joué un rôle dans ma vie et ont contribué à faire de moi la personne que je suis aujourd'hui, je vous dédie ce travail avec toute ma gratitude et mon respect.

Remerciements

Au terme de ce travail, il est agréable d'adresser quelques expressions de remerciements à toute personne dont l'intervention au cours de ce projet a favorisé son aboutissement. Nous souhaitons avant tout rendre grâce à Dieu, le Tout-Puissant et le Miséricordieux, pour nous avoir octroyé la force et l'endurance indispensables pour accomplir cet humble travail. La réalisation de ce travail n'aurait jamais pu se concrétiser sans l'aide de Dieu.

Je tiens à remercier chaleureusement **l'équipe de TEMACONCEPT** pour m'avoir accueilli au sein de leur startup et m'avoir offert l'opportunité d'effectuer mon stage PFA dans un cadre dynamique et innovant. Cette expérience m'a permis de découvrir le monde professionnel de manière concrète, d'enrichir mes compétences techniques et de mieux comprendre les enjeux liés à l'automatisation dans un environnement réel. Leur disponibilité, leur encadrement et leur confiance ont grandement contribué à la réussite de mon projet.

Je tiens également à exprimer toute ma reconnaissance à **Monsieur Abdelhaie Mrabet**, mon encadrant, pour son accompagnement précieux tout au long de ce projet, ses conseils avisés, sa disponibilité et son sens pédagogique m'ayant permis d'avancer avec méthode et confiance. Grâce à son suivi attentif et à ses encouragements constants, j'ai pu surmonter les difficultés rencontrées et enrichir mes connaissances, tant sur le plan technique que méthodologique.

Ne pouvant remercier individuellement toutes les personnes que j'ai côtoyées lors de mon stage au sein de **TEMACONCEPT**, mes remerciements vont à l'ensemble de la division des systèmes d'information ainsi qu'à leur contribution inestimable qui nous a permis de bénéficier de leur expertise, de leur vaste expérience et de leurs précieuses orientations professionnelles.

Table des matières

Liste des abréviations

LLM : Large Language Model

RAG : Retrieval-Augmented Generation

CSV : Comma-Separated Values

JSON : JavaScript Object Notation

NLP : Natural Language Processing

Liste des tableaux

Liste des figures

Introduction générale

L'affectation des professeurs dans les classes préparatoires constitue une étape stratégique pour garantir la qualité de l'enseignement. Cette opération, souvent réalisée manuellement, repose sur une grille de besoins définissant les postes à pourvoir et doit prendre en compte de multiples facteurs comme les compétences des enseignants, leurs préférences et diverses contraintes. Lorsque le nombre de professeurs et de classes augmente, cette tâche manuelle devient particulièrement complexe et chronophage , pouvant entraîner des erreurs, des retards et un manque d'optimisation.

La problématique centrale de ce projet est donc de déterminer comment concevoir un système capable d'automatiser cette affectation en respectant à la fois la grille de besoins et l'ensemble des contraintes pédagogiques, administratives et humaines. L'objectif principal est de concevoir et mettre en place une solution informatique visant à remplacer les méthodes manuelles par un processus rapide, fiable et optimisé.

Pour atteindre cet objectif, ce stage effectué au sein de TEMACONCEPT a porté sur le développement d'une solution algorithmique. Le système s'appuie sur deux sources de données principales : la grille de besoins des établissements et la liste détaillée des professeurs. Ces informations sont utilisées pour construire une matrice de compatibilité qui est ensuite traitée par l'algorithme hongrois pour garantir une affectation optimale. Le système gère également la complexité des cascades de mutations.

Enfin, un objectif innovant de ce projet est la conception d'un chatbot intelligent reposant sur un LLM. Cet outil permet aux utilisateurs, qu'ils soient administrateurs ou enseignants , de poser des questions en langage naturel sur les résultats d'affectation, offrant ainsi une interaction fluide et renforçant la transparence du processus.

Chapitre 1. Organisme d'accueil

1. Introduction

Dans ce chapitre, nous allons explorer TEMA CONCEPT, l'organisme où j'ai réalisé mon stage. Nous présenterons l'organisation générale et interne, ainsi que le lieu précis de mon stage. Un historique de TEMA CONCEPT sera fourni, accompagné de quelques chiffres clés illustrant la performance de l'entreprise. Enfin, nous conclurons ce chapitre pour offrir une vue d'ensemble du cadre professionnel dans lequel j'ai évolué.

2. Organisation générale

TEMA CONCEPT est une organisation structurée autour de plusieurs **départements stratégiques**, chacun disposant d'une expertise pointue dans un domaine spécifique de l'informatique, ce qui lui permet de couvrir l'ensemble du cycle de vie des projets technologiques. Cette structuration repose sur une logique de spécialisation et de complémentarité : un département est dédié à l'**infrastructure informatique**, chargé de concevoir, déployer et maintenir les environnements matériels et réseaux nécessaires au bon fonctionnement des systèmes, en garantissant performance, sécurité et haute disponibilité. Un autre département se concentre sur le **développement logiciel et applicatif**, où des équipes d'ingénieurs et d'analystes conçoivent des solutions sur mesure, adaptées aux besoins métiers des clients, en intégrant les meilleures pratiques de codage et les technologies les plus récentes.

Parallèlement, un pôle spécialisé dans la **cybersécurité** veille à la protection des données et des systèmes contre les menaces internes et externes, en mettant en place des politiques de sécurité, des audits réguliers et des solutions de surveillance proactive. Un autre département est dédié à l'**intégration des systèmes**, assurant la connexion fluide entre différentes applications, plateformes et bases de données, afin de garantir l'interopérabilité et la cohérence des flux d'information. Enfin, un service de **support et maintenance** assure un suivi opérationnel continu, offrant assistance technique, mises à jour et optimisation des performances.

Cette organisation en départements spécialisés permet à TEMA CONCEPT de proposer une **offre complète et cohérente**, allant de la conception de l'architecture technique à l'intégration et au suivi post-déploiement, tout en garantissant un haut niveau de qualité, de réactivité et d'innovation.

3. Organisation interne

Chaque département de TEMA CONCEPT travaille en étroite synergie avec les autres, dans une logique de **collaboration transversale** visant à offrir aux clients des solutions complètes, cohérentes et parfaitement intégrées. Cette coopération ne se limite pas à un simple échange d'informations : elle implique un **partage constant d'expertise**, une harmonisation des méthodes de travail et une coordination rigoureuse des interventions, depuis la phase de conception jusqu'au déploiement final. Les équipes spécialisées — qu'il s'agisse de l'infrastructure, du développement logiciel, de la cybersécurité, de l'intégration des systèmes ou du support technique — interviennent de manière complémentaire, chacune apportant sa valeur ajoutée pour garantir la qualité et la performance des livrables.

Au cœur de cette organisation, la gestion de projet joue un rôle central. Elle assure la planification des tâches, la répartition optimale des ressources, le suivi des jalons et la maîtrise des délais. Les chefs de projet agissent

comme des chefs d'orchestre, veillant à ce que chaque département avance au même rythme, que les dépendances entre tâches soient anticipées et que les éventuels obstacles soient levés rapidement. Cette approche structurée permet non seulement de minimiser les risques et d'éviter les redondances, mais aussi de garantir que la solution finale réponde pleinement aux besoins exprimés par le client, tout en respectant les contraintes techniques, budgétaires et réglementaires.

4. Présentation de lieu du stage

Mon stage s'est déroulé dans le bureau de Mr. Abdelhaie, au sein du petit local de TEMA CONCEPT.

Service développement	**Service infrastructure informatique**	**Service cybersécurité**	**Service administratif**
✓ 1 Architecte fonctionnel ✓ 1 Designer ✓ 2 ingénieurs conception/ Développement ✓ 1 Développeur full stock	✓ 2 Ingénieur Réseau ✓ 1 ingénieur DevOps ✓ 1 technicien spécialisé	✓ 2 ingénieurs ✓ 1 technicien assistant	✓ 1 assistance administrative et commerciale

Tableau 1: Organigramme du TEMACONCEPT

5. Historique de l’Organisme

Fondé il y a 18 ans, TEMA CONCEPT s'est progressivement imposé comme un acteur majeur dans le domaine des technologies de l'information, offrant une gamme variée de services à une clientèle diversifiée.

6. Quelques chiffres clés

- 18 ans d'existence
- Plus de 25 clients publics
- Plus de 15 clients privés
- 3 clients internationaux
- Une centaine de projets livrés
- Des dizaines de formations animées
- 100% taux de satisfaction client

7. Conclusion

Ce stage chez TEMA CONCEPT a été une expérience enrichissante, me permettant d'appliquer mes connaissances académiques à des projets concrets et de développer des compétences professionnelles précieuses dans le domaine de la data science.

Chapitre 2. Contexte et problématique du projet

1. Contexte du projet

Dans le système éducatif, et plus particulièrement dans les classes préparatoires, l'affectation des professeurs constitue une étape stratégique pour garantir la qualité de l'enseignement et le bon déroulement des programmes. Cette opération, souvent réalisée manuellement par les responsables administratifs, repose sur une grille de besoins définissant le nombre de postes à pourvoir par matière, niveau et filière. Elle doit également prendre en compte les compétences des enseignants, leurs disponibilités, leurs préférences, ainsi que des contraintes géographiques ou réglementaires. Or, lorsque le nombre de professeurs et de classes augmente, cette tâche devient particulièrement complexe et chronophage. Les méthodes traditionnelles, basées sur des tableaux et des échanges multiples, peuvent entraîner des erreurs, des retards et un manque d'optimisation dans la répartition des ressources humaines. Dans un contexte où la digitalisation des processus est devenue un levier majeur d'efficacité, l'automatisation de l'affectation des professeurs apparaît comme une solution moderne et performante, capable de réduire le temps de traitement, d'améliorer la précision des affectations et de garantir une meilleure équité dans la distribution des postes.

2. Problématique du projet

La question centrale à laquelle ce projet tente de répondre est la suivante : comment concevoir un système capable d'affecter automatiquement les professeurs aux classes préparatoires en respectant à la fois la grille de besoins et l'ensemble des contraintes pédagogiques, administratives et humaines ? Cette problématique soulève plusieurs défis. D'abord, il s'agit de modéliser correctement les données relatives aux enseignants (matières enseignées, niveaux maîtrisés, filières, disponibilités, préférences, ancienneté, rapprochement familial, etc.) et aux postes à pourvoir (matière, niveau, filière, localisation). Ensuite, il faut choisir un algorithme d'optimisation adapté, capable de traiter un grand volume de données tout en garantissant une affectation optimale et équitable. Enfin, le système doit être suffisamment flexible pour s'adapter aux changements fréquents, tels que les mutations, les absences imprévues ou la création de nouvelles classes. La difficulté réside donc dans la capacité à combiner rigueur mathématique, performance technique et prise en compte des réalités humaines du métier d'enseignant.

3. Objectifs du projet

L'objectif principal de ce projet est de concevoir et de mettre en place une solution informatique capable d'automatiser l'affectation des professeurs aux classes préparatoires, en se basant sur une grille de besoins prédéfinie par l'administration. Cette solution doit permettre de remplacer les méthodes manuelles, souvent longues et sujettes aux erreurs, par un processus rapide, fiable et optimisé. L'automatisation vise non seulement à réduire le temps nécessaire à la planification, mais aussi à améliorer la précision des affectations, en tenant compte des contraintes pédagogiques, des préférences des enseignants et des priorités institutionnelles. L'outil développé devra ainsi offrir une répartition équitable et transparente, tout en respectant les exigences réglementaires et organisationnelles.

Pour atteindre cet objectif global, le projet se fixe plusieurs objectifs spécifiques. Il convient de concevoir un modèle de données structuré, capable de représenter fidèlement les besoins des établissements (matières,

niveaux, filières, localisation) et les profils des enseignants (compétences, disponibilités, préférences, ancienneté, rapprochement familial). Sur cette base, un algorithme d'optimisation adapté tel que l'algorithme hongrois sera implémenté pour garantir une affectation optimale. L'outil devra également intégrer une interface conviviale permettant la saisie des données, le lancement des calculs et la visualisation claire des résultats.

En complément de ces fonctionnalités, un objectif innovant du projet est la conception d'un **chatbot intelligent** reposant sur un **LLM**. Ce chatbot sera intégré au système et permettra aux utilisateurs qu'ils soient administrateurs, enseignants ou responsables pédagogiques de poser des questions en langage naturel sur les résultats d'affectation. Il pourra, par exemple, expliquer pourquoi un professeur a été affecté à un poste précis, indiquer les critères pris en compte, fournir des statistiques globales ou détaillées, et répondre à toute interrogation liée au processus. Cette fonctionnalité offrira une interaction fluide et personnalisée avec le système, renforçant la transparence, la compréhension et l'accessibilité des informations produites par l'algorithme d'affectation.

Chapitre 3. Analyse et conception

I. Processus d'affectation

1. Introduction

L'affectation des professeurs aux classes préparatoires est une étape stratégique et déterminante dans l'organisation pédagogique d'un établissement. Elle consiste à attribuer chaque enseignant à un poste précis, défini par une **grille de besoins** élaborée par l'administration. Cette grille recense, pour chaque établissement, le nombre de postes à pourvoir par matière, niveau et filière, ainsi que leur localisation.

Traditionnellement, cette opération est réalisée manuellement par les responsables pédagogiques et administratifs. Elle repose sur la collecte et l'analyse d'un grand volume d'informations : compétences disciplinaires des enseignants, niveaux qu'ils peuvent assurer, filières dans lesquelles ils peuvent intervenir, préférences personnelles, contraintes géographiques, rapprochements familiaux, ancienneté, ainsi que les priorités fixées par l'institution. Ce travail, bien que maîtrisé par les équipes expérimentées, présente plusieurs limites : il est chronophage, sujet aux erreurs humaines et difficile à optimiser lorsque le nombre de professeurs et de postes augmente.

