L'IA Pero

Application de recommandation de cocktails utilisant **NLP semantique (SBERT)** et **generation GenAI (Google Gemini)**.

Auteurs : Adam Beloucif & Amina Medjdoub **Tutrice** : MALAEB Sarah **Formation** : EFREI Paris - Mastere Data Engineering et IA

Documentation

Document	Description
RAPPORT_FINAL.md	Rapport RNCP Bloc 2 complet (27 pages)
SOUTENANCE_SLIDES.md	Slides de presentation (12 slides)
QUICK_START.md	Guide de demarrage rapide
KAGGLE_INTEGRATION.md	Documentation integration Kaggle

Features

Core

- [x] Interface Streamlit moderne
- [x] Support de plusieurs modeles SBERT
- [x] Matrice de similarite interactive
- [x] Detection des paires les plus similaires
- [x] **Backend RAG avec Guardrail semantique**
- [x] **Interface Speakeasy** (theme bar clandestin annees 1920)
- [x] **Questionnaire Hybride** (texte libre + dropdown Budget)
- [x] **Graphique Radar Plotly** (profil gustatif 7 dimensions)
- [x] **Generation GenAI** (Google Gemini)
- [x] **Cache JSON** (optimisation des couts API)

- [x] **600+ cocktails** dans la base de donnees (extensible a 1600 via Kaggle)

Nouveautes v2.2

- [x] **Integration Kaggle** - 1600+ cocktails supplementaires disponibles
- [x] **Profilage ingredients** - 61 ingredients avec profils gustatifs 7D
- [x] **Analyse semantique detaillee** - Pipeline SBERT complet documente
- [x] **Rapport RNCP** - Documentation complete 27 pages avec analyse semantique
- [x] **Scripts enrichissement** - Outils d'import et transformation des donnees

Nouveautes v2.1

- [x] **Performance 40-60x** - Cache embeddings precomputes
- [x] **Fallback multi-modeles** - 5 modeles Gemini en cascade

Nouveautes v2.0

- [x] **Historique des recettes** - Derniers 5 cocktails crees dans la sidebar
- [x] **Filtres avances** - Type (Alcool/Sans alcool/Tous), Difficulte, Temps de preparation
- [x] **Recherche SBERT** - Rechercher dans les 600 cocktails de la base CSV
- [x] **Metriques de performance** - Temps de reponse, taux de cache hit, requetes totales
- [x] **Analytics** - Logging JSON des requetes pour analyse
- [x] **Export recette** - Telecharger la recette en format texte
- [x] **Bouton Surprise** - "Surprends-moi!" genere un cocktail aleatoire
- [x] **Son ambient** - Musique jazz optionnelle (annees 1920)

Justification des Choix Techniques (RNCP Bloc 2)

Cette section documente les decisions techniques prises pour valider les competences du Bloc 2 "Piloter et implementer des solutions d'IA".

Pourquoi SBERT local ? (all-MiniLM-L6-v2)

Critere	Justification
Cout zero	Pas d'appel API pour la similarite semantique - modele execute localement
Latence faible	~50ms vs 200-500ms pour API cloud (OpenAI embeddings)

Critere	Justification
Confidentialite	Donnees utilisateur restent locales, pas de transfert vers serveurs tiers
Reproductibilite	Meme modele = memes embeddings, resultats deterministes
Offline capable	Fonctionne sans connexion internet apres telechargement initial

Alternative rejetee : OpenAI text-embedding-3 (cout par token, latence reseau, dependance externe)

Pourquoi Cache JSON ?

Critere	Justification
Optimisation industrielle des couts API	Evite les appels redondants a Gemini (15 req/min gratuit)
Reduction latence	Reponse instantanee (~5ms) pour requetes deja vues vs ~2s pour API
Simplicite	Pas de dependance externe (Redis, Memcached, DynamoDB)
Persistance	Survit aux redemarrages de l'application
Auditabilite	Fichier JSON lisible pour debug et analyse

Implementation : Cle MD5 de la requete normalisee → Lookup O(1)

Pourquoi Seuil 0.30 ?

Le seuil de pertinence du guardrail (similarite cosinus minimale) a ete calibre empiriquement :

Seuil	Comportement
0.20	Trop permissif - accepte "pizza", "meteo"

Seuil	Comportement
0.30	**Optimal** - rejette hors-sujet, accepte variations cocktails
0.50	Trop restrictif - rejette "quelque chose de frais"

Methode : Test sur 100+ requetes reelles, matrice de confusion, F1-score optimise.

