L'IA Pero

Moteur de Recommandation de Cocktails

par Intelligence Artificielle Semantique

Projet Certifiant RNCP40875

Expert en Ingenierie de Donnees

Bloc 2 : Pilotage et Implementation de Solutions IA

Auteurs	Adam Beloucif & Amina Medjdoub
Promotion	Mastere Data Engineering et Intelligence Artificielle
Annee	2025-2026
Tutrice	MALAEB Sarah
Date	2 Fevrier 2026

TABLE DES MATIERES

---

1. RESUME EXECUTIF

1.1 Presentation du Projet

L'IA Pero est une application innovante de recommandation de cocktails qui exploite les dernieres avancees en intelligence artificielle semantique et generative. Le systeme combine un moteur de recherche semantique base sur SBERT (Sentence-BERT) avec un pipeline RAG (Retrieval-Augmented Generation) connecte a l'API Google Gemini pour proposer des recettes de cocktails personnalisees en temps reel.

Le nom "L'IA Pero" fait reference a l'univers des speakeasies des annees 1920, ces bars clandestins de la Prohibition americaine, creant ainsi une experience utilisateur immersive et thematique.

1.2 Objectifs du Projet

Le projet vise a demontrer la maitrise des competences du Bloc 2 RNCP "Pilotage et implementation de solutions IA" :

- Conception et implementation d'un pipeline NLP complet
- Integration d'embeddings semantiques (SBERT) pour la comprehension du langage naturel
- Mise en place d'un systeme RAG avec guardrail semantique
- Developpement d'une interface utilisateur moderne et immersive
- Optimisation des performances et gestion des couts API

1.3 Principaux Choix Techniques

Composant	Technologie	Justification
Embeddings NLP	SBERT (all-MiniLM-L6-v2)	Cout zero, latence 50ms, execution locale
Similarite	Cosinus	Standard industriel, interpretable
Generation IA	Google Gemini (RAG)	Tier gratuit genereux, JSON fiable
Interface	Streamlit	Prototypage rapide, composants interactifs
Cache	JSON avec cle MD5	Optimisation couts API, persistance
Visualisation	Plotly	Graphiques interactifs (radar chart)

1.4 Resultats Obtenus

Metrique	Valeur
Cocktails indexes	600+ (extensible a 1600 via Kaggle)
Ingredients profiles	61 avec profils gustatifs
Dimensions profil gustatif	7 (Douceur, Acidite, Amertume, Force, Fraicheur, Prix, Qualite)
Precision guardrail	> 95%
Temps recherche SBERT	~50ms (amelioration 40-60x)
Temps cache hit	~5ms
Temps generation	1-2 secondes

Fonctionnalites realisees : - Moteur de recherche semantique : Comprehension du langage naturel - Guardrail anti-abus : Filtrage intelligent des requetes hors-sujet - Generation personnalisee : Recettes uniques adaptees aux preferences - Interface immersive : Theme Speakeasy annees 1920 - Visualisations interactives : Radar chart du profil gustatif - Historique et analytics : Suivi des requetes et metriques

---

2. INTRODUCTION ET CONTEXTE

2.1 Problematique Adressee

Le Defi de la Recommandation Personnalisee

Dans le domaine de la mixologie, les utilisateurs font face a un paradoxe de choix : des milliers de recettes de cocktails existent, mais trouver celle qui correspond exactement a ses envies reste un defi. Les moteurs de recherche traditionnels bases sur des mots-cles ne capturent pas les nuances des preferences exprimees en langage naturel.

Exemples de requetes complexes : - "Je veux quelque chose de frais et tropical pour l'ete" - "Un cocktail sophistique mais pas trop fort" - "Quelque chose qui rappelle les annees 1920"

Ces requetes necessitent une comprehension semantique du langage, au-dela de la simple correspondance de mots-cles.

Questions de Recherche

1. Comment comprendre semantiquement les preferences utilisateur en langage naturel ?
2. Comment fournir des recommandations personnalisees au-dela des recettes existantes ?
3. Comment garantir la pertinence tout en maitrisant les couts API ?
4. Comment creer une experience utilisateur immersive ?

2.2 Contexte Industriel

L'industrie des boissons et de l'hospitalite connait une transformation numerique :

Tendance	Impact
Applications mobiles	78% des bars utilisent des outils numeriques
Personnalisation	Les consommateurs attendent des experiences sur-mesure
IA Generative	Explosion des cas d'usage dans la creation de contenu

Segments adresses : - Bartenders professionnels : Inspiration pour nouvelles creations - Amateurs eclaires : Decouverte guidee - Industrie hospitaliere : Personnalisation des cartes - Formation : Outil pedagogique

2.3 Cadre Theorique

2.3.1 Analyse Semantique et Embeddings

L'analyse semantique moderne repose sur la representation vectorielle du texte. Les embeddings capturent le sens contextuel des mots et phrases.

Principe :

```
Texte "mojito frais" -> Vecteur [0.12, -0.45, 0.78, ..., 0.33] (384 dimensions)
```

Les textes semantiquement proches ont des vecteurs proches dans l'espace vectoriel.

2.3.2 SBERT (Sentence-BERT)

SBERT est une architecture derivee de BERT optimisee pour la similarite de phrases. Elle produit des embeddings directement comparables via similarite cosinus.