Dans le cadre de ce projet, l'automatisation du processus d'affectation repose sur **deux sources de données principales** :

1. **La grille de besoins** des établissements, qui définit les postes à pourvoir avec leurs caractéristiques (matière, niveau, filière, ville).

2. **La liste détaillée des professeurs**, comprenant leur identité, leurs choix d'affectation classés par ordre de préférence, leur score (calculé selon des critères objectifs tels que l'ancienneté, la performance ou le rapprochement familial), ainsi que des informations précises sur leurs compétences (matières enseignées, niveaux maîtrisés, filières possibles) et leur situation géographique.

Ces deux ensembles de données sont fusionnés pour construire une **matrice de compatibilité et de pondération**. Chaque case de cette matrice représente le score attribué à l'affectation d'un professeur donné à un poste donné, en tenant compte des critères de compatibilité stricte (matière, niveau, filière) et des bonus/malus liés aux préférences, à la localisation ou à d'autres contraintes.

L'optimisation de cette affectation est réalisée à l'aide de **l'algorithme hongrois** (ou Kuhn-Munkres), qui permet de trouver la combinaison optimale maximisant le score global tout en respectant les contraintes. Après l'affectation initiale, le système gère également les **cascades de mutations** : lorsqu'un professeur change d'établissement, son ancien poste devient vacant et peut être attribué à un autre enseignant, déclenchant ainsi une série d'affectations successives.

Enfin, les résultats sont présentés de manière claire et exploitable : liste des affectations avec détails des critères appliqués, liste des enseignants non affectés avec justification, liste des postes non pourvus, et indicateurs de performance. Cette approche algorithmique et structurée permet non seulement de gagner du temps et d'améliorer la précision des affectations, mais aussi de renforcer la transparence et la traçabilité des décisions, tout en offrant une meilleure satisfaction aux enseignants et aux responsables pédagogiques.

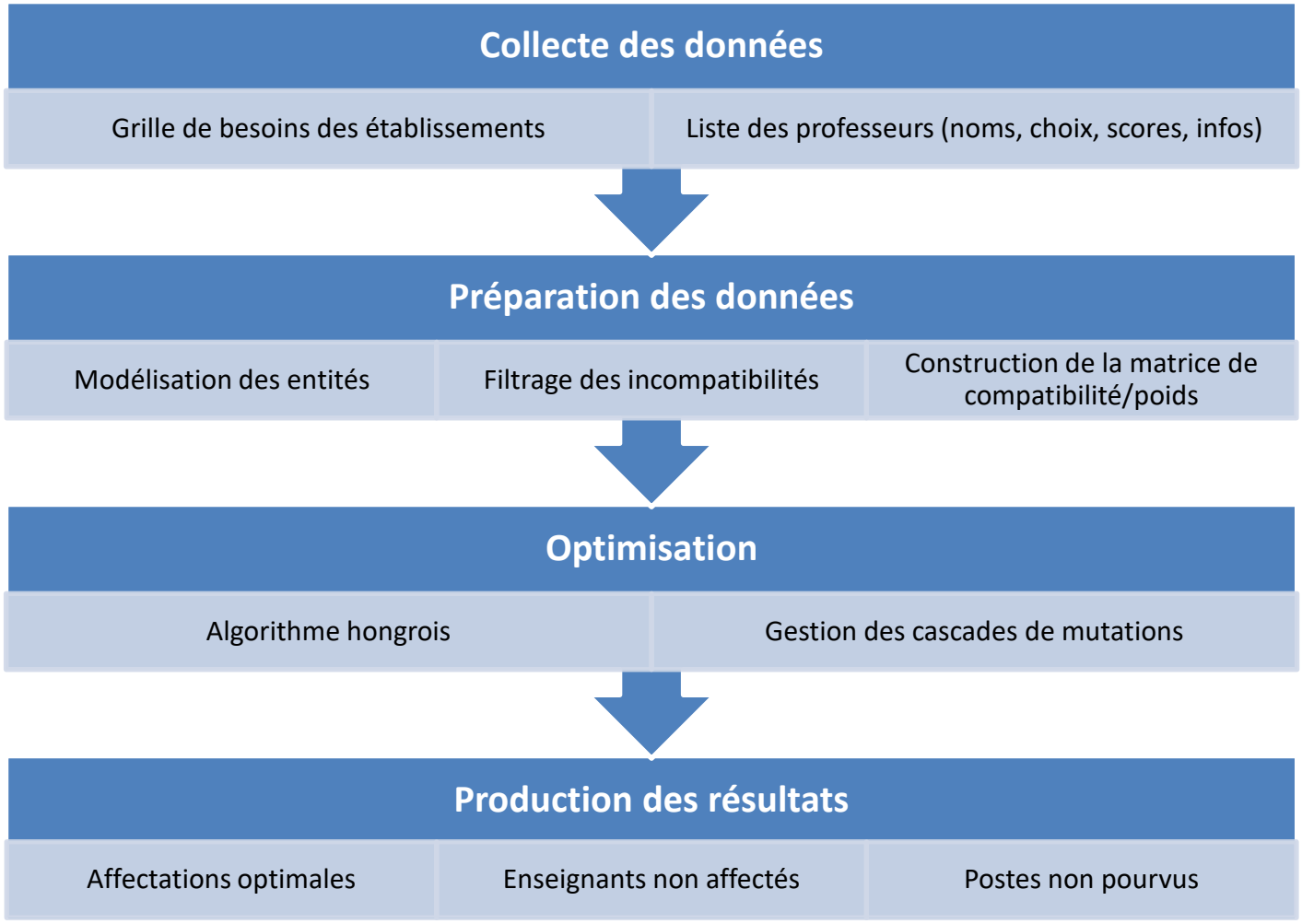


Figure 1:Schéma global du processus d'affectation automatisée.

2. Modélisation des données

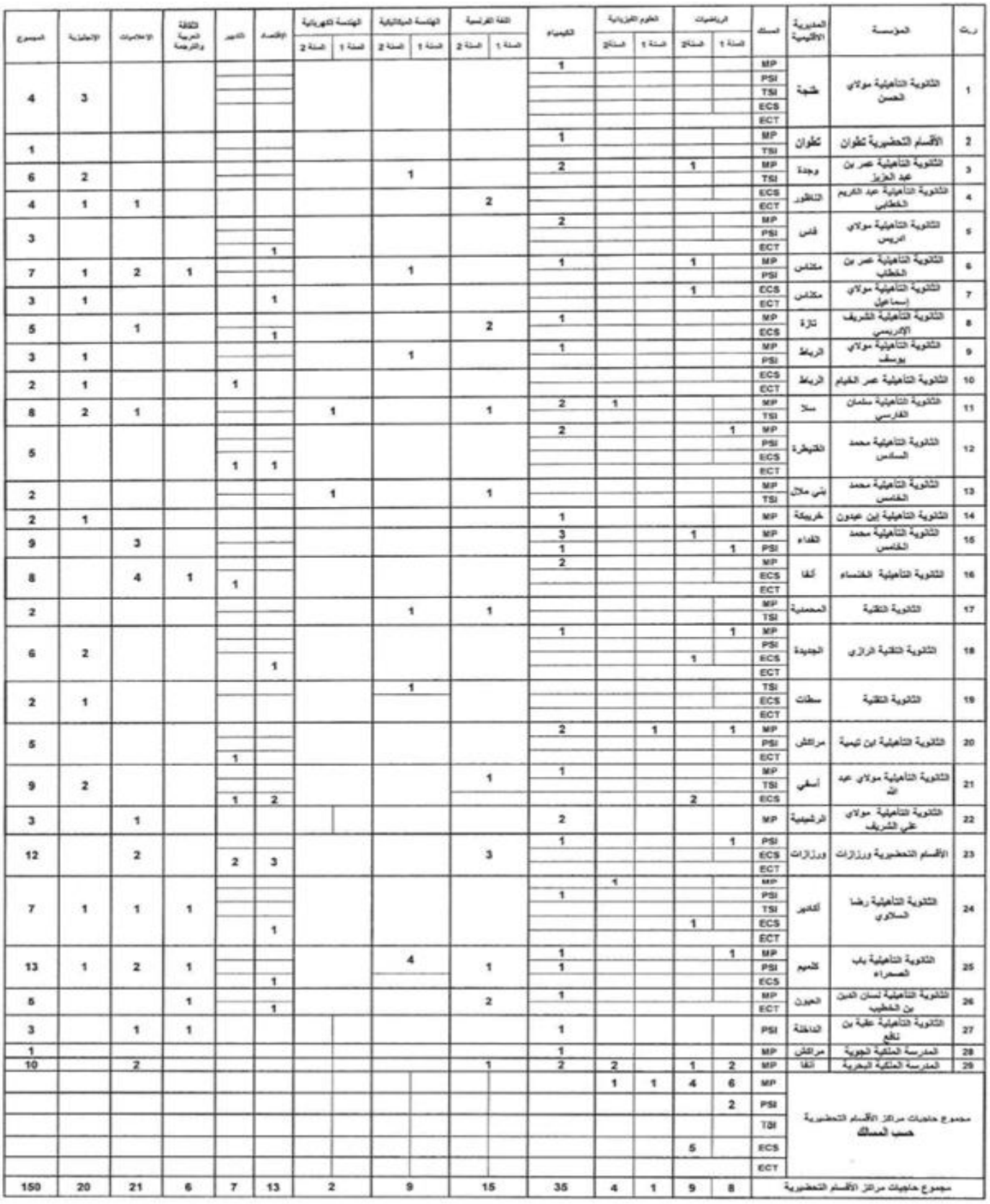
La modélisation des données constitue une étape fondamentale dans la conception d'un système d'information, en particulier lorsqu'il s'agit d'un projet complexe comme l'automatisation de l'affectation des professeurs aux classes préparatoires. Elle a pour objectif de représenter, de manière claire, structurée et non ambiguë, l'ensemble des informations nécessaires au fonctionnement du système, ainsi que les relations qui existent entre elles. Cette représentation sert de base à la fois à la conception de la base de données, à l'implémentation de l'algorithme d'optimisation et à l'intégration des modules complémentaires, comme le chatbot explicatif basé sur RAG.

Dans le contexte de ce projet, la modélisation des données revêt une importance particulière car elle doit intégrer **deux sources d'information distinctes mais complémentaires**. D'une part, **la grille de besoins** des établissements, qui recense pour chaque structure le nombre de postes à pourvoir, en précisant la matière, le niveau, la filière et la localisation. Cette grille constitue la vision institutionnelle et réglementaire des besoins

en ressources humaines. D'autre part, **la liste détaillée des professeurs**, qui contient non seulement leurs informations personnelles et professionnelles (nom, matière enseignée, niveaux maîtrisés, filières possibles, ville actuelle, établissement actuel), mais aussi leurs **choix d'affectation** classés par ordre de préférence, leur **score** (calculé selon des critères objectifs tels que l'ancienneté, la performance ou le rapprochement familial) et d'autres contraintes ou préférences spécifiques.

La modélisation doit permettre de **fusionner ces deux ensembles de données** afin de produire une **matrice de compatibilité et de pondération** qui servira de base à l'algorithme hongrois. Cette matrice est au cœur du processus d'optimisation : chaque cellule représente le score attribué à l'affectation d'un professeur donné à un poste donné, en tenant compte à la fois des contraintes de compatibilité stricte (matière, niveau, filière) et des bonus/malus liés aux préférences, à la localisation ou à d'autres critères définis par l'administration.

Un autre enjeu majeur de la modélisation est la **traçabilité**. Dans un système automatisé, il est essentiel de pouvoir expliquer et justifier chaque décision d'affectation. Cela implique de stocker non seulement le résultat final (professeur affecté à tel poste), mais aussi le détail des calculs ayant conduit à ce résultat : score total, contribution de chaque critère, règles appliquées et éventuelles pénalités. Ces informations sont indispensables pour l'audit interne, mais aussi pour alimenter le module de chatbot explicatif, qui doit être capable de répondre en langage naturel à des questions telles que : *« Pourquoi ce professeur a-t-il été affecté à cet établissement plutôt qu'à un autre ? »*.