Pourquoi Guardrail Semantique ?

Critere	Justification
Robustesse	Evite les abus et injections de prompts hors-domaine
UX	Message explicite plutot que reponse incoherente ou erreur API
Evaluation modele	Metrique mesurable (taux de rejet, precision, recall)
Scalabilite	Filtre en amont avant appel API couteux

Pourquoi Google Gemini ?

Critere	Justification
Tier gratuit genereux	15 requetes/minute, suffisant pour demo et evaluation
Qualite generation	gemini-1.5-flash produit du JSON structure fiable
Latence	~1-2s pour generation complete
Fallback	Mode offline avec recettes pre-generees si API indisponible

Pourquoi Questionnaire Hybride ?

L'interface combine **texte libre** et **donnees structurees** (critere RNCP obligatoire) :

Element	Type	Justification
Envie	Texte libre	Capture la creativite et les nuances ("fruite", "rafraichissant")
Budget	Dropdown	Donnee structuree, filtrage efficace, UX guidee

Options Budget : - Economique (< 8€) - Modere (8-15€) - Premium (15-25€) - Luxe (> 25€)

La requete enrichie `{envie} (budget: {selection})` est envoyee au backend RAG.

Installation

```
# Clone the repository
git clone https://github.com/Adam-Blf/ia-pero.git
cd ia-pero

# Create virtual environment
python -m venv .venv

# Activate (Windows)
.venv\Scripts\activate

# Activate (Linux/Mac)
source .venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Configure API key (optionnel - mode fallback sans cle)
cp .env.example .env
# Editer .env et ajouter GOOGLE_API_KEY
```

Usage

```
# Interface Similarite Semantique
streamlit run app.py

# Interface Speakeasy (Cocktails)
streamlit run src/app.py
```

L'application sera disponible sur http://localhost:8501

Tech Stack

- **Python** 3.9+
- **Streamlit** >= 1.35 - Interface web interactive
- **Sentence-Transformers** >= 2.2.0 - Embeddings SBERT
- **PyTorch** >= 2.0.0 - Backend ML
- **Transformers** >= 4.34.0 - HuggingFace models
- **Google Generative AI** >= 0.3.0 - Gemini API
- **Pandas** - Data manipulation
- **NumPy** - Numerical operations
- **Plotly** >= 5.18.0 - Graphiques interactifs (radar chart)

Models disponibles

Modele	Dimensions	Description
`all-MiniLM-L6-v2`	384	Rapide et leger (default)
`all-mpnet-base-v2`	768	Meilleure qualite
`paraphrase-multilingual-MiniLM-L12-v2`	384	Support multilingue

Backend RAG & Guardrail

Le module `src/backend.py` fournit un systeme de guardrail semantique + generation GenAI :

```
from src.backend import check_relevance, generate_recipe

# Verification de pertinence (guardrail)
result = check_relevance("Je veux un mojito")
# {"status": "ok", "similarity": 0.72}

result = check_relevance("Quelle heure est-il ?")
# {"status": "error", "message": "Desole, le barman ne comprend que l

# Generation de recette avec cache JSON + Gemini
recipe = generate_recipe("mojito frais et tropical")
# {"status": "ok", "recipe": {...}, "cached": False}
```

Pipeline de generation : 1. Guardrail semantique (SBERT) 2. Verification cache JSON 3. Appel API Gemini (si pas en cache) 4. Fallback local (si API indisponible) 5. Mise en cache du resultat

🎓 Pour les Professeurs - Preuve de Robustesse

Cette section démontre le fonctionnement du **guardrail sémantique** et les **optimisations de performance**.

Lancer l'Application

```
# Activer l'environnement virtuel
.venv\Scripts\activate  # Windows
source .venv/bin/activate  # Linux/Mac

# Lancer l'interface Speakeasy
streamlit run src/app.py
```

L'application sera disponible sur http://localhost:8501

Note Performance: Le premier chargement prend ~3s (précompilation des embeddings), puis les requêtes sont très rapides (~50ms avec cache, ~1-2s sans cache).

