ATTENTION : Il s'agit de la première version (draft) du projet. Elle n'est pas encore complète. La version finalisée vous sera partagée cet après-midi.

Projet IA Générative
Analyse Sémantique pour la Cartographie des Compétences et la Recommandation de Métiers
AISCA : Agent Intelligent Sémantique et Génératif pour la Cartographie des Compétences

Epreuve certifiante RNCP40875 Expert en ingénierie de données (Bloc de compétences à valider : Bloc 2)

Veuillez soumettre votre solution de ce projet sur MOODLE + GIT, en respectant la date limite fixée.
Toute tricherie ne sera pas tolérée et sera sanctionnée.
Vous devez travailler en binôme.

Cadre Théorique et Rôle du projet : Analyse Sémantique, IA Générative, Agent RAG et API Utilisées

Analyse Sémantique : Fondements, Méthodes et Usage dans le Projet

L'analyse sémantique est une composante essentielle du Traitement Automatique du Langage Naturel (NLP). Elle vise à comprendre le sens réel des mots, phrases et paragraphes, et non simplement à compter des occurrences ou analyser des mots-clés.

Contrairement aux approches statistiques classiques, l'analyse sémantique :

- dépasse les représentations numériques simples (TF-IDF, Bag-of-Words),
- capture les relations conceptuelles,
- prend en compte le contexte réel autour des mots,
- permet une compréhension beaucoup plus fine et robuste du langage humain.

Techniques fondamentales utilisées dans ce projet :

- Embeddings de mots et phrases : Word2Vec, GloVe, fastText
- Embeddings contextuels : BERT, RoBERTa, Sentence-BERT (SBERT)
- Mesures de similarité sémantique : Similarité cosinus
- Clustering / mapping sémantique pour structurer des référentiels

Applications professionnelles clés :

- Classification et clustering de textes
- Recherche d'information intelligente
- Analyse documentaire et résumé
- Agents conversationnels et assistants IA
- Recommandation et orientation professionnelle
- Cartographie automatisée des compétences

Rôle dans ce projet

Vous utiliserez l'analyse sémantique pour comparer les réponses libres de l'utilisateur à un référentiel structuré de compétences.
Votre moteur devra déterminer :

- quelles compétences sont couvertes,
- à quel degré (similarité),
- et quels métiers correspondent au profil analysé.

Le cœur de votre système repose donc sur la capacité du modèle (SBERT local) à comprendre et interpréter le texte comme un humain.

IA Générative : Rôle, Contraintes et Usage dans le Projet

L'IA Générative désigne un ensemble de techniques d'intelligence artificielle capables de produire automatiquement du contenu nouveau, qu'il s'agisse de texte, d'images, de code, d'audio ou encore de scénarios et plans d'action. Contrairement aux modèles traditionnels qui se limitent à classer, prédire ou détecter, l'IA générative peut créer, adapter et reformuler du contenu en fonction d'un contexte ou d'un objectif donné.

Parmi les différentes approches de l'IA générative, les modèles de langage avancés, appelés **LLMs (Large Language Models)**, occupent une place centrale pour tout ce qui concerne le texte.
Les LLM sont entraînés sur de vastes corpus de données textuelles et sont capables de comprendre des instructions, de raisonner, d'enrichir des réponses courtes, de générer des synthèses et d'adapter leur style. Ils constituent le moteur principal des applications modernes telles que les agents conversationnels, les systèmes de recommandations intelligentes et les assistants pédagogiques automatisés.

Rôle dans ce projet

Dans le cadre de ce projet AISCA, l'IA Générative, et en particulier les LLMs, est utilisée de façon ciblée pour :

- enrichir le texte utilisateur lorsque celui-ci est trop court ;
- générer un plan de progression personnalisé basé sur les écarts détectés ;
- produire une bio professionnelle synthétique ;
- renforcer la clarté et la pertinence des recommandations métier.

Son usage est contrôlé, limité et optimisé afin de respecter les contraintes du Free Tier et de garantir un fonctionnement fiable, économique et conforme aux meilleures pratiques du domaine.