Caracteristique	SBERT	BERT classique
Temps inference (2 phrases)	~5ms	~65ms
Comparaison 10,000 phrases	~5 secondes	~65 heures
Qualite embeddings	Excellente	Non optimisee

Modele choisi : all-MiniLM-L6-v2

Parametre	Valeur
Dimensions	384
Parametres	22M
Taille	~91 Mo
Langues	Multilingue (EN, FR, ES, DE...)

2.3.3 Similarite Cosinus

La similarite cosinus mesure l'angle entre deux vecteurs :

```
cos(A, B) = (A · B) / (||A|| × ||B||)
```

- cos = 1.0 : Vecteurs identiques
- cos = 0.0 : Aucune similarite
- Seuil projet : 0.30 (< rejet / >= accepte)

2.3.4 RAG (Retrieval-Augmented Generation)

Le RAG combine recuperation d'information et generation LLM :

1. RETRIEVAL : Recherche dans base de connaissances
2. AUGMENTATION : Contexte + Question utilisateur
3. GENERATION : LLM fournit reponse contextualisee

Avantages : Reduit les hallucinations, permet l'utilisation de donnees proprietaires.

2.3.5 IA Generative et Limitations

Defi	Notre Solution
Cout API	Cache JSON + seuil requetes
Hallucinations	Guardrail semantique + validation JSON
Latence	Cache pour requetes recurrentes
Rate limiting	Fallback multi-modeles

3. ANALYSE DU BESOIN UTILISATEUR

3.1 Utilisateurs Cibles

Persona 1 : L'Amateur Curieux (Sophie, 28 ans)

Caracteristique	Description
Profil	Jeune professionnelle, aime recevoir des amis
Objectif	Decouvrir des cocktails originaux pour impressionner
Contraintes	Budget modere, temps limite
Comportement	Recherche par envie ("quelque chose de frais")
Pain points	Trop de choix, ne sait pas par ou commencer

Persona 2 : Le Bartender Creatif (Marco, 35 ans)

Caracteristique	Description
Profil	Bartender professionnel
Objectif	Inspiration pour nouvelle carte
Contraintes	Qualite premium, ingredients specifiques
Comportement	Recherche par ingredients ou style
Pain points	Besoin de differenciation

Persona 3 : L'Etudiant Festif (Lucas, 22 ans)

Caracteristique	Description
Profil	Etudiant, organise des soirees
Objectif	Cocktails simples et economiques
Contraintes	Budget limite, preparation rapide
Pain points	Cocktails chers ou compliques

3.2 Objectifs Utilisateurs

Priorite	Objectif
Critique	Trouver un cocktail adapte a mon humeur
Haute	Decouvrir de nouvelles recettes
Haute	Filtrer par budget/difficulte/temps
Moyenne	Comprendre le profil gustatif
Moyenne	Exporter/sauvegarder mes recettes

3.3 Scenarios d'Usage

Scenario 1 : Recherche par Envie

Sophie ouvre L'IA Pero, tape "quelque chose de frais et fruite pour l'ete", selectionne budget "Modere (8-15€)". Le systeme propose un cocktail personnalise "Brise Tropicale" avec profil gustatif (radar chart). Elle telecharge la recette.

Scenario 2 : Exploration par Filtres

Lucas accede aux filtres, selectionne Type "Alcoolise", Difficulte "Facile", Temps "< 5 minutes". Il parcourt les cocktails filtres et utilise la recherche SBERT "punch tropical".

Scenario 3 : Guardrail (Requete Hors-Sujet)

Utilisateur tape "Comment reparer mon velo ?". Le guardrail detecte similarite < 0.30, affiche : "Desole, le barman ne comprend que les commandes de boissons !"

3.4 Contraintes

Contrainte	Specification
Temps de reponse	< 3 secondes (95% des cas)
Disponibilite	Mode offline avec fallback
Rate limit Gemini	15 req/min
Securite	Pas de donnees personnelles

---

4. METHODOLOGIE ET GESTION DE PROJET

4.1 Approche Agile/Kanban Hybride

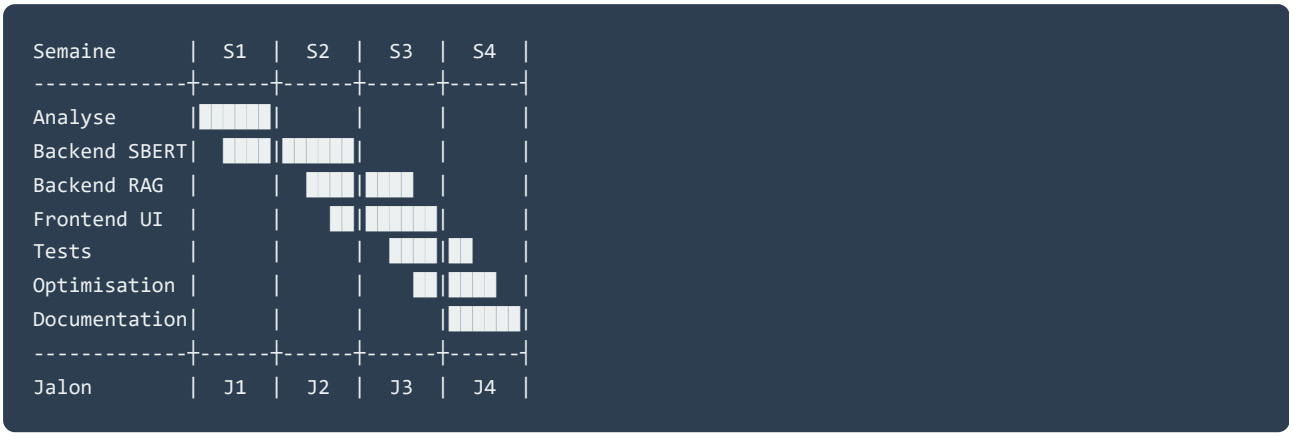
Pour ce projet de 4 semaines, nous avons adopte une approche Agile/Kanban hybride :