ر.ت	المؤسسة	المديرية الإقليمية	المسلك	الرياضيات السنة 1	الرياضيات السنة 2	العلوم الفيزيائية السنة 1	العلوم الفيزيائية السنة 2	الكيمياء	اللغة الفرنسية السنة 1	اللغة الفرنسية السنة 2	الهندسة الميكانيكية السنة 1	الهندسة الميكانيكية السنة 2	الهندسة الكهربائية السنة 1	الهندسة الكهربائية السنة 2	الإقتصاد	التدبير	الثقافة العربية والشرعية	الإعلاميات	الإنجليزية	المجموع
1	الثانوية التأهيلية مولاي الحسن	عثمنة	MP					1											3	4
			PSI																	
			TSI																	
			ECS																	
			ECT																	
2	الأقسام التحضيرية تطوان	تطوان	MP					1												1
			TSI																	
3	الثانوية التأهيلية عمر بن عبد العزيز	وجدة	MP	1				2			1								2	6
			TSI																	
4	الثانوية التأهيلية عبد الكريم الخطابي	الناظور	ECS						2									1	1	4
			ECT																	
5	الثانوية التأهيلية مولاي ادريس	فاس	MP					2												3
			PSI													1				
			ECT																	
6	الثانوية التأهيلية عمر بن الخطاب	مكناس	MP	1				1			1							2	1	7
			PSI																	
7	الثانوية التأهيلية مولاي إسماعيل	مكناس	ECS	1											1				1	3
			ECT																	
8	الثانوية التأهيلية الشريف الإدريسي	تازة	MP					1	2									1		5
			ECS												1					
9	الثانوية التأهيلية مولاي يوسف	الرباط	MP					1			1								1	3
			PSI																	
10	الثانوية التأهيلية عمر الخيام	الرباط	ECS													1			1	2
			ECT																	
11	الثانوية التأهيلية سلمان الفارسي	سلا	MP			1		2	1				1					1	2	8
			TSI																	
12	الثانوية التأهيلية محمد السادس	القنيطرة	MP		1			2												5
			PSI																	
			ECS												1	1				
			ECT																	
13	الثانوية التأهيلية محمد الخامس	بني ملال	MP						1				1							2
			TSI																	
14	الثانوية التأهيلية إبن عبدون	خريبكة	MP					1											1	2
15	الثانوية التأهيلية محمد الخامس	الفداء	MP	1				3										3		9
			PSI		1			1												
16	الثانوية التأهيلية الخنساء	أنفا	MP					2								1	1	4		8
			ECS																	
			ECT																	
17	الثانوية التقنية	المحمدية	MP						1		1									2
			TSI																	
18	الثانوية التقنية الرازي	الجديدة	MP		1			1											2	6
			PSI																	
			ECS	1											1					
			ECT																	
19	الثانوية التقنية	سطات	TSI								1								1	2
			ECS																	
			ECT																	
20	الثانوية التأهيلية إبن تيمية	مراكش	MP		1		1	2												5
			PSI													1				
			ECT																	
21	الثانوية التأهيلية مولاي عبد الله	أسفي	MP					1												9
			TSI						1										2	
			ECS	2											2	1				
22	الثانوية التأهيلية مولاي علي الشريف	الراشيدية	MP					2										1		3
23	الأقسام التحضيرية وزارات	وزارات	PSI		1			1												12
			ECS						3						3	2		2		
			ECT																	
24	الثانوية التأهيلية رضا السلاوي	أكادير	MP			1											1			7
			PSI					1												
			TSI																	
			ECS	1											1					
			ECT																	
25	الثانوية التأهيلية باب الصحراء	كلميم	MP		1			1												13
			PSI					1	1			4								
			ECS												1					
26	الثانوية التأهيلية لسان الدين بن الخطيب	العيون	MP					1												5
			ECT						2						1		1			
27	الثانوية التأهيلية علقة بن نافع	الداخلة	PSI					1										1		3
28	المدرسة الملكية الجوية	مراكش	MP					1												1
29	المدرسة الملكية البحرية	أنفا	MP	1	2			2	1											10
مجموع حاجيات مراكز الأقسام التحضيرية حسب المسلك			MP	4	6	1	1													
			PSI		2															
			TSI																	
			ECS	5																
			ECT																	
مجموع حاجيات مراكز الأقسام التحضيرية				9	8	4	1	35	15		9		2		13	7	6	21	20	150

Figure 2: Exemple d'une grille des besoins lancée par le ministère de l'Éducation

Pour la conception du système d'affectation, les données de base proviennent de deux fichiers au format JSON :

- **besoins_etablissements.json** : ce fichier contient la description détaillée des besoins en enseignants pour chaque établissement. On y retrouve, pour chaque structure, les filières, niveaux et matières à pourvoir, ainsi que le nombre de postes nécessaires et leur localisation.
- **professeurs_mutation.json** : ce fichier regroupe les informations relatives aux professeurs candidats à l'affectation ou à la mutation. Il inclut leur identité, leurs matières enseignées, leurs niveaux de compétence, leurs filières possibles, leur établissement actuel, leur ville, leurs choix d'affectation classés par ordre de préférence, leur score calculé selon des critères prédéfinis, ainsi que des indicateurs comme le rapprochement familial ou la disponibilité.

Figure 3: Exemple d'extrait du fichier besoins_etablissements.json

```
[
  {
    "id": "P001",
    "nom": "Salma Ouarzaz",
    "filiere": "TSI",
    "matiere": "Physique",
    "niveau": [
      "2ème année",
      "1ère année"
    ],
    "score": 81,
    "rapprochement": false,
    "etablissement_actuel": "Lycée Ribat (Meknès)",
    "choix": [
      "Lycée Oued",
      "Lycée Al Qods"
    ],
    "ville": "Tanger"
  },
  {
    "id": "P002",
    "nom": "Youssef Mahmoud",
    "filiere": "ECS",
    "matiere": "Chimie",
    "niveau": [
      "2ème année",
      "1ère année"
```


Figure 4: Exemple d'extrait du fichier professeurs_mutation.json

3. Algorithme hongrois pour l'affectation optimale

3.1 Introduction

L'algorithme hongrois, également connu sous le nom d'algorithme de Kuhn-Munkres, est une méthode combinatoire efficace pour résoudre le **problème d'affectation**. Ce problème consiste à attribuer un ensemble d'éléments (dans notre cas, des professeurs) à un autre ensemble (les postes à pourvoir) de manière à **optimiser un critère global** (ici, maximiser le score total d'affectation), tout en respectant des contraintes de compatibilité.

Dans le cadre de ce projet, l'algorithme hongrois est utilisé pour déterminer la meilleure correspondance possible entre les professeurs et les postes définis dans la grille de besoins, en tenant compte :

- Des **contraintes strictes** (matière, niveau, filière).
- Des **préférences exprimées** par les enseignants (choix d'affectation).
- Des **critères pondérés** (score, rapprochement familial, localisation, etc.).

L'algorithme est particulièrement adapté à ce type de problème car il garantit une solution optimale en un temps polynomial, même lorsque le nombre de professeurs et de postes est important.

3.2 Principe général du problème d'affectation

Le problème d'affectation est une problématique classique en recherche opérationnelle et en optimisation combinatoire. Il consiste à attribuer un ensemble de ressources (dans notre cas, des professeurs) à un ensemble de tâches ou de postes (les besoins des établissements) de manière à **optimiser un objectif global** tout en respectant un ensemble de contraintes. Dans le contexte de ce projet, l'objectif est de maximiser un **score global d'affectation** qui reflète à la fois :

- La satisfaction des besoins institutionnels (matière, niveau, filière, localisation).
- Les préférences et priorités des enseignants (choix d'affectation, rapprochement familial, etc.).

Ce problème se prête particulièrement bien à une modélisation mathématique et algorithmique, car il peut être représenté comme un **graphe biparti pondéré** ou comme une **matrice de compatibilité/poids**.

a. Définition formelle

On considère deux ensembles :

- $P = \{p_1, p_2, \dots, p_m\}$: l'ensemble des professeurs à affecter.
- $J = \{j_1, j_2, \dots, j_n\}$: l'ensemble des postes à pourvoir.

Chaque couple (p_i, j_k) est associé à un **score** $S_{\{ik\}}$qui mesure la pertinence de l'affectation du professeur $p_i au\ poste\ j_k$. Ce score est calculé à partir de plusieurs critères :

- **Compatibilité stricte** : matière, niveau, filière (si non respectée → $score = -\infty$ ou exclusion).

- **Préférences** : rang du choix exprimé par le professeur.
- **Critères sociaux** : rapprochement familial, proximité géographique.
- **Autres pondérations** : ancienneté, performance, contraintes administratives.

L'objectif est de trouver une **affectation binaire** $x_{\{ik\}}$ telle que :

$$x_{ik} = \begin{cases} 1 & \text{si le professeur } p_i \text{ est affecté au poste } j_k \\ 0 & \text{sinon} \end{cases}$$

b. Formulation mathématique

Le problème peut être formulé comme suit :

$$\max \sum_{i=1}^{m} \sum_{k=1}^{n} S_{ik} \cdot x_{ik}$$

Sous contraintes :

1. **Un professeur au plus par poste** :

$$\sum_{i=1}^{m} x_{ik} \leq 1 \quad \forall k$$

2. **Un poste au plus par professeur** :

$$\sum_{k=1}^{n} x_{ik} \leq 1 \quad \forall i$$

3. **Compatibilité stricte** :

$$x_{ik} = 0 \quad \text{si } p_i \text{ et } j_k \text{ sont incompatibles}$$

4. **Binarité** :

$$x_{ik} \in \{0,1\}$$

c. Représentation en graphe biparti

Le problème peut être représenté par un **graphe biparti pondéré** :

- Les sommets de gauche représentent les professeurs.
- Les sommets de droite représentent les postes.
- Chaque arête reliant un professeur à un poste est pondérée par le score $S_{\{ik\}}$.
- L'objectif est de trouver un **appairage parfait** qui maximise la somme des poids.

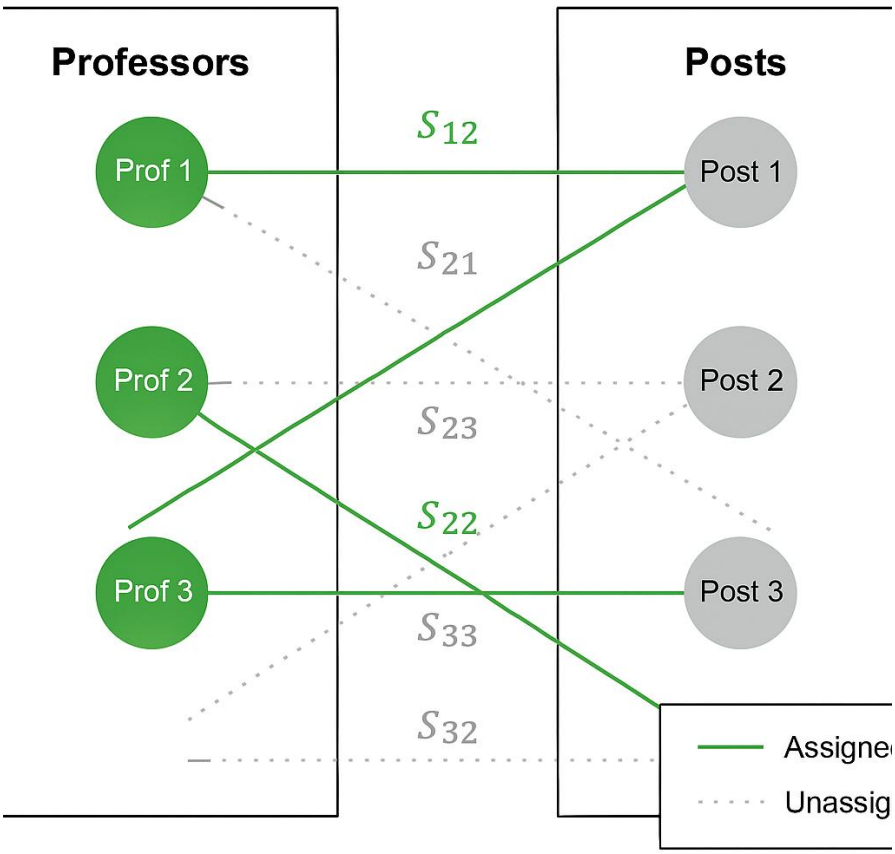


Figure 5: Schéma du problème d'affectation sous forme de graphe biparti

d. Représentation matricielle

Une autre représentation courante est la **matrice de compatibilité/poids** :

- Les lignes représentent les professeurs.
- Les colonnes représentent les postes.
- Chaque cellule contient le score S_{ik} ou une valeur très négative si l'affectation est impossible.

Professeur / Poste	Poste 1	Poste 2	Poste 3
Prof 1	85	60	- ∞
Prof 2	70	90	50
Prof 3	- ∞	75	80

Tableau 2: Exemple de matrice de compatibilité

3.3Transformation en problème de minimisation

a. Pourquoi transformer le problème

L'algorithme hongrois repose sur des opérations de **réduction de lignes et de colonnes** dans une matrice de coûts. Ces opérations n'ont de sens que si les valeurs représentent des coûts à minimiser. Si l'on appliquait l'algorithme directement sur une matrice de scores à maximiser, les étapes de réduction produiraient des résultats incohérents, car elles chercheraient à réduire des valeurs élevées (bonnes affectations) au lieu de les préserver.

La transformation permet donc de **conserver l'optimalité** tout en respectant la mécanique de l'algorithme.

b. Principe de la transformation

L'idée est simple :

- Plus un score est élevé, plus le coût associé doit être faible.
- On peut donc inverser l'échelle en soustrayant chaque score à une constante suffisamment grande.

La formule générale est :

$$C_{\{ij\}} = S_{\{max\}} - S_{\{ij\}}$$

où :

- $C_{\{ij\}}$: est le coût associé à l'affectation du professeur i au poste j.
- $S_{\{ij\}}$: est le score initial calculé pour cette affectation.
- $S_{\{max\}}$: est le score maximum présent dans la matrice S.

Ainsi :

- Si $S_{\{ij\}}$ est élevé (bonne affectation), $C_{\{ij\}}$ sera faible.
- Si $S_{\{ij\}}$ est faible (mauvaise affectation), $C_{\{ij\}}$ sera élevé.

3.4 Étapes de l'algorithme hongrois

❖ Étape 1 – Réduction des lignes

Objectif : garantir qu'au moins un zéro apparaisse dans chaque ligne de la matrice.

Procedure :

1. Pour chaque ligne de la matrice de coûts, on identifie la valeur minimale.
2. On soustrait cette valeur à tous les éléments de la ligne.
3. Résultat : chaque ligne contient au moins un zéro.

Intuition : Cette étape normalise les coûts par ligne, en mettant en évidence les affectations les plus avantageuses pour chaque professeur.

Exemple : Si la ligne pour Prof 1 est [5, 30, 50], le minimum est 5. Après soustraction : [0, 25, 45].

❖ Étape 2 – Réduction des colonnes

Objectif : garantir qu'au moins un zéro apparaisse dans chaque colonne.

Procédure :

1. Pour chaque colonne, on identifie la valeur minimale.
2. On soustrait cette valeur à tous les éléments de la colonne.
3. Résultat : chaque colonne contient au moins un zéro.

Intuition : Cette étape complète la normalisation en équilibrant les coûts par colonne, ce qui permet de préparer la recherche d'affectations optimales.

Exemple : Si la colonne pour Poste 2 est [25, 0, 15], le minimum est 0 → la colonne reste inchangée.

❖ Étape 3 – Couverture minimale des zéros

Objectif : déterminer si une affectation optimale est déjà possible.

Procédure :

1. On trace le **nombre minimal de lignes et de colonnes** nécessaires pour couvrir tous les zéros de la matrice.
2. Si ce nombre est égal à n (taille de la matrice), une affectation optimale est possible → passer à l'étape finale.
3. Sinon, passer à l'étape 4.