API recommandée
Vous utiliserez une API GenAI moderne et gratuite, par exemple :

- Google Gemini 2.5 Flash (recommandé pour son coût nul et sa rapidité)
- OpenAI free-tier si disponible
- ou un modèle local via Ollama (option zéro coût)
- ou autre modèle free-tier …

Contraintes obligatoires :

- Appels API strictement limités

- Implémentation d'un caching automatique
- Un seul appel pour la génération du plan de progression
- Un seul appel pour la bio finale

Contrainte	**Signification**	**Pourquoi ?**	**Implication technique**
Appels API strictement limités	N'appeler l'API que quand c'est absolument nécessaire	Respect quota Free Tier + éviter coûts	Minimiser l'usage de la GenAI
Caching automatique	Réutiliser les réponses déjà obtenues	Réduire appels API, accélérer	Cache JSON / dictionnaire / SQLite
Un seul appel pour le plan de progression	Prompt complet ➔ 1 réponse	Optimisation coût & performance	Regrouper tout dans un seul prompt
Un seul appel pour la bio finale	Prompt unique pour la bio	Économie + rapidité	Bio = 1 appel API

Agent RAG (Retrieval-Augmented Generation)

Vous devez comprendre la logique **RAG (Retrieval-Augmented Generation)** car elle structure l'ensemble de l'architecture de votre agent AISCA.
Un agent RAG combine la recherche d'information avec la génération IA, afin d'obtenir des réponses plus fiables, contextualisées et contrôlables qu'une génération brute.

Retrieval (Recherche)
Dans cette première étape, votre système extrait les informations réellement pertinentes avant tout appel à l'IA générative.
Cela inclut :

- l'extraction automatique des compétences pertinentes dans votre référentiel,
- l'identification des blocs faiblement couverts par l'utilisateur,
- l'analyse d'écarts entre le profil utilisateur et un profil métier cible.

Cette phase permet au système de **récupérer uniquement les éléments utiles**, afin d'éviter les hallucinations et de préparer un contexte fiable.

Augmented Context Construction
Une fois la recherche effectuée, vous construisez un **contexte enrichi** à transmettre au LLM.
Cette étape consiste à :

- enrichir le texte utilisateur (si nécessaire) avant l'analyse,
- intégrer dans le prompt les scores sémantiques, les compétences manquantes, les blocs clés ou les profils métier identifiés.

L'objectif est de fournir à la GenAI un **contexte précis, structuré et guidé**, pour obtenir une génération maîtrisée et alignée avec vos données.

Generation (Production)
Enfin, l'IA générative produit un contenu final à partir du contexte enrichi.
Elle génère :

- un **plan de progression personnalisé**,

- un **résumé professionnel / bio**,
- des **suggestions de métiers** plus détaillées et cohérentes.

La génération s'appuie directement sur les éléments récupérés et structurés dans les étapes précédentes, garantissant une production adaptée au profil utilisateur.

Pourquoi un Agent RAG ?

Le choix du modèle RAG n'est pas simplement technologique : il répond à des besoins réels dans le développement d'un outil d'analyse de compétences moderne.

Un agent RAG permet :

- Cohérence : les réponses sont fondées sur un référentiel maîtrisé.
- Fiabilité : réduction des erreurs et des hallucinations de la GenAI.
- Réduction des coûts : limitation des appels API et meilleure optimisation des prompts.
- Contrôle total : la génération reste guidée par vos données, et non produite « en roue libre ».
- Alignement industriel : c'est la méthode utilisée dans les assistants professionnels (HRTech, EdTech, LegalTech, etc.).

Rôle dans ce projet

Votre MVP AISCA : un Mini-Agent RAG

Votre projet AISCA est donc un **mini-agent RAG spécialisé en analyse de compétences**, capable de :

- comprendre les réponses de l'utilisateur,
- se baser sur un référentiel structuré,
- analyser les écarts,
- et produire des recommandations guidées par l'IA générative.

C'est exactement la logique des systèmes intelligents modernes déployés dans l'industrie.

Objectifs Pédagogiques du Projet

En réalisant ce projet complet d'IA Générative et NLP sémantique, vous apprendrez à :

- appliquer le prétraitement de texte et les embeddings sémantiques ;
- distinguer l'analyse numérique (scores bruts) de l'analyse sémantique contextualisée ;
- implémenter un moteur de similarité basé sur SBERT ;
- structurer un référentiel de compétences (format professionnel) ;
- développer une interface web (Streamlit ou autre) ;
- intégrer la GenAI de manière responsable, contrôlée et économique ;
- concevoir un pipeline complet NLP / Scoring / Recommandation / GenAI ;
- travailler en équipe pour produire une solution professionnelle ;
- présenter un prototype en conditions proches du monde réel.