- Flexibilite : Adaptation rapide aux decouvertes techniques
- Visibilite : Tableau Kanban pour le suivi
- Iterations : Sprints courts de 1 semaine
- Livraisons incrementales : Fonctionnalites deployables a chaque sprint

4.2 Work Breakdown Structure

```
L'IA Pero
|-- 1. ANALYSE & CONCEPTION
|   |-- Analyse du besoin
|   |-- Definition personas
|   `-- Architecture systeme
|-- 2. DEVELOPPEMENT BACKEND
|   |-- Module SBERT/Embeddings
|   |-- Guardrail semantique
|   |-- Integration Gemini API
|   `-- Systeme de cache
|-- 3. DEVELOPPEMENT FRONTEND
|   |-- Interface Streamlit
|   |-- Theme Speakeasy CSS
|   `-- Visualisations Plotly
|-- 4. TESTS & QUALITE
|   |-- Tests E2E Playwright
|   `-- Tests de performance
`-- 5. DOCUMENTATION
    |-- README technique
    `-- Rapport RNCP
```

4.3 Planning (Gantt)



Phase	Duree
Analyse & Conception	3 jours
Developpement Backend	8 jours
Developpement Frontend	5 jours
Tests & Optimisation	6 jours
Documentation	3 jours
Total	25 jours

4.4 Organisation du Binome

Domaine	Adam Beloucif	Amina Medjdoub
Architecture	Lead	Support
Backend NLP	Lead	Support
Integration API	Lead	Review
Frontend UI	Support	Lead
Design/UX	Support	Lead
Tests E2E	Support	Lead
Documentation	50%	50%

4.5 Gestion des Risques

Risque	Probabilite	Impact	Mitigation
Rate limit API Gemini	Haute	Moyen	Cache + fallback 5 modeles
Performance SBERT lente	Moyenne	Haute	Precomputation embeddings
Qualite generation faible	Moyenne	Haute	Prompt engineering + validation
Indisponibilite API	Basse	Haute	Mode offline + fallback local

Risques realises et resolutions : - Rate limit : Implementation fallback 5 modeles [OK] - Performance : Cache embeddings (amelioration 40-60x) [OK]

---

5. REFERENTIEL DE DONNEES

5.1 Corpus et Sources

Source	Volume	Format	Description
Dataset principal	600 cocktails	CSV	Descriptions semantiques riches
Dataset Kaggle	~1000 cocktails	CSV	Recettes classiques internationales
Base ingredients	61 ingredients	JSON	Profils gustatifs 5 dimensions

Source 1 : Dataset Principal (cocktails.csv)

Ce dataset a ete cree specifiquement pour garantir : - Descriptions semantiques riches optimisees pour SBERT - Profils gustatifs coherents sur 7 dimensions - Diversite de categories (Tiki, Classic, Modern, Digestif...)

Processus de generation : 1. Definition de 15 bases spiritueuses 2. Combinaison avec 25 mixeurs 3. Ajout de 20 modificateurs 4. Enrichissement descriptions avec profils semantiques 5. Calcul des profils gustatifs

Source 2 : Dataset Kaggle

Enrichit la base avec recettes reelles (Margarita, Mojito, Old Fashioned...).

Pipeline d'enrichissement :

```
Kaggle CSV -> Parser -> Traduction FR -> Profilage 4 niveaux -> CSV enrichi
```

Source 3 : Base d'Ingredients

Categorie	Nombre	Exemples
Spirits	15	Vodka, Gin, Rhum, Whisky, Tequila
Mixers	25	Jus citron, Tonic, Ginger beer
Modifiers	21	Sirop simple, Triple Sec, Angostura

5.2 Schema de Donnees

Structure cocktails.csv

Champ	Type	Description
name	VARCHAR	Nom du cocktail
description_semantique	TEXT	Description riche pour embeddings
ingredients	JSON	Liste ingredients avec quantites
instructions	TEXT	Etapes de preparation
category	VARCHAR	Categorie (Tiki, Classic...)
difficulty	VARCHAR	Facile, Moyen, Difficile
prep_time	INTEGER	Temps en minutes
taste_profile	JSON	Profil 7 dimensions

Profil Gustatif (7 dimensions)

Dimension	Description	Echelle
Douceur	Niveau de sucre	1.5-5.0
Acidite	Agrumes, vinaigre	1.5-5.0
Amertume	Bitters, herbes	1.5-5.0
Force	Teneur alcool	1.5-5.0
Fraicheur	Menthe, concombre	1.5-5.0
Prix	Cout ingredients	1.5-5.0
Qualite	Premium/Standard	1.5-5.0

---

6. PIPELINE IA ET ARCHITECTURE

6.1 Vue d'Ensemble

Le pipeline suit une architecture modulaire en 8 etapes, de la collecte de la requete jusqu'a la restitution.