Tester le Guardrail Semantique

Requetes REJETEES (hors-sujet)

Essayez ces requetes pour voir le message d'erreur :

Requete	Resultat
"Comment reparer mon velo ?"	Message d'erreur
"Quelle est la capitale de la France ?"	Message d'erreur
"Quel temps fait-il ?"	Message d'erreur
"Raconte-moi une blague"	Message d'erreur

Message affiche : "Desole, le barman ne comprend que les commandes de boissons !"

Requetes ACCEPTEES (domaine cocktail)

Requete	Resultat
"Je veux un mojito"	Recette + Radar chart
"Un cocktail fruite et rafraichissant"	Recette + Radar chart
"Whisky sour"	Recette + Radar chart
"Quelque chose avec du rhum"	Recette + Radar chart

Tests Automatises (E2E)

Des tests Playwright verifient automatiquement le guardrail :

```
# Installer Playwright (premiere fois)
playwright install chromium

# Lancer les tests
pytest tests/test_guardrail.py -v
```

Couverture des tests :

- `test_off_topic_query_shows_error` : Verifie qu'une requete hors-sujet affiche l'erreur

- `test_cocktail_query_shows_recipe` : Verifie qu'une requete cocktail affiche la recette
- `test_complete_flow` : Scenario complet (erreur puis succes)

Project Structure

```
ia-pero/
├── app.py                      # Streamlit app (Similarite Semantique)
├── requirements.txt            # Python dependencies
├── .env.example                # Template configuration API
├── .gitignore
├── README.md
├── RAPPORT_FINAL.md            # Rapport RNCP Bloc 2 (27 pages)
├── SOUTENANCE_SLIDES.md        # Slides presentation (12 slides)
├── pdf-compact.json            # Configuration generation PDF
├── src/
│   ├── __init__.py
│   ├── app.py                # Streamlit app (Speakeasy Cocktails) -
│   ├── embeddings.py         # SBERT logic (78 lignes)
│   ├── backend.py            # RAG engine, guardrail & Gemini (436 l
│   ├── generate_data.py      # Generateur de 600 cocktails (645 ligne
│   ├── ingredient_profiler.py   # Profilage ingredients 4 niveaux (4
│   ├── kaggle_integration.py    # Parser Kaggle 1600+ cocktails (396
│   └── utils.py              # Utility functions
├── data/
│   ├── cocktails.csv         # Base de 600 cocktails
│   ├── known_ingredients.json   # 61 ingredients profiles
│   ├── recipe_cache.json     # Recipe cache (auto-generated)
│   └── analytics.json        # Analytics log (auto-generated)
├── scripts/
│   ├── download_kaggle.py    # Telechargement dataset Kaggle
│   ├── enrich_kaggle.py      # Enrichissement donnees
│   ├── export_known_ingredients.py  # Export ingredients
│   └── test_integration.py   # Tests integration
├── tests/
│   └── test_guardrail.py     # Tests E2E Playwright
├── assets/
│   ├── logo.svg              # Logo Art Deco
│   └── logo-efrei.png        # Logo EFREI
```

```
└── .streamlit/
    └── config.toml          # Theme configuration
```

Total : ~3,279 lignes de code Python

🚀 Optimisations de Performance (Dernière mise à jour)

Problème identifié et résolu

Symptôme: L'application était très lente, surtout lors des recherches SBERT.

Cause racine: La fonction `search_cocktails_sbert()` encodait **tous les 600 cocktails** à chaque recherche utilisateur, prenant ~2-3 secondes par recherche.