Compétences RNCP visées (RNCP40875 – Bloc 2)

Ce projet IA Générative constitue l'un des projets majeurs de l'année.

Ce projet permet de valider certaines compétences du Bloc 2 : BC2 - Piloter et implémenter des solutions d'IA en s'aidant notamment de l'IA générative du référentiel RNCP40875.

Selon le référentiel, ce bloc couvre notamment :

Compétences principales évaluées

- Collecter, analyser et préparer des données structurées et non structurées
- Concevoir et implémenter des modèles Data Science / NLP / IA
- Évaluer et optimiser les modèles
- Prototyper des solutions IA (API, NLP, RAG, embeddings, GenAI)
- Développer des pipelines data de bout en bout
- Industrialiser une solution (architecture, performance, contraintes coût)
- Documenter et présenter une solution technique complète

Compétences transverses également mobilisées :

- Travail en équipe
- Documentation technique
- Présentation orale
- Respect des exigences fonctionnelles et non fonctionnelles
- Sécurisation et gestion responsable de l'IA générative

Liste Synthétique des Compétences Clés Mobilisées dans Ce Projet

Voici certaines compétences concrètes que vous allez développer durant ce projet (et tout au long de l'année) :

Compétences Data Science / NLP

- Prétraitement textuel avancé
- Construction d'embeddings (SBERT)
- Calcul de similarité cosinus
- Structuration et nettoyage de référentiels

Compétences IA Générative

- Prompt engineering
- Intégration d'une API GenAI (Gemini/OpenAI)
- Génération contextuelle (RAG)
- Mise en place d'un caching local

Compétences Data Engineering

- Capture et stockage structuré des données
- Architecture logique de pipeline IA
- Optimisation des performances et coût
- Versioning du code (Git)

Compétences Software

- Développement d'interface utilisateur (Streamlit)
- Déploiement local d'une application IA
- Visualisation (graphiques, radars, tableaux)

Compétences Professionnelles

- Conception d'un MVP
- Analyse documentaire et justification des choix
- Reporting et documentation technique
- Présentation orale du produit final

Veuillez consulter la grille de notation détaillée afin d'avoir une vision globale de l'ensemble des compétences, critères d'évaluation et modalités de notation.

Cahier des Charges du Projet : Agent Intelligent Sémantique et Génératif pour la Cartographie des Compétences AISCA

ATTENTION : Il s'agit de la première version (draft) du projet. Elle n'est pas encore complète. La version finalisée vous sera partagée cet après-midi.

Description du projet

Ce projet vous offre l'opportunité **d'appliquer concrètement l'analyse sémantique** en développant un outil web qui aide les utilisateurs à évaluer leurs compétences et à explorer des profils de métiers alignés avec celles-ci.

Objectif

Vous allez construire une application web où les utilisateurs remplissent un questionnaire décrivant leurs compétences et expériences sous forme de texte libre.
Ces entrées seront ensuite analysées sémantiquement et comparées à un référentiel de compétences.

Prérequis et exigences principales

- L'analyse doit être contextuelle et sémantique, non purement numérique.
- Les entrées utilisateur doivent être associées aux blocs de compétences à l'aide de techniques telles que :
  - Word embeddings (Word2Vec, GloVe, fastText)
  - Contextual embeddings (BERT, SBERT)
  - Mesures de similarité (cosine similarity, clustering sémantique).
- Un score de couverture sera calculé selon une formule combinant plusieurs blocs de compétences (pondérations, seuils ou règles d'agrégation).
- Le système doit générer un profil de compétences pour l'utilisateur et suggérer des métiers correspondants.

Résultats attendus

- Une interface web fonctionnelle du questionnaire.
- Un moteur d'analyse sémantique comparant les réponses avec le référentiel de compétences.
- Un score de couverture des compétences agrégé.
- Un module de recommandation de profils de métiers pertinents.
- Une visualisation des résultats (ex. graphique radar, barres, cartes de similarité).