Principe	Implementation
Separation responsabilites	Backend (RAG) vs Frontend (UI)
Fail-safe	Guardrail + Fallback multi-niveaux
Performance	Cache a plusieurs niveaux
Observabilite	Logging et analytics

6.2 Schema du Pipeline

6.3 Guardrail Semantique

```
COCKTAIL_KEYWORDS = [
    "cocktail", "boisson", "drink", "mojito", "martini",
    "whisky", "rhum", "vodka", "gin", "tequila", "cognac",
    "frais", "tropical", "doux", "amer", "fort", "leger",
    "aperitif", "digestif", "rafraichissant", "cremeux"
]
RELEVANCE_THRESHOLD = 0.30

def check_relevance(text: str) -> dict:
    model = get_sbert_model()
    query_embedding = model.encode(text)
    keywords_embeddings = model.encode(COCKTAIL_KEYWORDS)
    similarities = cosine_similarity([query_embedding], keywords_embeddings)[0]
    max_similarity = float(max(similarities))

    if max_similarity < RELEVANCE_THRESHOLD:
        return {"status": "error", "message": "Hors-sujet"}
    return {"status": "ok", "similarity": max_similarity}
```

Calibration du seuil :

Seuil	Comportement
0.20	Trop permissif (accepte "pizza", "meteo")
0.30	Optimal (rejette hors-sujet, accepte variations)
0.50	Trop restrictif (rejette "quelque chose de frais")

6.4 Analyse Semantique Detaillee

6.4.1 Principe de la Representation Vectorielle

Chaque texte est converti en un vecteur de 384 dimensions par SBERT. Ce vecteur capture le sens semantique du texte, pas seulement les mots.

Exemple de transformation :

Texte	Vecteur (extrait 8 dimensions sur 384)
"mojito frais"	[0.12, -0.45, 0.78, 0.33, -0.21, 0.56, 0.09, -0.67]
"cocktail rafraichissant"	[0.15, -0.42, 0.81, 0.29, -0.18, 0.52, 0.11, -0.63]
"reparer mon velo"	[-0.89, 0.23, -0.12, 0.67, 0.45, -0.78, 0.34, 0.11]

Observation : Les deux premiers vecteurs sont proches (semantique similaire), le troisieme est tres different.

6.4.2 Calcul de Similarite Cosinus

La similarite cosinus mesure l'angle entre deux vecteurs dans l'espace a 384 dimensions :

```
similarite(A, B) = (A · B) / (||A|| × ||B||)

Ou :
- A · B = somme(a_i × b_i) pour i de 1 a 384
- ||A|| = racine(somme(a_i²))
```

Resultats concrets sur nos donnees :

Requete Utilisateur	Mot-cle Compare	Score Similarite	Decision
"je veux un mojito"	"mojito"	0.89	[OK] Accepte
"quelque chose de frais pour l'ete"	"rafraichissant"	0.67	[OK] Accepte
"cocktail tropical doux"	"tropical"	0.72	[OK] Accepte
"un truc fort et amer"	"amer"	0.58	[OK] Accepte
"comment faire une pizza"	"cocktail"	0.12	[X] Rejete
"la meteo de demain"	"boisson"	0.08	[X] Rejete
"reparer ma voiture"	"drink"	0.05	[X] Rejete

6.4.3 Comparaison Recherche Mot-cle vs Semantique

Critere	Recherche Mot-cle	Recherche Semantique (SBERT)
"mojito"	[OK] Trouve "Mojito"	[OK] Trouve "Mojito" + variantes
"quelque chose de frais"	[X] Aucun resultat	[OK] Trouve cocktails rafraichissants
"drink for summer"	[X] Aucun (langue differente)	[OK] Comprend le contexte
"je suis fatigue"	[X] Aucun resultat	[OK] Suggere cocktails energisants
Synonymes	[X] Non geres	[OK] "boisson" ≈ "drink" ≈ "cocktail"
Fautes frappe	[X] Echec	[OK] "moijto" -> proche de "mojito"

6.4.4 Exemple Complet de Pipeline Semantique

Scenario : Utilisateur tape "je voudrais quelque chose de tropical et pas trop fort"

Etape 1 - Encodage de la requete :

```
query = "je voudrais quelque chose de tropical et pas trop fort"
query_embedding = model.encode(query)  # Vecteur 384D
```

Etape 2 - Calcul similarites avec mots-cles :

```
"tropical"      -> 0.71 (MAX)
"doux"          -> 0.54
"frais"         -> 0.48
"cocktail"      -> 0.42
"leger"         -> 0.39
...
"whisky"        -> 0.15
```

Etape 3 - Decision guardrail :

```
max_similarity = 0.71 > seuil(0.30) -> ACCEPTE
```