Méthode de couverture :

- Identifier les lignes sans affectation.
- Marquer les colonnes contenant des zéros dans ces lignes.
- Marquer les lignes contenant des zéros dans ces colonnes.
- Répéter jusqu'à ce qu'aucun nouveau marquage ne soit possible.
- Les lignes non marquées et les colonnes marquées définissent la couverture minimale.

Intuition : Cette étape vérifie si la matrice contient déjà suffisamment de zéros pour construire une solution optimale.

❖ Étape 4 – Ajustement de la matrice

Objectif : créer de nouveaux zéros pour permettre une affectation optimale.

Procédure :

1. Identifier la plus petite valeur non couverte par les lignes/colonnes tracées.

2. Soustraire cette valeur à tous les éléments non couverts.
3. Ajouter cette valeur à tous les éléments couverts deux fois (intersection ligne + colonne).
4. Revenir à l'étape 3.

Intuition : Cette étape réduit encore certains coûts pour créer de nouvelles opportunités d'affectation.

Exemple : Si la plus petite valeur non couverte est 5, on soustrait 5 aux cases non couvertes et on ajoute 5 aux cases couvertes deux fois.

❖ Étape 5 – Affectation finale

Objectif : sélectionner les zéros qui définissent l'affectation optimale.

Procédure :

1. Choisir un zéro dans une ligne/colonne qui n'a pas encore été affectée.
2. Marquer cette affectation et barrer la ligne et la colonne correspondantes.
3. Répéter jusqu'à ce que tous les professeurs soient affectés à un poste.

Intuition : On choisit les zéros de manière à ce qu'aucun professeur ni poste ne soit utilisé plus d'une fois.

Résultat : On obtient un ensemble d'affectations qui minimise le coût total (et donc maximise le score initial).

3.5 Application au contexte du projet

Le contexte est caractérisé par la coexistence de deux ensembles de données distincts mais complémentaires :

- Les **besoins des établissements** (besoins_etablissements.json), qui décrivent pour chaque structure les postes à pourvoir, avec des informations précises sur la matière, le niveau, la filière et la localisation.
- Les **professeurs en mutation** (professeurs_mutation.json), qui regroupent les informations personnelles et professionnelles des enseignants candidats à un changement d'affectation, ainsi que leurs préférences et contraintes.

L'objectif est de transformer ces données brutes en une **matrice de compatibilité pondérée**, puis d'utiliser l'algorithme hongrois pour déterminer la meilleure correspondance possible entre les deux ensembles, en maximisant la satisfaction globale et en respectant les contraintes institutionnelles.

Dans notre application, les **lignes** de la matrice représentent les professeurs et les **colonnes** représentent les postes à pourvoir. Chaque cellule de la matrice contient un **score** calculé à partir de plusieurs critères :

- **Compatibilité stricte** : un professeur ne peut être affecté qu'à un poste correspondant à sa matière, son niveau et sa filière. Si cette compatibilité n'est pas respectée, la cellule reçoit une valeur d'exclusion (par exemple $-\infty$ ou un coût très élevé après transformation).
- **Préférences exprimées** : un bonus est accordé si le poste correspond au premier, deuxième ou troisième choix du professeur.

- **Critères sociaux** : un bonus supplémentaire est attribué en cas de rapprochement familial ou si le poste est situé dans la même ville que l'établissement actuel.
- **Autres pondérations** : ancienneté, performance, contraintes administratives éventuelles.

Une fois les scores calculés, la matrice est transformée en **matrice de coûts** pour s'adapter à la formulation de minimisation de l'algorithme hongrois. Cette transformation garantit que les affectations les plus avantageuses (scores élevés) correspondent à des coûts faibles.

3.6 Gestion des cascades de mutations

a. Principe général

Le fonctionnement repose sur une logique simple mais puissante. Après l'affectation initiale réalisée par l'algorithme hongrois, certains professeurs quittent leur établissement pour rejoindre un nouveau poste. Leur départ libère des postes qui, à leur tour, peuvent être proposés à d'autres enseignants. Si un professeur est affecté à l'un de ces postes libérés, il libère à son tour son ancien poste, et ainsi de suite. Ce processus se poursuit tant qu'il existe des postes vacants pouvant être attribués à des candidats compatibles. On parle alors de **cascade de mutations**, car chaque mouvement déclenche un autre mouvement, créant une chaîne d'affectations successives.

b. Mise en œuvre dans le système

Dans notre application, la gestion des cascades de mutations intervient immédiatement après la première exécution de l'algorithme hongrois. Le système :

1. Identifie tous les postes libérés par les professeurs mutés.
2. Ajoute ces postes à une **liste dynamique de besoins résiduels**.
3. Relance un processus d'affectation ciblé, qui peut être :
    - Une nouvelle exécution de l'algorithme hongrois sur un sous-ensemble réduit (professeurs restants et postes vacants).
    - Ou une procédure simplifiée si le nombre de postes et de candidats est faible.

Chaque itération applique les mêmes règles de compatibilité et de pondération que l'affectation initiale : matière, niveau, filière, rapprochement familial, rang du choix, proximité géographique, etc..

3.7 Complexité et performance

La complexité temporelle de l'algorithme hongrois est de l'ordre de $O(n^3)$, où n représente la dimension de la matrice carrée de coûts (nombre de professeurs ou de postes, après équilibrage). Cela signifie que le temps de calcul croît proportionnellement au cube du nombre d'éléments à traiter. Par exemple :

- Pour n = 10, le nombre d'opérations est de l'ordre de 1 000 , ce qui est quasi instantané sur un ordinateur moderne.
- Pour n = 100, on atteint environ 1 000 000 opérations, ce qui reste très raisonnable.
- Même pour n = 500, le volume de calcul reste gérable avec des ressources informatiques actuelles.

Cette complexité est très favorable comparée à une recherche exhaustive, qui aurait une complexité de $O(n!)$ et deviendrait impraticable dès n supérieur à 15.

3.8 Avantages de l'algorithme hongrois dans ce projet

L'utilisation de l'algorithme hongrois dans ce projet présente un avantage majeur : il garantit l'obtention d'une solution optimale pour l'affectation des professeurs aux postes disponibles, en maximisant la satisfaction globale tout en respectant les contraintes définies. Sa capacité à traiter simultanément un grand nombre de candidats et de postes, avec des critères multiples tels que la compatibilité matière/niveau, les préférences exprimées, le rapprochement familial ou l'ancienneté, en fait un outil parfaitement adapté à la complexité du contexte éducatif. Grâce à sa complexité polynomiale $O(n^3)$, il reste performant même pour des volumes importants de données, ce qui permet de lancer plusieurs itérations, notamment pour gérer les cascades de mutations, sans compromettre les délais de traitement. Sa flexibilité est un autre atout : il suffit d'ajuster la matrice de coûts pour intégrer de nouvelles règles ou pondérations, sans modifier la structure de l'algorithme. Cette adaptabilité garantit la pérennité de la solution face à l'évolution des politiques éducatives ou des priorités administratives.

Au-delà de ses performances techniques, l'algorithme hongrois apporte une réelle valeur ajoutée en termes d'équité et de transparence. Sa nature déterministe assure que, pour un même jeu de données et les mêmes paramètres, il produira toujours la même solution, ce qui renforce la confiance des parties prenantes et élimine toute suspicion de traitement arbitraire. Chaque affectation peut être justifiée par un score calculé et par les étapes de traitement, offrant ainsi une traçabilité complète des décisions. Cette transparence est renforcée par l'intégration avec le module explicatif du projet, qui permet de détailler les raisons de chaque affectation. Enfin, en optimisant l'utilisation des ressources humaines disponibles et en réduisant le nombre de postes vacants, l'algorithme contribue directement à l'efficacité globale du système éducatif, tout en améliorant la satisfaction des enseignants et des établissements.

4. Affichage des résultats

Cette section présente la restitution finale produite par l'algorithme hongrois après traitement des données d'entrée et, le cas échéant, gestion des cascades de mutations. L'objectif est de fournir une vision synthétique et détaillée des affectations réalisées, des postes restés vacants et des indicateurs de performance de la campagne. L'affichage est conçu pour être à la fois lisible par un opérateur humain et exploitable par des systèmes tiers.

```
=== RÉSULTATS DE L'AFFECTATION ===
Total affectations: 150
Postes non pourvus: 482
Professeurs non affectés: 0
Itérations de cascade: 130

=== AFFECTATIONS ===
Salma Ouarzaz -> Lycée Al Qods | TSI | 1ère année | Physique [Ancien]
Youssef Mahmoud -> Lycée Masira | ECS | 2ème année | Chimie [Ancien]
Fatima Choukri -> Lycée Medina | ECS | 1ère année | traduction [Ancien]
Samir Mansouri -> Lycée El Fassi | ECS | 1ère année | Physique [Ancien]
Walid Rifi -> Lycée Atlas | TSI | 1ère année | Chimie [Ancien]
Rachid Mansouri -> Lycée Bouregreg | MP | 2ème année | Informatique [Nouveau]
Salma Bouayad -> Lycée Atlas | TSI | 2ème année | SI [Rapprochement]
Mohammed Saâdi -> Lycée Bouregreg | MP | 1ère année | traduction [Ancien]
Imane Mansouri -> Lycée Tiznit | MP | 2ème année | Physique [Ancien]
Mohammed Bouayad -> Lycée Masira | ECT | 2ème année | SI [Rapprochement]
Samir Choukri -> Lycée Al Khawarizmi | PSI | 2ème année | Chimie [Ancien]
Abdellah Ait Ben -> Lycée Tiznit | MP | 2ème année | Anglais [Rapprochement]
Salma Choukri -> Lycée Saadi | ECS | 2ème année | Français [Rapprochement]
Salma Zoufri -> Lycée Ibn Sina | TSI | 2ème année | Français [Nouveau]
Yassine Choukri -> Lycée Ibn Sina | ECS | 2ème année | Mathématiques [Rapprochement]
Mohammed Benomar -> Lycée Al Khawarizmi | ECT | 2ème année | Physique [Rapprochement]
Nabil Ait Ben -> Lycée El Fassi | PSI | 1ère année | Mathématiques [Rapprochement]
Abdellah Ouarzaz -> Lycée Kasbah | MP | 1ère année | Français [Rapprochement]
...
- 1er choix satisfaits: 86
- 2ème choix satisfaits: 19
- 3ème choix satisfaits: 0
- Score moyen: 89.98
```

Figure 6: Exemple d'affichage des résultats de l'algorithme hongrois

L'affichage des résultats constitue la phase finale du processus d'affectation. Il permet de restituer, de manière claire et exploitable, les décisions prises par l'algorithme hongrois après traitement des données et gestion éventuelle des cascades de mutations. Cette étape ne se limite pas à une simple liste : elle fournit à la fois une vision globale de la campagne et un détail précis des affectations, tout en garantissant la transparence et la traçabilité des choix effectués.

II. Chatbot explicatif et interactif

1. Introduction

Dans le cadre de ce projet, le chatbot explicatif occupe une place centrale en tant qu'interface intelligente entre le système d'affectation et ses utilisateurs finaux. Conçu pour accompagner et clarifier les résultats produits par l'algorithme hongrois, il a pour mission de rendre les décisions d'affectation compréhensibles, transparentes et justifiables. Dans un contexte où les règles de compatibilité, les pondérations et les priorités peuvent sembler complexes, cet outil interactif permet de traduire les calculs techniques en explications claires et adaptées au profil de chaque interlocuteur.

L'intégration du chatbot répond à un double objectif : améliorer l'accessibilité des informations et renforcer la confiance dans le processus d'affectation. En permettant aux enseignants, aux chefs d'établissement et aux

services administratifs de poser des questions précises sur les affectations, il offre un canal direct d'explication et de dialogue. Chaque réponse s'appuie sur les données réelles issues du système, garantissant ainsi la fiabilité des informations fournies. Cette approche contribue à réduire les incompréhensions, à limiter les contestations et à fluidifier la communication entre les différentes parties prenantes.

Enfin, le chatbot s'inscrit dans une logique d'innovation et de modernisation des outils de gestion des ressources humaines dans l'éducation. Sa capacité à interagir en langage naturel, à fournir des réponses contextualisées et à s'adapter aux besoins spécifiques de chaque utilisateur en fait un levier stratégique pour optimiser l'expérience globale. Au-delà de son rôle d'assistant virtuel, il devient un véritable médiateur numérique, capable de rapprocher la complexité algorithmique des attentes humaines, tout en soutenant les objectifs de transparence, d'efficacité et d'équité du projet.

2. Cahier des charges fonctionnel

2.1. Objectif général

Le cahier des charges fonctionnel définit l'ensemble des fonctionnalités attendues du **chatbot explicatif** intégré au système d'affectation des professeurs. L'objectif principal est de fournir un outil interactif capable de répondre, en langage naturel, aux questions des utilisateurs concernant les résultats produits par l'algorithme hongrois.

Il doit permettre de **comprendre, justifier et analyser** chaque affectation, tout en garantissant la transparence et la fiabilité des informations transmises. Ce module s'inscrit dans une démarche d'amélioration de l'expérience utilisateur, en réduisant les incompréhensions et en facilitant l'accès aux données de manière simple et intuitive.

2.2. Public cible

Le chatbot doit s'adresser à plusieurs profils d'utilisateurs, chacun ayant des besoins spécifiques :

- **Enseignants** : comprendre leur affectation, vérifier la prise en compte de leurs préférences et contraintes.
- **Chefs d'établissement** : consulter les affectations de leur établissement et comprendre les arbitrages effectués.
- **Services administratifs** : disposer d'un outil d'explication rapide pour répondre aux demandes et réclamations.
- **Décideurs** : analyser les résultats globaux et identifier les points d'amélioration du processus.