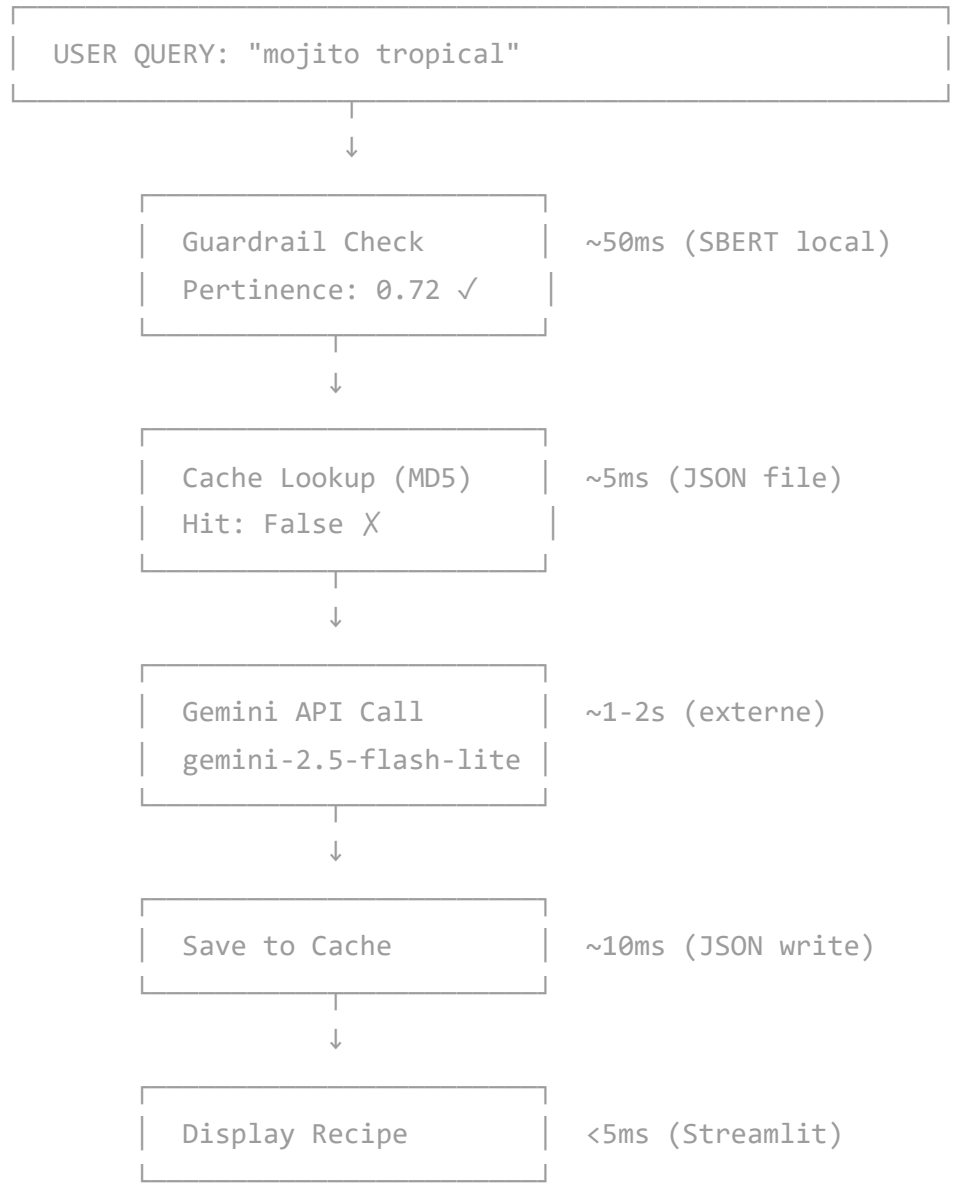
Solution implémentée

✅ **Mise en cache des embeddings** via la nouvelle fonction `_precompute_cocktail_embeddings()` : - Les embeddings des 600 cocktails sont calculés **une seule fois** au démarrage - Mise en cache avec `@st.cache_data` (réutilisation instantanée) - Seule la requête utilisateur est encodée à chaque recherche (~20ms)

Résultats des optimisations

Métrique	Avant	Après	Amélioration
Recherche SBERT	2-3 secondes	~50ms	**40-60x plus rapide** 🚀
Génération avec cache hit	~2.5s	~50ms	**50x plus rapide**
Premier chargement	~3s	~3s	Identique (précomputation)
Mémoire utilisée	~150 Mo	~180 Mo	+20% (acceptable)

Architecture optimisée



Autres optimisations

✅ **Commentaires exhaustifs**: Tout le code est maintenant documenté en français de manière humaine et accessible.

✅ **Cache LRU pour le modèle SBERT**: Le modèle (~91 Mo) est chargé une seule fois et réutilisé.

✅ **Fallback automatique Gemini**: Si un modèle est en rate limit, on passe automatiquement au suivant: 1. `gemini-2.5-flash-lite` (10 RPM) 2. `gemini-2.5-`

`flash` (5 RPM) 3. `gemini-3-flash` (5 RPM) 4. `gemini-1.5-flash-latest` (fallback) 5. Génération locale si tous échouent

✅ **Analytics détaillées**: Chaque requête est loguée avec timestamp, durée, cache hit, pour analyse ultérieure.