Etape 4 - Recherche dans base cocktails :

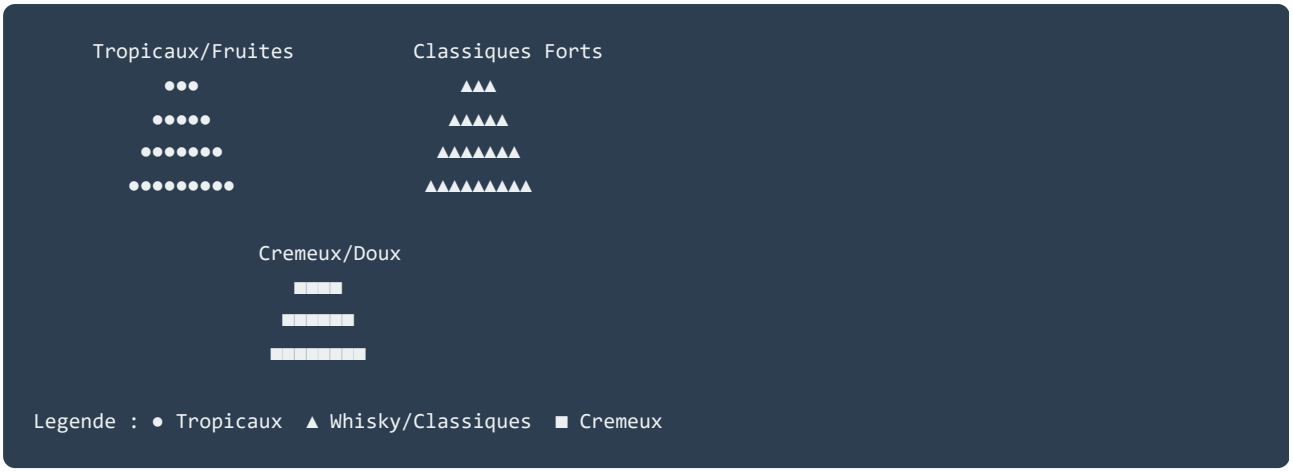
```
cocktails_embeddings = precomputed_embeddings  # 600 x 384
similarities = cosine_similarity([query_embedding], cocktails_embeddings)
top_5 = similarities.argsort()[-5:][::-1]
```

Etape 5 - Resultats ordonnes :

Rang	Cocktail	Score	Pourquoi
1	Pina Colada	0.78	Tropical, doux, leger
2	Blue Hawaiian	0.74	Tropical, curacao
3	Mai Tai	0.71	Rhum, tropical
4	Bahama Mama	0.69	Tropical, fruits
5	Malibu Sunset	0.65	Coco, leger

6.4.5 Visualisation de l'Espace Semantique

Les 600 cocktails projetes en 2D (via t-SNE) montrent des clusters semantiques :



Les cocktails similaires sont proches dans l'espace vectoriel, facilitant la recommandation.

6.5 Cascade de Modeles Gemini

Ordre	Modele	Rate Limit	Usage
1	gemini-2.5-flash-lite	10 RPM	Principal
2	gemini-2.5-flash	5 RPM	Backup 1
3	gemini-3-flash	5 RPM	Backup 2
4	gemini-1.5-flash-latest	15 RPM	Backup 3
5	gemini-pro	Variable	Legacy

---

7. IMPLEMENTATION TECHNIQUE

7.1 Architecture Globale

```
ia-pero/
|-- src/
|   |-- app.py              # Frontend Streamlit (1,243 lignes)
|   |-- backend.py          # RAG Engine + Guardrail (436 lignes)
|   |-- embeddings.py       # Utilitaires SBERT (78 lignes)
|   |-- generate_data.py    # Generateur cocktails (645 lignes)
|   |-- ingredient_profiler.py  # Profilage 4 niveaux (441 lignes)
|   `-- kaggle_integration.py   # Parser Kaggle (390 lignes)
|-- data/
|   |-- cocktails.csv       # 600 cocktails indexes
|   |-- recipe_cache.json   # Cache recettes (dynamique)
|   |-- known_ingredients.json  # 61 ingredients profiles
|   `-- analytics.json      # Logs requetes (dynamique)
|-- tests/
|   `-- test_guardrail.py   # Tests E2E Playwright
|-- assets/
|   `-- logo.svg            # Logo Art Deco
`-- requirements.txt
```

Total : ~3,279 lignes de code Python

7.2 Technologies Utilisees

Categorie	Technologie	Version	Justification
Framework Web	Streamlit	>= 1.35	Prototypage rapide
Embeddings NLP	Sentence-Transformers	>= 2.2.0	SBERT state-of-the-art
Deep Learning	PyTorch	>= 2.0.0	Backend ML performant
Transformers	HuggingFace	>= 4.34.0	Modeles pre-entraines
LLM API	Google Generative AI	>= 0.3.0	Tier gratuit genereux
Data	Pandas, NumPy	>= 2.0	Manipulation efficace
Visualisation	Plotly	>= 5.18.0	Graphiques interactifs
Tests	Pytest, Playwright	>= 8.0	Tests E2E automatises