2.3. Contraintes et exigences

Le système doit respecter un ensemble de contraintes :

- **Confidentialité** : seules les données autorisées doivent être accessibles selon le profil utilisateur.
- **Exactitude** : les réponses doivent être basées sur les données réelles issues du système d'affectation.
- **Disponibilité** : le chatbot doit être accessible 24h/24 et 7j/7.
- **Performance** : temps de réponse inférieur à 3 secondes pour la majorité des requêtes.

- **Ergonomie** : interface claire, intuitive et adaptée aux différents supports (ordinateur, tablette, smartphone).

2.4. Fonctionnalités principales

Les fonctionnalités principales du chatbot explicatif reposent sur sa capacité à fournir, en langage naturel, des réponses précises, contextualisées et personnalisées aux questions des utilisateurs concernant les résultats de l'algorithme d'affectation. Il permet d'effectuer des recherches ciblées sur un professeur, un poste ou un établissement, et d'expliquer en détail les critères et pondérations ayant conduit à une affectation donnée, y compris la prise en compte des préférences, de la compatibilité matière/niveau, du rapprochement familial ou de l'ancienneté. Le chatbot offre également la possibilité de comparer plusieurs affectations, de présenter les résultats sous forme synthétique ou détaillée selon le besoin, et de gérer les cas complexes comme les cascades de mutations ou les incompatibilités. Grâce à un accès direct aux données réelles et à un moteur d'explication structuré, il garantit la fiabilité des informations fournies, tout en proposant une interface ergonomique et accessible sur différents supports, afin de rendre la consultation fluide, rapide et adaptée aux profils variés des utilisateurs.

3. Architecture technique

3.1 Introduction

L'architecture technique du système a été conçue comme une structure modulaire et évolutive, capable d'intégrer harmonieusement l'algorithme hongrois, la gestion des cascades de mutations et un moteur d'explication basé sur la technologie RAG pour le chatbot, tout en garantissant performance, sécurité et maintenabilité. Elle repose sur une séparation claire des couches données, traitement, explication et présentation afin de faciliter la maintenance, d'optimiser les flux d'information et de permettre l'évolution future du système sans remise en cause de son cœur fonctionnel. Cette organisation permet de traiter efficacement de grands volumes de données, de produire des résultats fiables et traçables, puis de les rendre accessibles et compréhensibles via des interfaces adaptées aux différents profils d'utilisateurs, assurant ainsi transparence, rapidité et pertinence dans l'ensemble du processus d'affectation.

3.2 Vue d'ensemble de l'architecture

L'architecture globale du système a été pensée comme un ensemble cohérent de composants interconnectés, organisés en couches logiques distinctes mais complémentaires, afin de garantir la **clarté fonctionnelle**, la **modularité** et la **scalabilité**. Cette structuration permet de séparer les responsabilités, de faciliter la maintenance et d'assurer que chaque évolution ou mise à jour puisse être réalisée sans perturber l'ensemble du dispositif. L'objectif est de disposer d'une infrastructure capable de traiter efficacement de grands volumes de données, d'exécuter des calculs complexes (notamment l'algorithme hongrois et la gestion des cascades de mutations), puis de restituer les résultats de manière intelligible grâce à un moteur d'explication basé sur la technologie **RAG.**

L'architecture repose sur **quatre couches principales** :

1. **Couche de données** – Elle centralise toutes les informations nécessaires au fonctionnement du système : données sur les enseignants, les postes, les préférences, les règles d'affectation, les pondérations et les résultats. Cette couche inclut également les mécanismes d'indexation et d'optimisation des requêtes pour accélérer l'accès aux données, notamment pour le moteur RAG.

2. **Couche de traitement** – Elle regroupe les modules de logique métier, dont la préparation des données, l'exécution de l'algorithme hongrois, la gestion des cascades de mutations et le calcul des indicateurs de performance (taux de satisfaction, postes vacants, score moyen). Cette couche est optimisée pour garantir des temps de calcul réduits même sur de grands ensembles de données.

3. **Couche d'explication (RAG)** – Véritable passerelle entre les données brutes et l'utilisateur final, elle permet au chatbot d'interroger la base, de récupérer les informations pertinentes et de générer des réponses contextualisées et compréhensibles. Cette couche s'appuie sur des techniques de traitement du langage naturel (NLP) pour analyser les requêtes et sur un moteur de génération augmentée pour formuler les explications.

4. **Couche de présentation** – Elle regroupe toutes les interfaces par lesquelles les utilisateurs interagissent avec le système : interface web, chatbot interactif, affichage en terminal, et version mobile. Elle est conçue pour offrir une expérience fluide, ergonomique et adaptée aux différents profils d'utilisateurs (enseignants, chefs d'établissement, services administratifs).

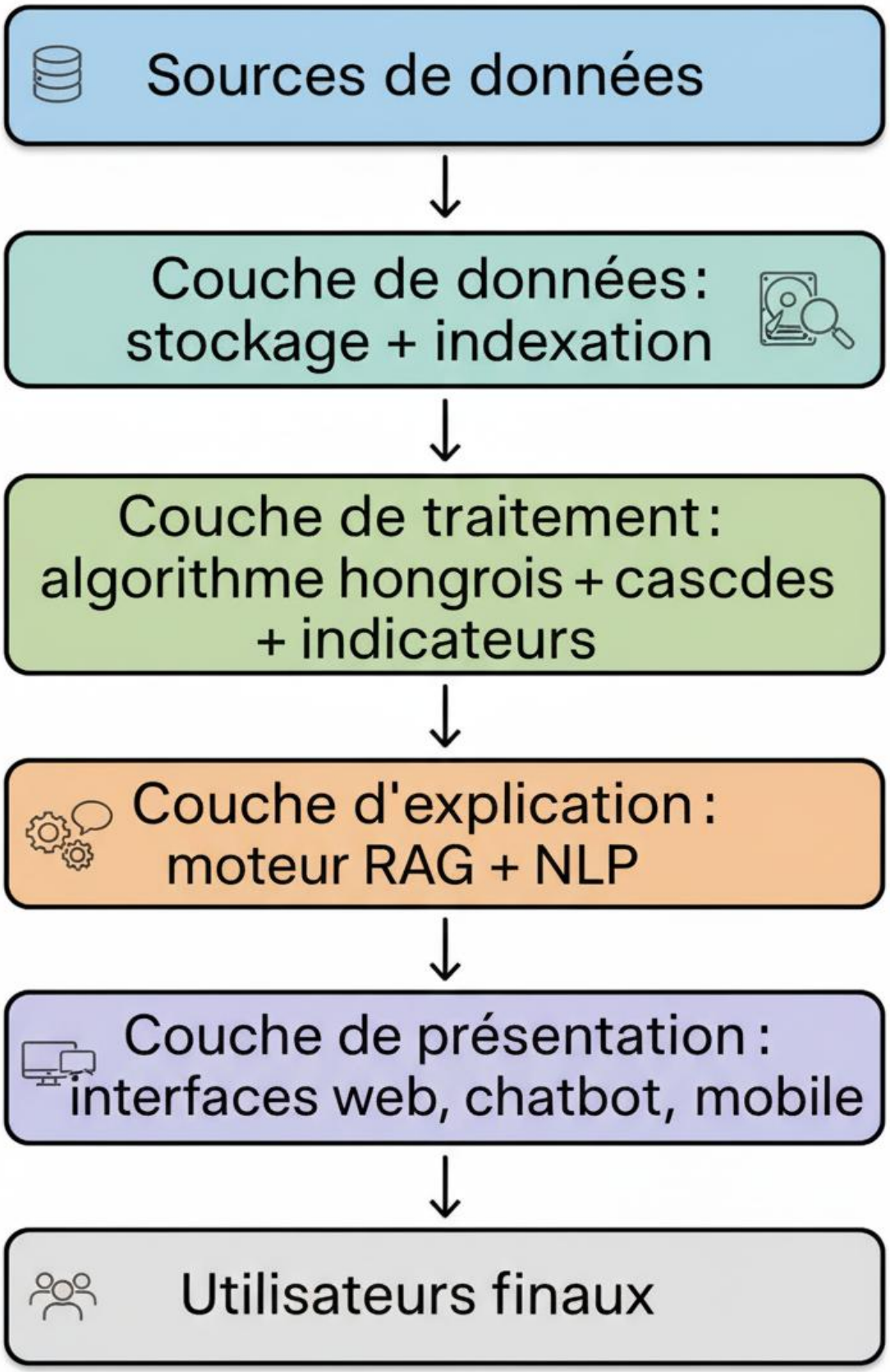


Figure 7: Schéma global de l'architecture du système.

3.3 Couche de données

La couche de données, dans le cadre de cette architecture, est exclusivement dédiée au **stockage et à la gestion des résultats finaux d'affectation** produits par l'algorithme hongrois et la gestion des cascades de mutations. Elle ne conserve **aucune donnée brute ou intermédiaire**, mais uniquement les résultats validés, afin de garantir la clarté, la traçabilité et la conformité aux besoins du moteur d'explication et des interfaces de restitution.

Ces résultats sont archivés et organisés dans trois formats complémentaires :

- **CSV** : format tabulaire, idéal pour l'exploitation dans des tableurs ou pour des traitements statistiques rapides.
- **JSON** : format structuré et hiérarchisé, parfaitement adapté aux échanges avec le moteur RAG et aux intégrations dans des applications web ou API.
- **TXT** : format simplifié, utilisé pour des exports légers, des consultations rapides ou des sauvegardes archivées.

3.4 Couche de traitement

La couche de traitement représente le **noyau opérationnel** du système, là où s'exécute toute la logique métier permettant de transformer les données issues de la couche de stockage en résultats d'affectation exploitables. Elle commence par un **module de préparation des données**, qui assure le nettoyage, la validation et la mise en forme des fichiers entrants (CSV, JSON, TXT) afin de les rendre compatibles avec les traitements ultérieurs. Cette étape inclut la détection et la correction d'incohérences, la normalisation des formats et la construction d'une **matrice de compatibilité pondérée** reliant chaque enseignant à chaque poste potentiel. Une fois cette matrice constituée, elle est transmise au **module d'optimisation**, qui applique l'algorithme hongrois pour déterminer l'affectation optimale en tenant compte des contraintes réglementaires, des préférences exprimées et des pondérations définies. Ce processus garantit que chaque poste est attribué de manière à maximiser la satisfaction globale tout en respectant les priorités établies.

Après l'exécution de l'algorithme principal, la couche de traitement active le **module de gestion des cascades de mutations**, qui identifie les postes libérés par les premières affectations et procède à des réaffectations successives jusqu'à stabilisation complète du système. Vient ensuite le **module de calcul des indicateurs**, chargé de produire des mesures clés telles que le taux de satisfaction des choix, le nombre de postes vacants restants ou le score moyen par affectation. Enfin, un **module de génération des résultats finaux** compile l'ensemble des données validées et les exporte dans les formats prévus (CSV, JSON, TXT), prêts à être consommés par la couche d'explication (moteur RAG) et les interfaces de présentation. Cette organisation modulaire permet non seulement d'optimiser les performances et la précision des affectations, mais aussi de faciliter la maintenance et l'évolution future du système.

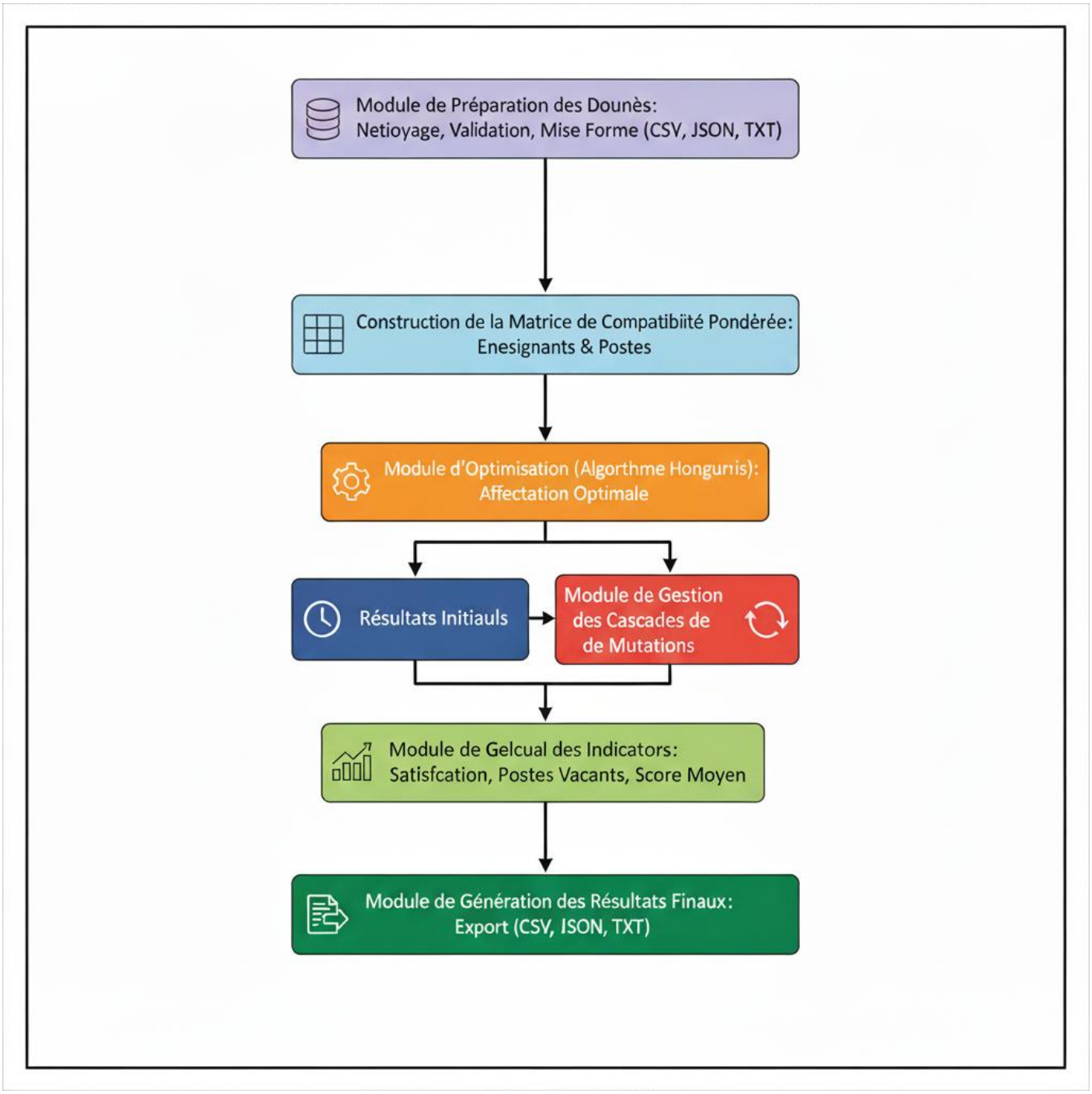


Figure 8: Flux de traitement

3.5 Moteur RAG

a. Rôle du RAG

Dans notre système, le RAG est conçu pour **exploiter directement les résultats finaux d'affectation** (stockés en formats CSV, JSON et TXT) et les règles associées, afin de fournir des explications transparentes sur les décisions prises par l'algorithme hongrois et les cascades de mutations. Lorsqu'un utilisateur interroge le chatbot par exemple pour comprendre pourquoi un enseignant a été affecté à un poste précis ou pourquoi un autre poste n'a pas été attribué — le RAG commence par analyser la requête grâce à un module de **NLP**. Il identifie les entités clés (nom de l'enseignant, identifiant du poste, établissement, critère de priorité) et l'intention de la question. Ensuite, le module Retrieval interroge les fichiers de résultats et les métadonnées

pour extraire uniquement les informations pertinentes. Ces données sont ensuite transmises au module Generation, qui les reformule en une réponse claire, structurée et adaptée au contexte, tout en respectant le niveau de détail attendu par le profil utilisateur (enseignant, chef d'établissement, agent administratif).