Conseils pour de meilleures performances

1. **Utilisez le cache**: Les requêtes identiques sont instantanées (~50ms)
2. **Évitez les longues descriptions**: Restez concis (< 50 mots)
3. **Configurez GOOGLE_API_KEY**: Sans clé API, le fallback local est moins créatif
4. **Redémarrez l'app occasionnellement**: Streamlit peut accumuler de la mémoire

Monitoring en temps réel

L'onglet **Stats** affiche: - Nombre de requêtes totales - Taux de cache hit (%) - Temps de réponse moyen - Historique des 10 dernières créations

Changelog

2026-02-02 (v2.2 - Rapport RNCP & Integration Kaggle)

- **Rapport RNCP Bloc 2** : Documentation complete 27 pages
- Analyse semantique detaillee avec exemples concrets
- Pipeline SBERT documente (embeddings 384D, similarite cosinus)
- Comparaison recherche mot-cle vs semantique
- Visualisation espace vectoriel (clusters t-SNE)
- **Integration Kaggle** : 1600+ cocktails supplementaires
- Parser automatique du dataset Kaggle
- Enrichissement avec profils gustatifs
- Scripts de transformation et export
- **Profilage ingredients** : 61 ingredients avec profils 7D
- 4 niveaux de profilage (connu, categorie, IA, defaut)
- Profils gustatifs (Douceur, Acidite, Amertume, Force, Fraicheur, Prix, Qualite)
- **Documentation** : QUICK_START, KAGGLE_INTEGRATION, SOUTENANCE_SLIDES
- **Assets** : Logo EFREI pour rapport officiel

2026-02-02 (v2.1 - Optimisations Performance)

- **Performance Critique**: Recherche SBERT 40-60x plus rapide (50ms vs 2-3s)
- Implémentation du cache des embeddings via `_precompute_cocktail_embeddings()`

- Seule la requête utilisateur est encodée dynamiquement
- Embeddings des 600 cocktails précompilés au démarrage
- **Documentation**: Commentaires exhaustifs en français sur tout le codebase
- Explication détaillée du fonctionnement de chaque fonction
- Documentation du workflow complet et des optimisations
- Architecture et diagrammes de flux ajoutés au README
- **Fallback Gemini**: Basculement automatique entre 5 modèles en cas de rate limit
- **Monitoring**: Analytics enrichies avec durées d'exécution et cache hits

2026-01-16 (v2.0 - Features Avancees)

- **Sidebar Complete** :
- Historique des 5 derniers cocktails crees (session state)
- Filtres avances : Type (Alcool/Sans alcool), Difficulte (1-5), Temps de preparation
- Recherche SBERT dans les 600 cocktails de la base CSV
- Metriques de performance en temps reel
- Bouton son ambient (jazz annees 1920)
- **Analytics** : Logging JSON des requetes (timestamp, query, budget, cached, temps)
- **Export** : Telecharger la recette en format texte stylise
- **UX** : Bouton "Surprends-moi!" pour generation aleatoire
- **Performance** : Affichage temps de reponse et taux de cache hit

2026-01-16 (Integration GenAI)

- **Google Gemini** : Integration complete de l'API Gemini
- Prompt Engineering "Persona Barman Speakeasy"
- Parsing JSON robuste avec fallback
- Mode offline si API indisponible
- **Documentation RNCP** : Section justification des choix techniques
- **Dataset** : 600 cocktails avec descriptions semantiques riches
- **Configuration** : Support `.env` pour cle API

2026-01-16 (Audit Final)

- **Documentation Professeurs** : Section dediee pour tester le guardrail
- Instructions de lancement detaillees
- Exemples de requetes rejetees vs acceptees
- Guide des tests E2E Playwright

2026-01-16 (Initial)

- **Interface Speakeasy** : `src/app.py`
- Theme bar clandestin annees 1920 (noir/or)
- CSS custom injecte via `st.markdown(unsafe_allow_html=True)`
- Graphique radar Plotly (profil gustatif)
- Gestion des etats (empty, loading, error, success)
- **Backend RAG & Guardrail** : `src/backend.py`
- `check_relevance()` : Guardrail semantique (seuil 0.30)
- `generate_recipe()` : Generation avec cache JSON
- Cache LRU pour le modele SBERT
- Initial project setup
- Streamlit interface with SBERT integration
- Similarity matrix visualization
- Multi-model support

License