PROMPT D'IMPLÉMENTATION RAG

Système LLM + RAG sur GCP — Modèles gratuits (Free Tier)

Thèse POC · Copilote & Orchestrateur Demi-Fond · Vertex AI + Tokens d'essai

1. MASTER PROMPT — IMPLÉMENTATION COMPLÈTE

Ce prompt est utilise dans Google AI Studio, Vertex AI Workbench ou dans le code Python via Gemini gratuite.

1.1 PROMPT — Section A : Contexte et objectif du projet

```
Tu es un ingénieur ML senior expert en RAG (Retrieval-Augmented Generation)
sur Google Cloud Platform.

Tu vas m'aider à implémenter de A à Z un système RAG pour une application
d'entraînement sportif (demi-fond).


CONTEXTE DU PROJET :
_____________________________________________________________________________
_

• Thèse universitaire : POC d'IA pour entraîneurs en athlétisme (demi-fond)

• Deux architectures cibles :

  1. COPILOTE : assistant léger pour 1 entraîneur / <10 athlètes (budget
<50€/mois)

  2. ORCHESTRATEUR : système agentique pour fédération / >50 athlètes
(budget >500€/mois)

• Contrainte budget : utiliser EXCLUSIVEMENT les services GCP en free tier /
tokens d'essai

• Stack cible : Python 3.11 + FastAPI + pgvector (Cloud SQL) + Vertex AI
Gemini

• Le corpus RAG contient 20 chunks structurés sur l'entraînement en demi-
fond


CONTRAINTE PRINCIPALE : Utiliser les modèles GCP disponibles gratuitement :

  - Vertex AI : gemini-1.5-flash (FREE : 15 req/min, 1M tokens/min, 1500
req/jour)

  - Vertex AI Embeddings : text-multilingual-embedding-002 (FREE : 250 req/
min)

  - Alternative si quota dépassé : Groq API (llama3-70b, GRATUIT, 14400 req/
j)

  - Embedding de secours local : sentence-transformers/all-MiniLM-L6-v2 (0€,
0 limite)
_____________________________________________________________________________
_
```

1.2 PROMPT — Section B : Architecture à implémenter

```
ARCHITECTURE COPILOTE À IMPLÉMENTER :
═══════════════════════════════════════════════════════════════════════════

Stack complet :

  [Documents sources] → Cloud Storage (GCS)
       ↓
  [Ingestion Pipeline] → Cloud Run Job (Python)
       ├─ Chunking : RecursiveCharacterTextSplitter (400 tokens, overlap 80)
       ├─ Embedding : text-multilingual-embedding-002 (Vertex AI, GRATUIT)
       └─ Stockage : pgvector sur Cloud SQL PostgreSQL 15 (e2-micro, ~7€/
mois)

  [API FastAPI] → Cloud Run Service
       ├─ Endpoint POST /query
       ├─ Recherche ANN : pgvector (cosine similarity, top-k=5)
       ├─ Re-ranking : cross-encoder léger (local dans le container)
       ├─ Generation : gemini-1.5-flash (Vertex AI FREE TIER)
       │    └─ Fallback auto : Groq llama3-8b si quota dépassé
       └─ Score de confiance : calcul composite (similarité + couverture)

Base de données pgvector :
  CREATE EXTENSION IF NOT EXISTS vector;
  CREATE TABLE rag_chunks (
    id SERIAL PRIMARY KEY,
    chunk_id VARCHAR(50) UNIQUE NOT NULL,
    content TEXT NOT NULL,
    embedding vector(768),  -- text-multilingual-embedding-002 = 768D
    metadata JSONB,
    created_at TIMESTAMP DEFAULT NOW()
  );
  CREATE INDEX ON rag_chunks USING ivfflat (embedding vector_cosine_ops)
    WITH (lists = 100);  -- ivfflat pour ANN rapide

═══════════════════════════════════════════════════════════════════════════
```

1.3 PROMPT — Section C : Code Python complet à générer

Génère le code Python complet pour les modules suivants.

Chaque module doit être un fichier Python autonome et testé.

MODULE 1 : config.py — Configuration centralisée

- Classe Settings avec pydantic-settings
- Variables : GCP_PROJECT, GCP_REGION, VERTEX_MODEL='gemini-1.5-flash', EMBED_MODEL='text-multilingual-embedding-002', DB_URL (Cloud SQL), GROQ_API_KEY (fallback), TOP_K=5, CHUNK_SIZE=400, CHUNK_OVERLAP=80, SIMILARITY_THRESHOLD=0.65, MAX_TOKENS_RESPONSE=1024
- Lecture depuis Secret Manager ou .env local (dev)

MODULE 2 : embedder.py — Génération d'embeddings

- Fonction principale : get_embedding(text: str) → list[float]
- Primaire : Vertex AI text-multilingual-embedding-002 via google-cloud-aiplatform
- Fallback auto si RateLimitError : sentence-transformers/all-MiniLM-L6-v2 local
- Fonction batch : get_embeddings_batch(texts: list[str]) → list[list[float]]
- Gestion retry avec exponential backoff (tenacity)
- Log du provider utilisé à chaque appel

MODULE 3 : chunker.py — Découpage sémantique

- Fonction : chunk_document(text, source_meta) → list[ChunkModel]
- Modèle Pydantic ChunkModel : chunk_id, content, metadata (discipline, cycle, theme, niveau, source, langue, version)
- Séparateurs adaptés au markdown : [\n## , \n### , \n\n, \n, . ,]
- Génération chunk_id : '{source_id}_chunk_{index:04d}'
- Extraction automatique tags depuis le contenu (regex sur mots-clés)