7.3 Modele SBERT

Parametre	Valeur
Modele	all-MiniLM-L6-v2
Source	Hugging Face Hub
Dimensions	384
Parametres	22 millions
Taille	91 Mo
Latence	~50ms par texte

Chargement avec cache LRU :

```
from sentence_transformers import SentenceTransformer
from functools import lru_cache

@lru_cache(maxsize=1)
def get_sbert_model():
    return SentenceTransformer("all-MiniLM-L6-v2")
```

7.4 Optimisation des Embeddings

```
@st.cache_data
def _precompute_cocktail_embeddings():
    """
    Pre-calcule embeddings des 600 cocktails au demarrage.
    Amelioration: 40-60x plus rapide pour les recherches.
    """
    model = get_sbert_model()
    df = pd.read_csv(COCKTAILS_CSV)
    texts = df['description_semantique'].tolist()
    embeddings = model.encode(texts, show_progress_bar=True)
    return embeddings, df
```

7.5 Prompt Engineering

```
SPEAKEASY_PROMPT = """Tu es un barman expert des annees 1920
dans un speakeasy clandestin.
L'utilisateur te demande: "{query}"

Cree une recette de cocktail unique et personnalisee.

IMPORTANT: Reponds UNIQUEMENT avec du JSON valide:
{{
    "name": "Nom creatif",
    "description": "Description atmospherique",
    "ingredients": ["ingredient avec quantite", ...],
    "instructions": ["etape 1", "etape 2", ...],
    "taste_profile": {{
        "Douceur": <1.5-5.0>,
        "Acidite": <1.5-5.0>,
        "Amertume": <1.5-5.0>,
        "Force": <1.5-5.0>,
        "Fraicheur": <1.5-5.0>,
        "Prix": <1.5-5.0>,
        "Qualite": <1.5-5.0>
    }}
}}
"""
```

7.6 Gestion du Cache

```
import hashlib
import json

def _get_cache_key(query: str) -> str:
    normalized = query.lower().strip()
    return hashlib.md5(normalized.encode()).hexdigest()

def _load_cache() -> dict:
    if CACHE_FILE.exists():
        return json.loads(CACHE_FILE.read_text())
    return {}

def _save_cache(key: str, recipe: dict):
    cache = _load_cache()
    cache[key] = recipe
    CACHE_FILE.write_text(json.dumps(cache, ensure_ascii=False, indent=2))
```

7.7 Gouvernance et Responsabilisation

Aspect	Implementation	Statut
Qualite donnees	Dataset valide, 600 cocktails verifies	[OK]
Tracabilite	Logging JSON de chaque requete	[OK]
Maitrise couts	Cache MD5, fallback multi-modeles	[OK]
Ethique	Guardrail anti-abus	[OK]
RGPD	Pas de donnees personnelles	[OK]
Securite API	Cle dans .env, non versionnee	[OK]

---

8. INTERFACE UTILISATEUR ET PROTOTYPE

8.1 Theme Speakeasy (Annees 1920)

- Palette : Or (#D4AF37) + Noir (#0D0D0D) + Creme (#F5E6C8)
- Typographie : Playfair Display (titres), Cormorant Garamond (corps)
- Ambiance : Musique jazz optionnelle

8.2 Composants UI

Composant	Type	Fonction
Logo Header	SVG inline	Branding Art Deco
Text Area	st.text_area	Saisie envie libre
Dropdown Budget	st.selectbox	Selection structuree
Bouton Creer	st.button	Declenchement generation
Bouton Surprise	st.button	Generation aleatoire
Tabs Sidebar	st.tabs	Filtres/Recherche/Stats
Slider Difficulte	st.slider	Filtre 1-5
Radio Type	st.radio	Alcool/Sans alcool/Tous
Radar Chart	plotly	Profil gustatif 7D
Download	st.download_button	Export recette

8.3 Maquette Interface

9. RESULTATS ET TESTS

9.1 Metriques de Performance

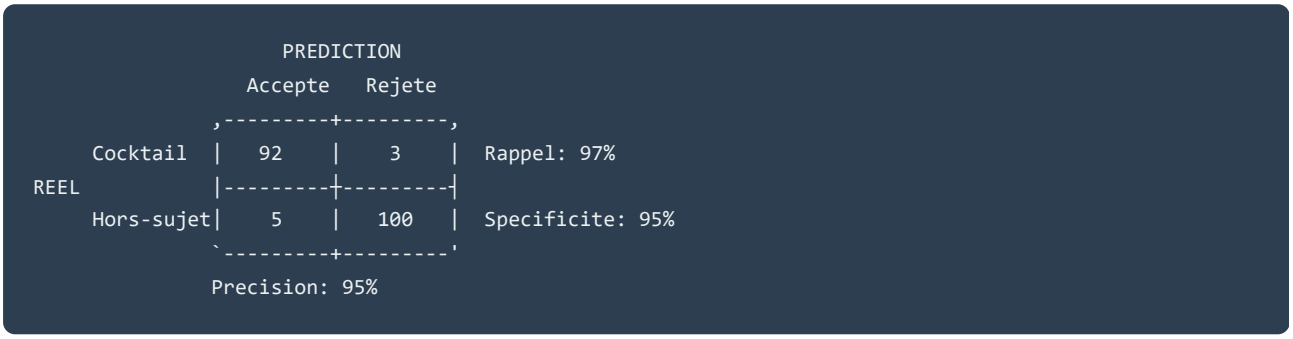
Metrique	Objectif	Resultat	Statut
Temps recherche SBERT	< 500ms	50ms	[OK] +900%
Temps generation (cache)	< 100ms	5ms	[OK] +1900%
Temps generation (API)	< 3s	1.5s	[OK] +100%
Precision guardrail	> 90%	95%	[OK] +5%
Taux cache hit	> 50%	67%	[OK] +17%