Recommandations:

Collecte des réponses (questionnaire)

Vous devez concevoir un **questionnaire web recueillant les réponses via différents types de questions** :

- Échelle de Likert (ex. : "Évaluez votre niveau en Python de 1 à 5").
- Questions ouvertes (texte libre décrivant compétences ou projets).

- Questions guidées (ex. : "Avez-vous déjà utilisé des techniques de tokenization ?").
- Questions à choix multiples (simple ou multiple).
- Cases à cocher pour sélectionner plusieurs compétences.

Le développement de l'interface Web

Pour rendre l'outil plus attrayant et convivial, vous pouvez développer une **interface utilisateur interactive** en utilisant :

- **N'importe quel framework web de votre choix** (par exemple : Flask, Django, Dash, React).
- Ou **Streamlit** (recommandé pour les prototypes rapides).
  Streamlit (https://streamlit.io/) est un framework Python open-source qui permet aux data scientists et aux ingénieurs en IA/ML de créer et de déployer rapidement des applications de données interactives avec seulement quelques lignes de code.
  Il vous permet de construire des applications puissantes et dynamiques en quelques minutes.
  Documentation disponible : https://docs.streamlit.io/

L'interface doit :

- Être intuitive et réactive.
- Permettre aux utilisateurs de naviguer facilement entre différents types de questions.
- Collecter et stocker les réponses dans un format structuré (CSV, JSON ou base de données).
- Transmettre les réponses au module d'analyse sémantique pour traitement.

Attention :
Le questionnaire ne doit pas se limiter à des valeurs numériques — il doit inclure du **texte contextuel et sémantique**.
Vous ne faites pas simplement des calculs numériques ; vous utilisez la similarité sémantique des embeddings de texte pour mesurer dans quelle mesure le profil d'un utilisateur couvre les compétences dans chaque bloc, puis vous agrégez ces résultats afin d'obtenir un score de couverture significatif et une recommandation de poste pertinente.

Exemple de mécanisme de notation basé sur la similarité sémantique et les blocs de compétences :

Vous trouverez ci-dessous un **exemple concret** que vous pouvez considérer comme source d'inspiration.
N'oubliez pas que l'analyse doit rester **sémantique et contextuelle** (et non simplement basée sur des comptages numériques).

Étape 1 – Définir les blocs de compétences
Supposons que le cadre comporte 3 blocs de compétences :

1. **Data Analysis** – {"data cleaning…", "data visualization", "statistics", "Python"}

2. **Machine Learning** – {"classification", "regression", "neural networks", "model evaluation"}
3. **NLP** – {"tokenization", "word embeddings", "transformers", "semantic analysis"}

Étape 2 – Collecter les entrées utilisateur (questionnaire)
Un utilisateur répond au questionnaire en texte libre :

- "I have experience in cleaning data with Python and making dashboards."
- "I studied regression models and deep learning."

Étape 3 – Appliquer la correspondance sémantique (Semantic Matching)

- Convertir les termes du référentiel et les réponses de l'utilisateur en **embeddings** (par ex. **SBERT**).
- Calculer la **similarité cosinus** entre les embeddings des textes utilisateur et chaque phrase de compétence.
- Exemple de scores de similarité que vous pouvez obtenir (les valeurs sont données à titre illustratif) :
  - Entrée utilisateur vs. Bloc **Data Analysis** → 0.85 (high match)
  - Entrée utilisateur vs. Bloc **Machine Learning** → 0.78 (medium-high match)
  - Entrée utilisateur vs. Bloc **NLP** → 0.40 (low match)

Étape 4 – Calcul du score
Définir une formule simple pour combiner les scores de chaque bloc en un score global :

$$\boldsymbol{Coverage\ score} = \frac{\sum_{i=1}^{n} \mathrm{Wi} * \mathrm{Si}}{\sum_{i=1}^{n} \mathrm{Wi}}$$

où
S_i = score de similarité pour le bloc i
W_i = poids pour le bloc i (valeur par défaut = 1 si tous les blocs ont la même importance, ou ajusté si certaines compétences sont prioritaires)

Exemple **:**

- Data Analysis: S1=0.85, W1=1
- Machine Learning: S2=0.78, W2=1
- NLP: S3=0.40, W3=1

$$\boldsymbol{Coverage\ score} = \frac{0.85 + 0.78 + 0.403}{3} = 0.68$$

Étape 5 – Recommandation de métier (de profil de poste)