Le rôle du RAG ne se limite pas à **expliquer** : il contribue aussi à **renforcer la confiance** des utilisateurs dans le système en rendant les décisions traçables et vérifiables. Chaque réponse générée peut être reliée à ses données sources, ce qui permet de justifier objectivement les affectations. De plus, le RAG est conçu pour être **évolutif** : il peut intégrer de nouvelles sources d'information (par exemple, des notes de service ou des règles mises à jour) sans nécessiter de refonte complète. Enfin, il joue un rôle clé dans l'**expérience utilisateur**, en transformant un ensemble de données techniques et complexes en explications pédagogiques, compréhensibles et personnalisées, ce qui facilite l'appropriation des résultats par toutes les parties prenantes.

b. Étapes de construction du chatbot RAG

1. Analyse des besoins et définition des cas d'usage

La première étape consiste à identifier précisément les objectifs du chatbot et les scénarios dans lesquels il interviendra. Dans notre contexte, il s'agit de fournir des explications claires et traçables sur les résultats d'affectation produits par l'algorithme hongrois et les cascades de mutations. Cette phase implique de définir les types de questions que les utilisateurs (enseignants, chefs d'établissement, services administratifs) sont susceptibles de poser, le niveau de détail attendu dans les réponses, ainsi que les contraintes réglementaires ou organisationnelles à respecter. Cette analyse sert de base à la conception de l'architecture RAG et à la structuration des données nécessaires.

2. Préparation et structuration des données

Une fois les besoins définis, il est essentiel de préparer les données qui alimenteront le module Retrieval. Dans notre système, cela signifie extraire les résultats finaux d'affectation (CSV, JSON, TXT) et les enrichir avec des métadonnées pertinentes : identifiants des enseignants et des postes, scores, rangs des choix, règles appliquées, motifs de priorisation. Ces données sont ensuite indexées dans un moteur de recherche interne optimisé pour les requêtes textuelles et sémantiques. L'objectif est de garantir que le RAG puisse retrouver rapidement et avec précision les informations nécessaires à la génération d'une réponse.

3. Conception du module Retrieval

Le module Retrieval est chargé de localiser, dans la base indexée, les éléments les plus pertinents par rapport à la question posée. Sa conception implique le choix d'une méthode de recherche adaptée : recherche par mots-clés, recherche vectorielle basée sur des embeddings, ou combinaison des deux. Dans notre cas, un moteur hybride est privilégié pour capter à la fois les correspondances exactes (ex. identifiant de poste) et les correspondances sémantiques (ex. "mutation pour rapprochement familial"). Ce module doit être optimisé pour minimiser le temps de réponse tout en maximisant la pertinence des documents récupérés.

4. Conception du module Generation

Le module Generation reçoit les données extraites par le Retrieval et les transforme en une réponse claire, structurée et adaptée au profil de l'utilisateur. Il s'appuie sur un modèle de traitement du langage naturel capable de reformuler les informations techniques en langage compréhensible, tout en conservant la précision et la traçabilité. Dans notre système, ce module est paramétré pour inclure, lorsque nécessaire,

les références aux données sources (ex. fichier CSV ou ligne JSON) afin de renforcer la transparence. Il doit également gérer différents niveaux de détail selon le type d'utilisateur et le contexte de la question.

5. **Intégration et orchestration des composants**

Une fois les modules Retrieval et Generation développés, ils sont intégrés dans une architecture orchestrée qui gère le flux complet : réception de la question, analyse NLP, recherche, génération de la réponse et restitution au chatbot. Cette orchestration inclut également la gestion des erreurs (ex. absence de données pertinentes), la mise en cache des réponses fréquentes et la journalisation des interactions pour analyse et amélioration continue. L'intégration doit être fluide afin que l'utilisateur perçoive le chatbot comme un interlocuteur unique et cohérent, et non comme une juxtaposition de modules techniques.

6. **Tests, optimisation et déploiement**

La dernière étape consiste à tester le chatbot dans des conditions réelles, en simulant différents scénarios d'utilisation. Les tests portent sur la pertinence des réponses, la rapidité d'exécution, la robustesse face aux requêtes ambiguës et la capacité à gérer des volumes importants de demandes. Les retours des utilisateurs pilotes permettent d'ajuster les paramètres du Retrieval, d'améliorer la formulation des réponses et d'optimiser les performances globales. Une fois validé, le chatbot est déployé en production, avec un suivi continu pour intégrer les évolutions des règles d'affectation et des besoins utilisateurs.

c. Schéma explicatif du RAG

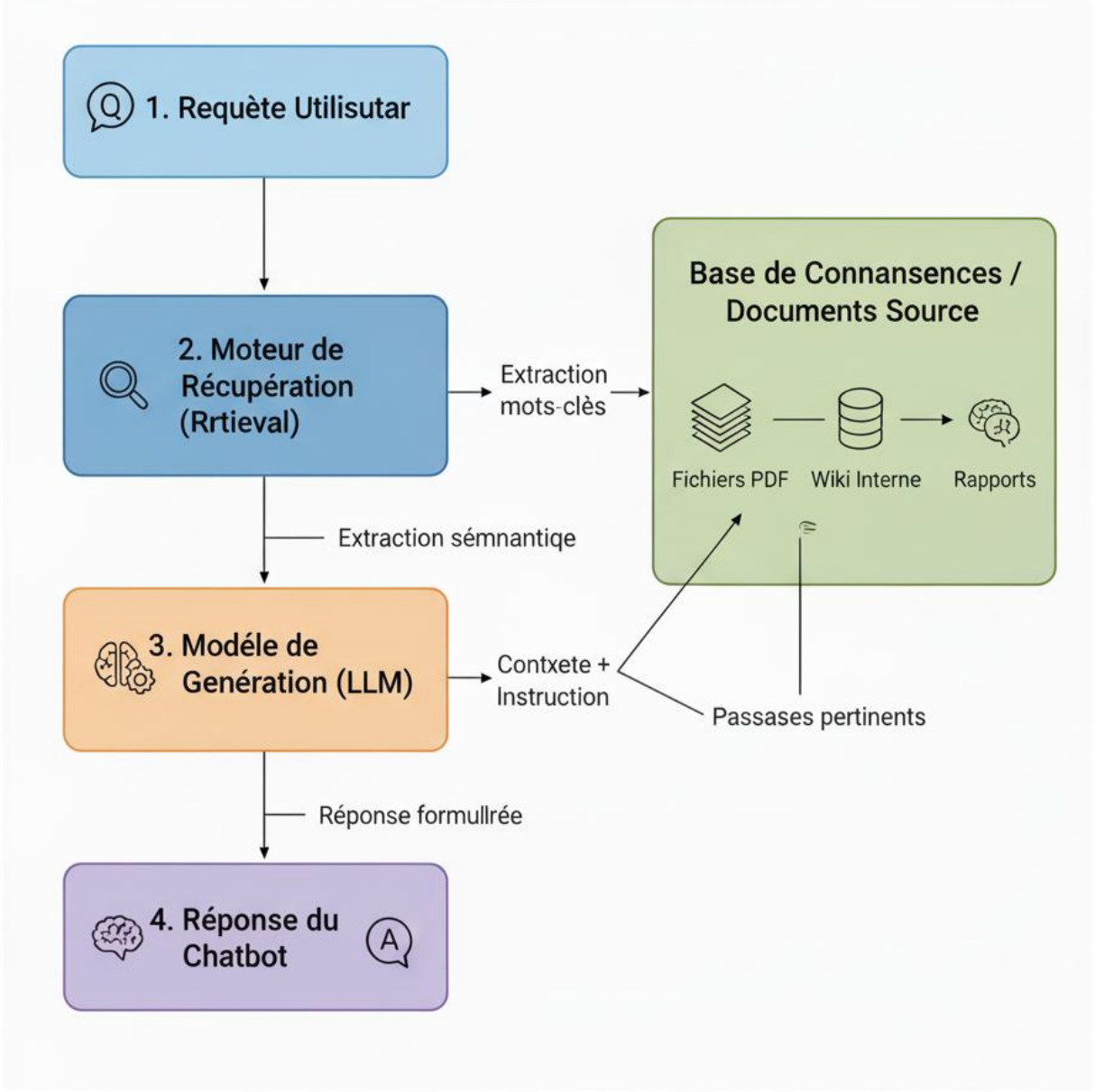


Figure 9: Fonctionnement du RAG dans le chatbot

d. Avantages du RAG dans ce projet

L'intégration d'un module **RAG** dans ce projet apporte une valeur ajoutée majeure en combinant la précision d'une recherche ciblée dans les résultats d'affectation avec la fluidité d'une génération en langage naturel, ce qui permet de fournir aux utilisateurs des explications claires, contextualisées et traçables sur les décisions prises par l'algorithme. Grâce à sa capacité à interroger directement les fichiers finaux (CSV, JSON, TXT) et à exploiter les règles et pondérations appliquées, le RAG garantit une transparence totale et renforce la

confiance des enseignants, des chefs d'établissement et des services administratifs dans le système. Il améliore également l'efficacité opérationnelle en réduisant le temps nécessaire pour obtenir une réponse pertinente, tout en adaptant le niveau de détail au profil et aux besoins de l'interlocuteur. Sa modularité permet d'intégrer facilement de nouvelles sources d'information ou des évolutions réglementaires, assurant ainsi la pérennité et l'évolutivité de la solution. Enfin, en transformant des données techniques complexes en explications pédagogiques et personnalisées, le RAG contribue directement à l'accessibilité et à l'appropriation des résultats par toutes les parties prenantes.

3.6Couche de présentation

La couche de présentation constitue le **point de contact direct** entre le système et l'utilisateur final. Dans le cadre de ce projet, elle se matérialise principalement par l'**interface utilisateur du chatbot**, conçue pour offrir une expérience fluide, intuitive et adaptée aux différents profils (enseignants, chefs d'établissement, agents administratifs). Cette interface joue un rôle clé dans l'accessibilité du système : elle traduit la complexité des traitements internes et des données en interactions simples, compréhensibles et visuellement claires.

3.7Conclusion

Cette architecture modulaire, intégrant un moteur RAG performant, permet de gérer efficacement l'ensemble du processus d'affectation et d'explication. Elle garantit **fiabilité**, **transparence** et **scalabilité**, tout en offrant une expérience utilisateur optimale.

4. Interface et expérience utilisateur

La couche de présentation constitue le **point de contact direct** entre le système et l'utilisateur final. Dans le cadre de ce projet, elle se matérialise principalement par l'**interface utilisateur du chatbot**, conçue pour offrir une expérience fluide, intuitive et adaptée aux différents profils (enseignants, chefs d'établissement, agents administratifs). Cette interface joue un rôle clé dans l'accessibilité du système : elle traduit la complexité des traitements internes et des données en interactions simples, compréhensibles et visuellement claires.