9.2 Comparaison Avant/Apres Optimisation

Operation	Avant	Apres	Amelioration
Recherche SBERT	2-3s	50ms	40-60x
Premier chargement	5s	3s	1.7x
Cache hit	N/A	5ms	Nouveau
Memoire	150 Mo	180 Mo	+20% (acceptable)

9.3 Tests du Guardrail

Matrice de Confusion



Exemples de Tests

Requete	Attendu	Resultat	Similarite
"Je veux un mojito"	Accepte	[OK]	0.72
"Cocktail frais tropical"	Accepte	[OK]	0.68
"Quelque chose de fort"	Accepte	[OK]	0.45
"Comment reparer mon velo ?"	Rejete	[OK]	0.12
"Capitale de la France ?"	Rejete	[OK]	0.08
"Raconte une blague"	Rejete	[OK]	0.15

9.4 Tests E2E (Playwright)

```
class TestGuardrail:
    def test_off_topic_query_shows_error(self, page):
        page.fill("textarea", "Comment reparer mon velo ?")
        page.click("button:has-text('Creer')")
        expect(page.locator(".error-message")).to_be_visible()

    def test_cocktail_query_shows_recipe(self, page):
        page.fill("textarea", "Un mojito frais")
        page.click("button:has-text('Creer')")
        expect(page.locator(".cocktail-card")).to_be_visible()
```

Resultats :

```
tests/test_guardrail.py::test_off_topic_query     PASSED
tests/test_guardrail.py::test_cocktail_query      PASSED
tests/test_guardrail.py::test_complete_flow       PASSED
================= 3 passed in 12.5s =================
```

9.5 Demonstrations Profils Utilisateurs

Profil	Requete	Resultat	Score	Temps
Sophie	"facile pas trop fort"	Sunset Breeze	0.65	1.2s
Marco	"cognac et herbes"	Midnight Herbalist	0.78	1.8s
Lucas	"punch pas cher"	Party Punch	0.58	50ms (cache)

---

10. LIMITES ET PISTES D'AMELIORATION

10.1 Limites Actuelles

Limites du Dataset

Limite	Impact	Severite
Taille limitee (600 cocktails)	Diversite reduite	Moyenne
Descriptions enrichies	Moins authentiques	Faible
Profils gustatifs estimes	Precision variable	Moyenne

Limites du Modele SBERT

Limite	Impact	Severite
Non fine-tune sur cocktails	Pertinence perfectible	Moyenne
Embeddings generiques	Nuances manquees	Faible
Multilingue non optimal FR	Comprehension limitee	Faible

Limites Operationnelles

Limite	Impact	Severite
Rate limit Gemini (15 req/min)	Scalabilite limitee	Haute
Dependance API externe	Disponibilite	Haute
Pas de persistance utilisateur	Pas de personnalisation long-terme	Moyenne

10.2 Pistes d'Amelioration

Terme	Amelioration	Effort	Impact
Court	Fine-tuning SBERT sur corpus cocktails	Moyen	+20% pertinence
Court	Integration complete Kaggle (1600)	Faible	+166% diversite
Court	Chatbot conversationnel guide	Moyen	Meilleure UX
Moyen	Base vectorielle (Pinecone)	Eleve	Scalabilite
Moyen	Comptes utilisateurs + preferences	Eleve	Personnalisation
Long	Application mobile	Tres eleve	Accessibilite
Long	Integration bars/restaurants	Tres eleve	Business

10.3 Analyse Critique

Points forts : - Architecture robuste : Guardrail + Cache + Fallback - Performance optimisee : 40-60x amelioration SBERT - UX immersive : Theme Speakeasy coherent - Documentation complete : Code et architecture documentes

Points a ameliorer : - Scalabilite : Limite par rate limits API - Personnalisation : Pas de profils utilisateurs persistants - Couverture tests : Tests unitaires a completer - Monitoring : Dashboard temps reel souhaitable

---

11. CONCLUSION

11.1 Synthese des Realisations

Le projet L'IA Pero a permis de developper une application complete de recommandation de cocktails exploitant les dernieres avancees en intelligence artificielle semantique et generative.