- Si **Coverage Score ≥ 0.7** → "Data Scientist"
- Si **Coverage Score between 0.5 and 0.7** → "ML Engineer"
- Si **Coverage Score < 0.5** → "Entry-level Analyst"

Comment former vos données de référence :

1. **Competency Blocks / Blocs de compétences**
   - Chaque bloc est une catégorie de compétences liées.
   - À l'intérieur de chaque bloc, listez plusieurs compétences ou savoirs-faire exprimés sous forme de courtes phrases.

- Exemples de blocs : Data Analysis, Machine Learning, NLP, Project Management.

2. **Job Profiles / Métiers ou Profils de postes**
   - Chaque profil de poste est lié à un ou plusieurs blocs de compétences et/ou à plusieurs compétences individuelles.
   - Un profil de poste est représenté par l'ensemble des compétences requises.
   - Exemple: Data Scientist requires coverage in Data Analysis + Machine Learning + NLP.
3. **Pourquoi ce format ?**
   - Il garantit que le système peut comparer les entrées utilisateur à des cibles sémantiques bien structurées.
   - Il permet au mécanisme de scoring de calculer la couverture (dans quelle mesure chaque bloc est satisfait).
   - Il rend le cadre facile à étendre ou à mettre à jour avec de nouvelles compétences ou de nouveaux profils de poste à l'avenir.

Voici un exemple :

Dans les cadres réels, **une seule compétence** (par exemple, *programmation en Python*) peut être **pertinente pour plusieurs profils de poste** (*Data Analyst*, *ML Engineer*, *Data Scientist*, *NLP Engineer*).

Reference Data Structure:

CompetencyID	Competency	BlockID	BlockName
C01	data cleaning	1	Data Analysis
C02	data visualization	1	Data Analysis
C03	Python programming	1	Data Analysis
C04	regression	2	Machine Learning
C05	neural networks	2	Machine Learning
C06	tokenization	3	NLP
C07	transformers	3	NLP

JobID	JobTitle	RequiredCompetencies
J01	Data Analyst	C01; C02; C03
J02	ML Engineer	C03; C04; C05
J03	Data Scientist	C01; C02; C03; C04; C05; C06; C07
J04	NLP Engineer	C03; C05; C06; C07

Par exemple, Python programming (C03) fait partie de :

- Data Analyst (J01)
- ML Engineer (J02)
- Data Scientist (J03)
- NLP Engineer (J04)

Ainsi, supposons que l'utilisateur mentionne fortement la programmation en Python : dans ce cas, le score de couverture augmentera pour tous les profils de poste qui incluent cette compétence.

Vous pouvez utiliser des référentiels standards de compétences et de savoir-faire pour créer votre jeu de données de référence, tels que :

https://www.data.gouv.fr/datasets/repertoire-operationnel-des-metiers-et-des-emplois-rome/
https://assets.ide-conseil-webmarketing.fr/wp-content/uploads/2019/05/European-e-Competence-Framework-3.0_FR.pdf

Exemple de pseudocode Python : Score sémantique avec SBERT

Voici un exemple de pseudocode en Python montrant comment vous pouvez implémenter un mécanisme de score sémantique en utilisant les embeddings SBERT, la similarité cosinus, et une formule de calcul du score.