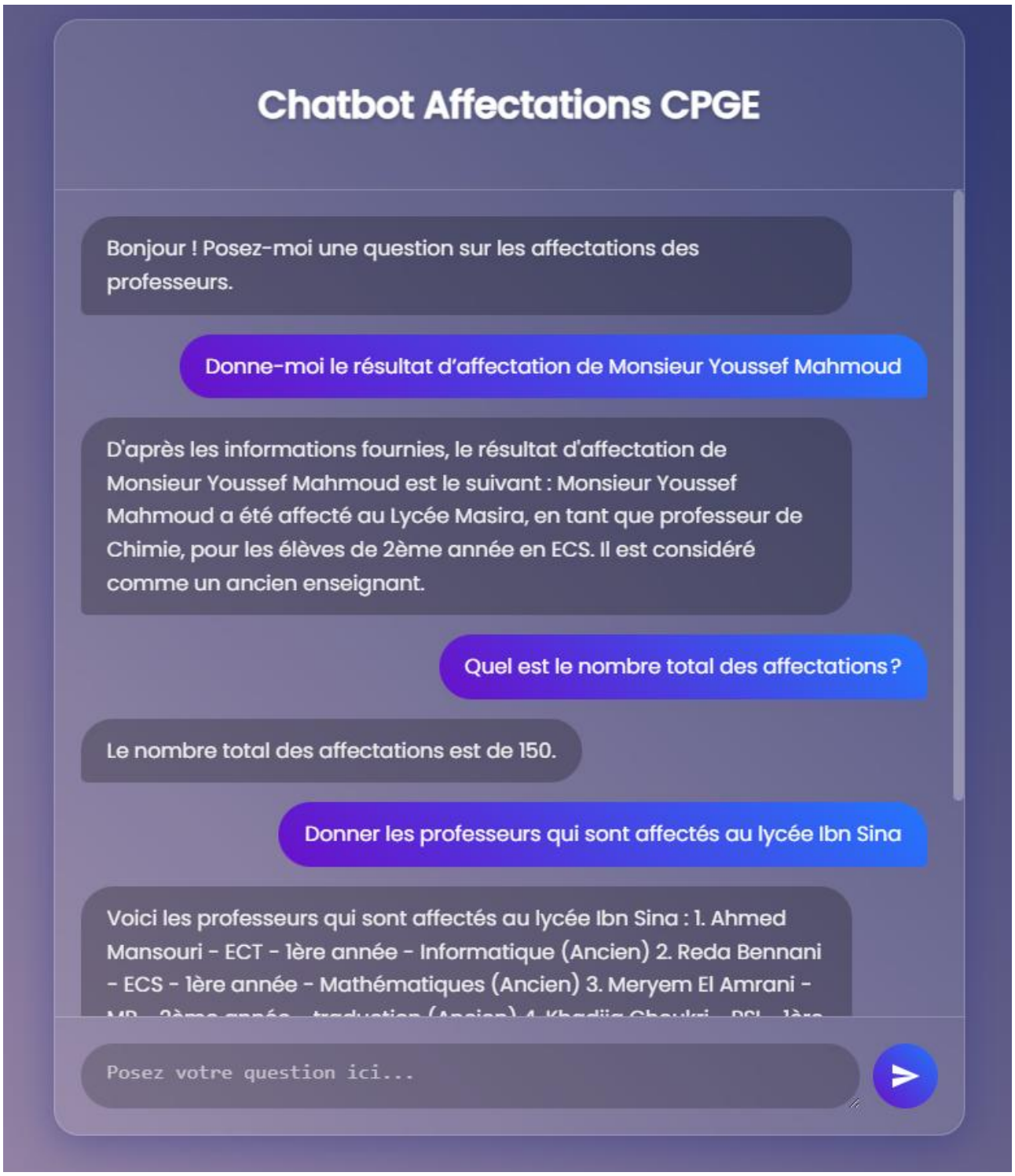


Figure 10: Interface utilisateur du chatbot

5. Cas d'usage représentatifs

Les cas d'usage représentatifs illustrent concrètement la manière dont le système, et en particulier le **chatbot RAG**, répond aux besoins opérationnels des différentes parties prenantes. Ils permettent de démontrer la pertinence de l'architecture mise en place et de valider sa capacité à traiter des situations variées, allant de la simple consultation d'information à l'explication approfondie d'une décision d'affectation.

Cas d'usage 1 – Consultation individuelle d'affectation Un enseignant souhaite savoir à quel poste il a été affecté et comprendre pourquoi ce choix a été retenu. Il interagit avec le chatbot en posant une question en langage naturel : *« Pourquoi ai-je été affecté au lycée X et non à mon premier choix ? »*. Le RAG analyse la requête, identifie l'enseignant, interroge les résultats finaux (CSV/JSON/TXT) et récupère les informations pertinentes : rang du choix, score obtenu, contraintes réglementaires appliquées (ex. rapprochement familial, barème de points). Le module Generation reformule ces données en une réponse claire, expliquant la logique de l'algorithme hongrois et, si nécessaire, les effets des cascades de mutations.

Cas d'usage 2 – Vue globale pour un chef d'établissement Un chef d'établissement souhaite obtenir la liste complète des enseignants affectés à son établissement pour la rentrée. Il demande au chatbot : *« Qui a été affecté dans mon lycée cette année ? »*. Le RAG filtre les résultats par identifiant d'établissement et restitue un tableau synthétique comprenant les noms, disciplines, statuts et éventuelles mentions spécifiques (nouvelle affectation, mutation interne). Cette présentation peut être enrichie de graphiques montrant la répartition par discipline ou par ancienneté.

Cas d'usage 3 – Analyse d'impact d'un changement de règle Un agent administratif souhaite simuler l'effet d'une modification de règle (par exemple, augmentation du poids du critère de rapprochement familial). Il formule la demande au chatbot, qui déclenche un recalcul via la couche de traitement, applique la nouvelle pondération dans la matrice de compatibilité, exécute l'algorithme hongrois, puis renvoie les résultats simulés. Le RAG explique ensuite les différences observées par rapport à l'affectation initiale.

Cas d'usage 5 – Suivi et reporting Un responsable académique souhaite obtenir un rapport synthétique sur le taux de satisfaction des affectations dans son académie. Le chatbot compile les indicateurs calculés par la couche de traitement (taux de premier choix, nombre de postes vacants, score moyen) et les restitue sous forme de tableau et de graphiques, prêts à être intégrés dans un rapport institutionnel.

6. Avantages et valeur ajoutée

L'intégration du système d'affectation enrichi par un **chatbot RAG** apporte une combinaison unique de **performance algorithmique**, **transparence décisionnelle** et **accessibilité utilisateur**. Sur le plan opérationnel, l'utilisation de l'algorithme hongrois, couplé à la gestion automatisée des cascades de mutations, garantit des affectations optimisées respectant à la fois les contraintes réglementaires et les préférences individuelles. Cette optimisation réduit significativement le nombre de postes vacants et améliore le taux de satisfaction des bénéficiaires, ce qui se traduit par une meilleure stabilité organisationnelle.

Sur le plan de la **communication et de la confiance**, le RAG joue un rôle déterminant en rendant les décisions traçables et compréhensibles. Chaque affectation peut être expliquée de manière claire, avec des références directes aux données sources et aux règles appliquées, ce qui renforce la crédibilité du processus auprès des enseignants, des chefs d'établissement et des services administratifs. Cette transparence contribue à réduire les contestations et à fluidifier les échanges entre les parties prenantes.

En termes d'**expérience utilisateur**, la couche de présentation via le chatbot offre une interface intuitive, accessible sur plusieurs plateformes (web, mobile, application dédiée), permettant à tout utilisateur d'obtenir rapidement des réponses personnalisées, sans avoir à naviguer dans des documents techniques complexes. Le système adapte le niveau de détail et le format des réponses au profil de l'interlocuteur, rendant l'information à la fois pertinente et exploitable.

7. Limites et perspectives

Malgré ses performances et sa valeur ajoutée, le système présente certaines **limites** inhérentes à sa conception et à son contexte d'utilisation. Sur le plan technique, la qualité des réponses du chatbot RAG dépend directement de la **qualité, de la complétude et de la fraîcheur des données** disponibles dans la couche de stockage. Des données incomplètes, obsolètes ou mal formatées peuvent entraîner des réponses partielles ou imprécises. De plus, bien que l'algorithme hongrois et la gestion des cascades de mutations soient optimisés pour la majorité des cas, ils peuvent rencontrer des difficultés face à des scénarios atypiques ou à des contraintes exceptionnelles non prévues dans les règles initiales. Sur le plan de l'expérience utilisateur, certaines requêtes formulées de manière ambiguë ou trop générales peuvent nécessiter des reformulations ou des précisions, ce qui peut allonger le temps de réponse. Enfin, la dépendance à des ressources informatiques suffisantes (serveurs, index de recherche, modèles NLP) impose une surveillance continue des performances pour éviter toute dégradation en période de forte charge.

En termes de **perspectives**, plusieurs axes d'évolution sont envisageables pour renforcer la robustesse et l'efficacité du système. L'intégration de **mécanismes d'auto-apprentissage** permettrait au RAG d'améliorer progressivement la pertinence de ses réponses en fonction des interactions passées et des retours utilisateurs. L'ajout de **sources de données complémentaires** (par exemple, des bases réglementaires mises à jour en temps réel ou des historiques d'affectation multi-années) offrirait un contexte plus riche pour l'explication des décisions. Sur le plan algorithmique, l'exploration de méthodes hybrides combinant l'algorithme hongrois avec d'autres approches d'optimisation pourrait améliorer la gestion des cas complexes. Enfin, du côté de l'interface, le développement de **fonctionnalités interactives avancées** (visualisations dynamiques, simulateurs d'affectation, filtres personnalisés) renforcerait l'engagement des utilisateurs et leur compréhension du processus. Ces évolutions, associées à un suivi régulier des besoins institutionnels et réglementaires, garantiront la pérennité et la montée en puissance de la solution dans les années à venir.

8. Conclusion

Le développement de ce système d'affectation, enrichi par un chatbot RAG et structuré autour d'une architecture modulaire, démontre qu'il est possible de concilier rigueur algorithmique, transparence décisionnelle et accessibilité utilisateur. En intégrant un moteur d'optimisation performant (algorithme hongrois), une gestion automatisée des cascades de mutations et un module RAG capable de fournir des explications contextualisées, le projet répond à la fois aux exigences techniques et aux attentes des différentes parties prenantes. L'approche adoptée permet non seulement d'optimiser les affectations en respectant les contraintes réglementaires et les préférences individuelles, mais aussi de rendre le processus compréhensible et vérifiable par tous.

Conclusion générale

Ce projet démontre qu'il est possible de concevoir un système d'affectation à la fois **performant, transparent et centré sur l'utilisateur**, en combinant une architecture algorithmique robuste et une interface conversationnelle intelligente. La première composante, dédiée au **processus d'affectation**, s'appuie sur l'algorithme hongrois et la gestion automatisée des cascades de mutations pour optimiser la répartition des postes en respectant les contraintes réglementaires et les préférences individuelles. Cette approche garantit des résultats équilibrés, réduit les postes vacants et améliore le taux de satisfaction globale, tout en assurant la traçabilité des décisions.

La seconde composante, incarnée par le **chatbot RAG**, transforme ces résultats techniques en **explications claires, contextualisées et personnalisées**. Grâce à la combinaison d'un module de recherche ciblée (Retrieval) et d'un module de génération en langage naturel (Generation), le chatbot rend le processus d'affectation compréhensible pour tous les profils d'utilisateurs — enseignants, chefs d'établissement ou services administratifs — et renforce la confiance dans le système. L'interface multi-plateforme, ergonomique et interactive, facilite l'accès à l'information et permet un dialogue fluide, tout en s'adaptant au contexte et au niveau de détail attendu.

En réunissant **rigueur algorithmique** et **accessibilité conversationnelle**, ce projet apporte une **valeur ajoutée stratégique** : il optimise la prise de décision, fluidifie la communication entre les acteurs et pose les bases d'une solution évolutive, capable d'intégrer de nouvelles règles, données ou fonctionnalités. Il s'agit ainsi d'un outil complet, à la fois moteur d'efficacité opérationnelle et vecteur de transparence, qui peut servir de modèle pour d'autres domaines nécessitant des décisions complexes et justifiables.

Webographie

TEMACONCEPT – Solutions de référence

Hungarian algorithm - Wikipedia

Hungarian Maximum Matching Algorithm | Brilliant Math & Science Wiki

What is RAG (Retrieval Augmented Generation)? | IBM

Annexes

Annexe A : Implémentation du Processus d'Affectation

Ce code Python constitue le cœur du module de traitement. Il utilise la bibliothèque **Numpy** pour les calculs matriciels et **Scipy** pour l'implémentation de l'algorithme hongrois (linear_sum_assignment).

1. Préparation des Données

La première étape consiste à "aplatir" les données hiérarchiques du fichier besoins_etablissements.json. Chaque "besoin" (par exemple, 2 postes de Maths en PSI) est transformé en autant de postes individuels uniques dans la liste `postes_list`.

```python
import numpy as np
from scipy.optimize import linear_sum_assignment
from collections import defaultdict, deque

def optimal_affectation(besoins_etablissements, professeurs_mutation, max_iterations=500):
    # 1. Préparation des données
    postes_list = []
    postes_index = {}
    index = 0

    for etab in besoins_etablissements:
        for filiere, niveaux in etab["filières"].items():
            for niveau, matieres in niveaux.items():
                for matiere, besoin in matieres.items():
                    for _ in range(besoin):
                        poste_id = f"{etab['établissement']}|{filiere}|{niveau}|{matiere}"
                        postes_list.append({
                            "id": poste_id,
                            "etablissement": etab['établissement'],
                            "filiere": filiere,
                            "niveau": niveau,
                            "matiere": matiere,
                            "ville": etab['ville']
                        })
                        postes_index[poste_id] = index
                        index += 1
```

2. Construction de la Matrice de Compatibilité/Poids

C'est l'étape cruciale. Une matrice poids_matrix est créée où les lignes représentent les professeurs et les colonnes les postes.