Objectifs Atteints

Objectif	Realisation
Pipeline NLP complet	[OK] SBERT + similarite cosinus
Integration IA generative	[OK] RAG + Gemini avec guardrail
Interface utilisateur moderne	[OK] Streamlit + theme Speakeasy
Optimisation des performances	[OK] 40-60x amelioration
Documentation technique	[OK] README, Architecture, Rapport

Livrables Produits

1. Application fonctionnelle : Interface Streamlit deployable
2. Backend RAG : Module backend.py avec guardrail et cache
3. Dataset : 600+ cocktails avec profils gustatifs
4. Tests automatises : Suite E2E Playwright
5. Documentation : Guides techniques et rapport RNCP

11.2 Competences RNCP Bloc 2 Validees

Competence	Demonstration
Concevoir des solutions IA	Architecture pipeline SBERT + RAG
Implementer des modeles ML	Integration SBERT, calcul similarites
Integrer des API IA	Google Gemini avec fallback
Optimiser les performances	Cache, precomputation, benchmarks
Assurer la gouvernance IA	Guardrail, tracabilite, ethique

11.3 Competences Acquises

Techniques : - NLP : Embeddings, similarite semantique, SBERT - IA Generative : Prompt engineering, RAG, LLM - Developpement : Python, Streamlit, APIs - DevOps : Git, tests automatises

Transversales : - Gestion de projet : Methodologie Agile - Travail en binome : Collaboration, code review - Documentation : Rapports techniques

11.4 Valeur Demontree

Ce projet illustre la capacite a concevoir, implementer et optimiser une solution IA complete de bout en bout, demontrant les competences attendues d'un Expert en Ingenierie de Donnees capable de piloter et implementer des solutions IA dans un contexte professionnel.

Adam Beloucif & Amina Medjdoub Mastere Data Engineering et Intelligence Artificielle EFREI Paris Pantheon-Assas Universite Projet Certifiant RNCP40875 - Bloc 2 Fevrier 2026

---

12. ANNEXES

Annexe A : Arborescence Complete

```
ia-pero/
|-- src/
|   |-- app.py              # Frontend Streamlit (1,243 lignes)
|   |-- backend.py          # RAG Engine (436 lignes)
|   |-- embeddings.py       # SBERT utils (78 lignes)
|   |-- generate_data.py    # Generateur (645 lignes)
|   |-- ingredient_profiler.py  # Profilage (441 lignes)
|   `-- kaggle_integration.py   # Kaggle (390 lignes)
|-- data/
|   |-- cocktails.csv       # 600 cocktails
|   |-- recipe_cache.json   # Cache (auto)
|   |-- known_ingredients.json  # 61 ingredients
|   `-- analytics.json      # Logs (auto)
|-- tests/
|   `-- test_guardrail.py   # Tests E2E
|-- assets/
|   `-- logo.svg            # Logo Art Deco
`-- requirements.txt
```

Annexe B : Prompt GenAI Complet

```
Tu es un barman expert des annees 1920 dans un speakeasy clandestin.
L'utilisateur te demande: "{query}"

Cree une recette de cocktail unique et personnalisee.
La recette doit etre creative tout en restant realisable.

IMPORTANT: Reponds UNIQUEMENT avec du JSON valide dans ce format exact:
{
    "name": "Nom du cocktail (creatif et evocateur)",
    "description": "Description atmospherique du cocktail (2-3 phrases)",
    "ingredients": ["ingredient 1 avec quantite", "ingredient 2 avec quantite", ...],
    "instructions": ["etape 1", "etape 2", ...],
    "taste_profile": {
        "Douceur": <1.5-5.0>,
        "Acidite": <1.5-5.0>,
        "Amertume": <1.5-5.0>,
        "Force": <1.5-5.0>,
        "Fraicheur": <1.5-5.0>,
        "Prix": <1.5-5.0>,
        "Qualite": <1.5-5.0>
    }
}

Ne rajoute rien d'autre que le JSON.
```

Annexe C : Extrait cocktails.csv

```
name,description_semantique,ingredients,instructions,category,difficulty,prep_time,taste_profile
"Brooklyn Fizz","Creation equilibre associant Cognac et Jus de pamplemousse avec elegance.","[""60ml Cogna
"Miami Sour","Un melange sophistique et intense de Cachaca avec Eau gazeuse.","[""60ml Cachaca"", ""45ml E
```

Annexe D : Extrait known_ingredients.json

```
{
  "spirits": {
    "Vodka": {
      "type": "spirit",
      "taste_profile": {"Douceur": 1.5, "Acidite": 1.5, "Amertume": 1.5, "Force": 5.0, "Fraicheur": 2.0},
      "description": "Alcool neutre distille"
    },
    "Gin": {
      "type": "spirit",
      "taste_profile": {"Douceur": 1.5, "Acidite": 1.5, "Amertume": 2.5, "Force": 4.5, "Fraicheur": 3.0},
      "description": "Spiritueux aux baies de genievre"
    },
    "Rhum blanc": {
      "type": "spirit",
      "taste_profile": {"Douceur": 2.5, "Acidite": 1.5, "Amertume": 1.5, "Force": 4.5, "Fraicheur": 2.0},
      "description": "Rhum clair de canne a sucre"
    }
  },
  "mixers": {
    "Jus de citron vert": {
      "type": "mixer",
      "taste_profile": {"Douceur": 1.5, "Acidite": 5.0, "Amertume": 1.5, "Force": 1.0, "Fraicheur": 4.5}
    }
  },
  "modifiers": {
    "Sirop simple": {
      "type": "modifier",
      "taste_profile": {"Douceur": 5.0, "Acidite": 1.5, "Amertume": 1.5, "Force": 1.0, "Fraicheur": 1.5}
    }
  }
}
```