Utilisez-le uniquement comme source d'inspiration — personnalisez votre propre solution.

```
# === Step 1: Import libraries ===
from sentence_transformers import SentenceTransformer, util
import numpy as np

# === Step 2: Define competency framework (blocks) ===
competency_blocks = {
    "Data Analysis": ["data cleaning", "data visualization", "statistics", "Python"],
    "Machine Learning": ["classification", "regression", "neural networks", "model evaluation"],
    "NLP": ["tokenization", "word embeddings", "transformers", "semantic analysis"]
}

# === Step 3: Collect user inputs (from questionnaire) ===
user_inputs = [
    "I have experience cleaning data with Python and building dashboards.",
    "I studied regression models and deep learning."
]

# === Step 4: Load SBERT model for embeddings ===
model = SentenceTransformer("all-MiniLM-L6-v2")

# Encode user inputs
user_embeddings = model.encode(user_inputs, convert_to_tensor=True)

# === Step 5: Calculate semantic similarity for each block ===
block_scores = {}

for block, competencies in competency_blocks.items():
    # Encode competency block phrases
    block_embeddings = model.encode(competencies, convert_to_tensor=True)

    # Compare each user input to competencies using cosine similarity
    similarities = util.cos_sim(user_embeddings, block_embeddings)

    # Take max similarity per user input and average across inputs
```

```
        max_similarities = [float(sim.max()) for sim in similarities]
        block_score = np.mean(max_similarities)

        block_scores[block] = block_score

# === Step 6: Weighted scoring formula (all equal weight here) ===
final_score = np.mean(list(block_scores.values()))

# === Step 7: Recommend job profile based on thresholds ===
if final_score >= 0.7:
    recommendation = "Data Scientist"
elif final_score >= 0.5:
    recommendation = "ML Engineer"
else:
    recommendation = "Entry-level Analyst"

# === Step 8: Display results ===
print("Block scores:", block_scores)
print("Final coverage score:", round(final_score, 2))
print("Recommended job profile:", recommendation)
```

Exigences Fonctionnelles Détaillées (EF)

EF1 : Acquisition de la Donnée

- **EF1.1 : Questionnaire Hybride** : Le questionnaire doit combiner des questions numériques (échelle de Likert pour l'auto-déclaration de niveau) **ET** des questions ouvertes pour la collecte de preuves contextuelles (Ex: "Décrivez un projet où vous avez utilisé Python pour l'analyse de données").
- **EF1.2 : Structuration** : Les réponses collectées doivent être stockées dans un format structuré (CSV, JSON, ou base de données locale) pour le traitement PNL.

EF2 : Moteur PNL Sémantique (Cœur du Projet - Coût Zéro)

- **EF2.1 : Référentiel de Compétences** : **Vous devez** créer un référentiel des blocs de compétences (ex. *Data Viz*, *NLP*, *ML*) et des profils de postes associés, inspiré de standards (ex. ROME).
- **EF2.2 : Modélisation Sémantique** : **Vous devez** utiliser un modèle d'embeddings Open-Source et local (ex. **SBERT** via la librairie SentenceTransformer) pour transformer les entrées utilisateur et les phrases du référentiel en vecteurs.
- **EF2.3 : Mesure de Similarité** : Le moteur doit calculer la **Similarité Cosinus** entre les vecteurs utilisateur et les vecteurs de compétences du référentiel.

EF3 : Système de Scoring et Recommandation

- EF3.1 : Formule de Score : Vous devez implémenter une formule de score pondérée pour calculer le Score de Couverture Sémantique.

- **EF3.2 : Recommandation** : Le système doit proposer les 3 profils de postes pour lesquels l'utilisateur obtient le score de couverture le plus élevé.

EF4 : Augmentation par GenAI (Stratégique et Limitée)

- **EF4.1 : Augmentation de l'Entrée (Pre-Processing)** : **Vous devez** développer une fonction qui utilise la GenAI pour enrichir les phrases d'entrée utilisateur jugées trop courtes (moins de 5 mots), en ajoutant du contexte technique pour améliorer la précision des embeddings PNL locaux.
    - Exigence d'usage : Appel **limité à un usage conditionnel** (si la phrase est trop courte).
- **EF4.2 : Génération du Plan de Progression** : Le système doit générer un texte personnalisé (par GenAI) qui identifie les **compétences prioritaires** à développer (celles ayant les plus faibles scores de similarité) et propose un **chemin d'apprentissage** précis.
    - Exigence d'usage : Un **seul appel API** pour la sortie finale.
- **EF4.3 : Synthèse de Profil** : Générer une courte biographie professionnelle accrocheuse (style Executive Summary).

Livrables :

- **Vous devez soumettre les livrables sur MOODLE + Git/GitHub** :
    - **Code source de la solution fonctionnelle + documentation (Note de groupe)**
    - **Support de Présentation du projet**
- Présentation et démonstration en classe **lors de la dernière séance du cours** : Tous les membres du groupe doivent participer ; **Evaluation/Note individuelle**)
- Grille d'évaluation est partagée dans un document séparé.

GOOD LUCK!