- **Incompatibilité :** Si un professeur n'est pas compatible (matière, niveau, filière), la case garde une valeur très négative (-1e10), ce qui équivaut à $-\infty$ et garantit que l'algorithme ne choisira jamais cette affectatiom.
- **Calcul du score :** Si l'affectation est possible, un score (poids) est calculé en additionnant les différents bonus décrits dans le rapport : score de base, rapprochement familial , rang du choix , et proximité de la ville.

```
# 2. Matrice de compatibilité et de poids
n_profs = len(professeurs_mutation)
n_postes = len(postes_list)

# CORRECTION : Utiliser une grande valeur négative au lieu de -inf
poids_matrix = np.full((n_profs, n_postes), -1e10)

prof_index = {prof['id']: i for i, prof in enumerate(professeurs_mutation)}

for j, poste in enumerate(postes_list):
    for i, prof in enumerate(professeurs_mutation):
        # Vérifier compatibilité
        compatible = (
            poste['matiere'] == prof['matiere'] and
            poste['niveau'] in prof['niveau'] and
            any(f in prof['filiere'].split('/') for f in [poste['filiere']])
        )

        if not compatible:
            continue  # Garde la valeur -1e10

        # Calcul du score
        score = prof['score'] * 0.5

        if prof['rapprochement'] and prof['etablissement_actuel'] is not None:
            score += 100

        if poste['etablissement'] in prof['choix']:
            position = prof['choix'].index(poste['etablissement']) + 1
            score += [50, 30, 10][min(position-1, 2)]

        if 'ville' in prof and prof['ville'] == poste['ville']:
            score += 20

        if (prof['etablissement_actuel'] and
            prof['etablissement_actuel'] != poste['etablissement']):
            score -= 10

        poids_matrix[i, j] = score
```

3. Exécution de l'Algorithme Hongrois

L'algorithme hongrois est exécuté via linear_sum_assignment. Notez le -poids_matrix : la fonction scipy résout un **problème de minimisation**, alors que nous avons un **problème de maximisation** (maximiser le score). En passant la matrice négative, nous transformons le problème, comme expliqué dans la section 3.3 du rapport.

```python
# 3. Algorithme d'appariement optimal
row_ind, col_ind = linear_sum_assignment(-poids_matrix)

# 4. Affectations initiales
affectations = {}
postes_occupes = set()
profs_affectes = set()

for r, c in zip(row_ind, col_ind):
    if poids_matrix[r, c] > -1e9:  # Seuil pour compatibilité
        prof_id = professeurs_mutation[r]['id']
        poste_id = postes_list[c]['id']
        affectations[prof_id] = poste_id
        postes_occupes.add(poste_id)
        profs_affectes.add(prof_id)
```

4. Gestion des Cascades de Mutations

Cette section implémente la logique décrite en 3.6.

1. On identifie les professeurs qui ont obtenu une mutation et qui **libèrent leur ancien poste**.
2. Ces postes libérés sont ajoutés à une file d'attente (deque).
3. Le système boucle sur cette file : il prend un poste vacant, cherche le meilleur professeur *non encore affecté* pour ce poste.
4. Si un professeur est trouvé, il est affecté, et s'il libère à son tour son propre poste, celui-ci est ajouté à la file, créant ainsi la "cascade".

```python
# 5. Gestion des cascades de mutations
# ------------------------------------
# File pour traiter les mutations en cascade
mutation_queue = deque()

# Identifier les mutations qui libèrent des postes
for prof_id, poste_id in affectations.items():
    prof = next((p for p in professeurs_mutation if p['id'] == prof_id), None)
    nouveau_poste = next((p for p in postes_list if p['id'] == poste_id), None)
    if prof is None or nouveau_poste is None:
        continue
    # Si le professeur change d'établissement, libérer son ancien poste
    if (prof['etablissement_actuel'] and
        prof['etablissement_actuel'] != nouveau_poste['etablissement']):
        # Créer un nouveau poste vacant
        ancien_poste_id = f"{prof['etablissement_actuel']}|{prof['filiere']}|{prof['niveau'][0]}|{prof['matiere']}"
        mutation_queue.append(ancien_poste_id)

# Traitement des cascades
iteration = 0                    (parameter) max_iterations: int
while mutation_queue and iteration < max_iterations:
    nouveau_poste_id = mutation_queue.popleft()

    # Vérifier si le poste est déjà pourvu
    if nouveau_poste_id in postes_occupes:
        continue

    # Trouver le meilleur professeur non affecté pour ce poste
    meilleur_score = -np.inf
    meilleur_prof = None

    for prof in professeurs_mutation:
        if prof['id'] in profs_affectes:
            continue

        # Vérifier compatibilité
        if not (nouveau_poste_id.endswith(prof['matiere']) and
            any(niv in nouveau_poste_id for niv in prof['niveau']) and
            any(fil in nouveau_poste_id for fil in prof['filiere'].split('/'))):
            continue

        # Calculer le score de compatibilité
        score = prof['score'] * 0.5
        if prof['rapprochement'] and prof['etablissement_actuel'] is not None:
            score += 100

        # Vérifier si c'est le meilleur candidat
        if score > meilleur_score:
            meilleur_score = score
            meilleur_prof = prof

    # Affecter le meilleur professeur trouvé
    if meilleur_prof:
        affectations[meilleur_prof['id']] = nouveau_poste_id
        postes_occupes.add(nouveau_poste_id)
        profs_affectes.add(meilleur_prof['id'])

        # Vérifier si cette mutation crée un nouveau poste vacant
        if (meilleur_prof['etablissement_actuel'] and
            not nouveau_poste_id.startswith(meilleur_prof['etablissement_actuel'])):
            nouvel_ancien_poste = f"{meilleur_prof['etablissement_actuel']}|{meilleur_prof['filiere']}|{meilleur_prof['niveau'][0]}
            mutation_queue.append(nouvel_ancien_poste)

    iteration += 1
```

5. Formatage des Résultats et Évaluation

Enfin, le code formate les résultats finaux et identifie les professeurs non affectés et les postes non pourvus. La fonction evaluer_performance est utilisée pour générer les statistiques finales présentées dans le rapport (Figure 6).

```python
    # 6. Formatage des résultats
    # ------------------------
    resultats = []
    for prof_id, poste_id in affectations.items():
        prof = next((p for p in professeurs_mutation if p['id'] == prof_id), None)
        poste = next((p for p in postes_list if p['id'] == poste_id), None)
        if prof is None or poste is None:
            continue
        resultats.append({
            "prof_id": prof_id,
            "prof_nom": prof['nom'],
            "poste_id": poste_id,
            "etablissement": poste['etablissement'],
            "filiere": poste['filiere'],
            "niveau": poste['niveau'],
            "matiere": poste['matiere'],
            "type": "Rapprochement" if prof['rapprochement'] else
                    "Ancien" if prof['etablissement_actuel'] else "Nouveau"
        })

    # 7. Identification des non-affectés
    non_affectes = []
    for prof in professeurs_mutation:
        if prof['id'] not in profs_affectes:
            non_affectes.append({
                "prof_id": prof['id'],
                "prof_nom": prof['nom'],
                "raison": "Aucun poste compatible" if not any(
                    poids_matrix[prof_index[prof['id']], j] > -np.inf
                    for j in range(n_postes)
                ) else "Concurrence",
            })

    postes_non_pourvus = []
    for poste in postes_list:
        if poste['id'] not in postes_occupes:
            postes_non_pourvus.append(poste)

    return {
        "affectations": resultats,
        "non_affectes": non_affectes,
        "postes_non_pourvus": postes_non_pourvus,
        "iterations_cascade": iteration
    }


# Fonction pour calculer les statistiques de performance
def evaluer_performance(affectations):
    stats = {
        'rapprochement': 0,
        'ancien': 0,
        'nouveau': 0,
        'choix_satisfaits': {1: 0, 2: 0, 3: 0},
        'score_total': 0,
        'count': 0
    }

    for aff in affectations:
        stats[aff['type'].lower()] += 1
        stats['count'] += 1
        prof = next((p for p in professeurs_mutation if p['id'] == aff['prof_id']), None)
        if prof is not None:
            stats['score_total'] += prof['score']
            if aff['etablissement'] in prof['choix']:
                position = prof['choix'].index(aff['etablissement']) + 1
                if position <= 3:
                    stats['choix_satisfaits'][position] += 1

    if stats['count'] > 0:
        stats['score_moyen'] = stats['score_total'] / stats['count']
    else:
        stats['score_moyen'] = 0
    return stats
```

Annexe B : Implémentation du Chatbot RAG avec Flask et LangChain

Ce code Python correspond au serveur web (utilisant Flask) qui alimente le chatbot. Il met en œuvre l'architecture RAG en utilisant la bibliothèque LangChain pour orchestrer le processus, FAISS comme base de données vectorielle, et Groq pour l'accès au LLM.

1. Initialisation et Chargement des Données

```python
import os
from flask import Flask, render_template, request
from dotenv import load_dotenv
import numpy as np
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.prompts import PromptTemplate
from langchain_groq import ChatGroq
from langchain.chains import create_retrieval_chain
from langchain.chains.combine_documents import create_stuff_documents_chain
import pandas as pd
import json

# Charger les variables d'environnement (ex: GROQ_API_KEY)
load_dotenv()

app = Flask(__name__, template_folder="templates", static_folder="static")

# --- Initialisation du Chatbot (exécuté une seule fois au démarrage) ---

print("Initialisation du chatbot...")

# 1. Charger les textes depuis le dossier /data
all_texts = []
DATA_FOLDER = "data"
for filename in os.listdir(DATA_FOLDER):
    path = os.path.join(DATA_FOLDER, filename)
    if filename.lower().endswith(".txt"):
        try:
            with open(path, encoding="utf-8") as f:
                all_texts.append(f.read())
        except Exception as e:
            print(f"Erreur lecture TXT {filename}: {e}")
    elif filename.lower().endswith(".csv"):
        try:
            # Lire le CSV en texte (conserve l'en-tête)
            df = pd.read_csv(path, dtype=str, keep_default_na=False)
            text = f"=== FICHIER: {filename} ===\n" + df.to_csv(index=False)
            all_texts.append(text)
        except Exception as e:
            print(f"Erreur lecture CSV {filename}: {e}")
    elif filename.lower().endswith(".json"):
        try:
            with open(path, encoding="utf-8") as f:
                data = json.load(f)
            text = f"=== FICHIER: {filename} ===\n" + json.dumps(data, ensure_ascii=False, indent=2)
            all_texts.append(text)
        except Exception as e:
            print(f"Erreur lecture JSON {filename}: {e}")

```

Au démarrage du serveur, l'application exécute une initialisation unique. Elle lit tous les fichiers de résultats (CSV, JSON, TXT) depuis un dossier data/. Ces fichiers constituent la **base de connaissances** sur laquelle le chatbot fondera ses réponses.

2. Création de la Base de Connaissances (Vector Store)

Une fois les textes chargés, ils sont découpés en petits morceaux ("chunks"). Chaque morceau est ensuite transformé en un vecteur numérique (un "embedding") à l'aide d'un modèle de HuggingFace. Tous ces vecteurs sont stockés dans un index **FAISS**, qui permet une recherche sémantique ultra-rapide. Cet index (retriever) est le composant **"Retrieval" (R)** de RAG.

```
52   # 2. Découper le texte en chunks
53   text_splitter = RecursiveCharacterTextSplitter(
54       chunk_size=800,
55       chunk_overlap=100
56   )
57   chunks = text_splitter.split_text("\n\n".join(all_texts))
58
59   # 3. Générer des embeddings et créer le VectorStore
60   print("Création des embeddings et du vector store...")
61   embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
62   vectorstore = FAISS.from_texts(chunks, embedding_model)
63   retriever = vectorstore.as_retriever(search_kwargs={"k": 5})
64
```

3. Configuration du LLM et de la Chaîne RAG

Ici, nous configurons le composant **"Generation" (G)** de RAG en initialisant le LLM (via **Groq**).

Ensuite, nous créons le **PromptTemplate**. C'est l'instruction fondamentale qui guide le LLM : il *doit* utiliser le contexte (`{context}`) fourni par le `retriever` pour formuler sa réponse et ne doit pas inventer d'informations.

Enfin, `create_retrieval_chain` assemble le tout en un pipeline RAG complet.

```
65   # 4. Définir le modèle LLM via Groq
66   try:
67       # Solution temporaire : passer la clé directement
68       # Remplacez "VOTRE_CLE_API_GROQ" par votre vraie clé
69       api_key = "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"
70
71       llm = ChatGroq(
72           api_key=api_key,
73           model="llama-3.3-70b-versatile", # ou "llama-3.3-70b-versatile"
74           temperature=0.7,
75           max_tokens=1024
76       )
77   except Exception as e:
78       print(f"Erreur lors de l'initialisation du LLM Groq. Vérifiez votre clé API. Erreur: {e}")
79       llm = None
80
```

```python
# 5. Créer la chaîne RAG (Recherche + Génération)
prompt_template = PromptTemplate.from_template(
    """Tu es un assistant spécialisé dans les affectations des professeurs en classes préparatoires (CPGE).
Utilise uniquement les informations présentes dans les documents fournis dans <context> pour répondre.
Ne fabrique aucune information hors des documents. Si l'information manque, indique-le clairement.

Instructions :
- Réponds en français, de façon claire et structurée.
- Si la requête est ambiguë, demande une précision.
si l'information est montionneée dans les documents plus qu'une fois c'est normal,ce n'est pas un error.
ne donne pas une reponse contient des fleches (->).
Tu peus donner des informations supplementaires a propos CPGE si tu les connais et si elles sont en rapport avec la question.
<context>
{context}
</context>

Question : {input}

Réponse :
"""
)

question_answer_chain = create_stuff_documents_chain(llm, prompt_template)
rag_chain = create_retrieval_chain(retriever, question_answer_chain)

print("Chatbot prêt.")
```

4. Serveur Web (Flask) et Inférence en Temps Réel

Cette partie gère l'interface web. La route / affiche la page du chat. La route /get est l'API qui reçoit la question de l'utilisateur (msg).

L'appel rag_chain.invoke({"input": msg}) déclenche tout le processus RAG en temps réel :

1. Le retriever trouve les chunks pertinents pour la question msg.
2. Ces chunks sont insérés dans le {context} du prompt.
3. Le LLM génère la réponse finale en se basant sur ce prompt.

```python
@app.route("/")
def index():
    """Affiche la page principale du chat."""
    return render_template("chat.html")

@app.route("/get", methods=["POST"])
def chat():
    """Reçoit un message de l'utilisateur et retourne la réponse du chatbot."""
    if not llm or not rag_chain:
        return "Erreur: Le chatbot n'est pas correctement initialisé."

    try:
        msg = request.form["msg"]
        print(f"Message reçu: {msg}")

        response = rag_chain.invoke({"input": msg})
        answer = response.get("answer", "Je n'ai pas pu générer de réponse.")

        print(f"Réponse générée: {answer}")
        return str(answer)
    except Exception as e:
        print(f"Erreur lors de l'invocation de la chaîne RAG: {e}")
        return "Désolé, une erreur est survenue lors du traitement de votre demande."

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```