MODULE 4 : indexer.py — Ingestion dans pgvector

- Connexion Cloud SQL via cloud-sql-python-connector

1.4 PROMPT — Section D : Prompt système du LLM

```
Le prompt système suivant doit être injecté dans chaque appel au LLM.
C'est le 'cerveau' du RAG — il garantit la qualité et la traçabilité des réponses.

SYSTEM_PROMPT = '''
Tu es un assistant expert en planification de l'entraînement en demi-fond
(800m, 1500m, 3000m, 3000m steeple, 5000m).

IDENTITÉ : Tu t'appelles ATHL-IA. Tu assistes les entraîneurs FFA dans leur
planification quotidienne. Tu es rigoureux, précis et pédagogue.

RÈGLES STRICTES :
1. SOURCE OBLIGATOIRE : Cite [CHUNK DMFD_XXX] après chaque affirmation factuelle.
2. CALCULS PRÉCIS : Pour toute allure, donne TOUJOURS km/h ET min:ss/km.
   Formule : Allure_kmh = VMA × pourcentage. Min/km = 60 / allure_kmh.
3. ADAPTATION NIVEAU : Adapte la réponse au profil athlète fourni.
4. CONTRE-INDICATIONS : Signale systématiquement les risques (âge, blessure).
5. INCERTITUDE : Si le contexte est insuffisant, dis CLAIREMENT ce que tu
   ne sais pas et propose 2-3 questions de clarification.
6. FORMAT RÉPONSE (OBLIGATOIRE) :
   ---
   [RÉPONSE] : Contenu principal structuré
   [SOURCES] : Liste des chunk_ids utilisés
   [ATTENTION] : Contre-indications / points de vigilance (ou 'Aucun')
   [CONFIANCE] : XX% — N sources utilisées — Qualité: Haute/Moyenne/Faible
   ---

PROFIL ATHLÈTE (contexte dynamique) :
{athlete_profile}

CONTEXTE RAG (chunks récupérés par similarité sémantique) :
{retrieved_context}

QUESTION DE L'ENTRAÎNEUR :
```

1.5 PROMPT — Section E : Score de confiance et fallback

```
Implémente le calcul du score de confiance composite et la logique de
fallback LLM.


SCORE DE CONFIANCE (0-100%) :
─────────────────────────────────────────────────────────────

score = (
  0.40 × max(chunk.score for chunk in chunks)     # Meilleure similarité
cosine
+ 0.20 × mean(chunk.score for chunk in chunks[:5]) # Cohérence des sources
+ 0.20 × coverage_score(query, chunks)             # Thèmes couverts /
requis
+ 0.10 × freshness_score(chunks)                   # Fraîcheur des sources
+ 0.10 × 0.80                                      # Bonus LLM bien
paramétré
) × 100


Labels : >= 75 → 'Haute (citer sans réserve)'
         50-75 → 'Moyenne (valider avec expertise)'
         < 50  → 'Faible (contexte insuffisant, reformuler la question)'


LOGIQUE FALLBACK LLM (à implémenter dans generator.py) :
─────────────────────────────────────────────────────────────

def get_llm_client():
    try:
        # 1. Tentative Vertex AI Gemini Flash (GRATUIT)
        import vertexai
        vertexai.init(project=settings.GCP_PROJECT,
location=settings.GCP_REGION)
        return 'vertex', GenerativeModel('gemini-1.5-flash')
    except Exception:
        # 2. Fallback Groq (GRATUIT 14400 req/j)
        from groq import Groq
        return 'groq', Groq(api_key=settings.GROQ_API_KEY)


LOGIQUE FALLBACK EMBEDDING :
─────────────────────────────────────────────────────────────
```

2. PROMPTS SPÉCIALISÉS — CLOUD SHELL ET AI STUDIO

2.1 Prompt setup infrastructure GCP (Cloud Shell)

Coller directement dans Google Cloud Shell après connexion au projet GCP :

```
# ═══════════════════════════════════════════════════════════════════
# SETUP RAG DEMI-FOND — GCP CLOUD SHELL — Exécuter dans l'ordre
# ═══════════════════════════════════════════════════════════════════

export PROJECT_ID='rag-demifond-poc'        # Remplacer par ton project ID
export REGION='europe-west1'                 # Région Europe (RGPD)
export DB_INSTANCE='rag-db-instance'
export DB_NAME='rag_demifond'

# 1. Créer le projet et activer les APIs nécessaires
gcloud projects create $PROJECT_ID --name='RAG Demi-Fond POC'
gcloud config set project $PROJECT_ID

gcloud services enable \
  run.googleapis.com \
  sqladmin.googleapis.com \
  aiplatform.googleapis.com \
  secretmanager.googleapis.com \
  storage.googleapis.com \
  cloudbuild.googleapis.com

# 2. Cloud SQL avec pgvector (e2-micro = GRATUIT dans les limites free tier)
gcloud sql instances create $DB_INSTANCE \
  --database-version=POSTGRES_15 \
  --tier=db-f1-micro \
  --region=$REGION \
  --storage-size=10GB \
  --no-backup

gcloud sql databases create $DB_NAME --instance=$DB_INSTANCE
gcloud sql users set-password postgres --instance=$DB_INSTANCE --prompt-for-
password

# 3. Activer pgvector (se connecter à l'instance)
gcloud sql connect $DB_INSTANCE --user=postgres --database=$DB_NAME
```

2.2 Prompt pour Google AI Studio — Test rapide RAG

Utilise ce prompt directement dans https://aistudio.google.com avec gemini-1.5-flash (GRATUIT). Il permet de valider le comportement du LLM avant intégration dans le code.

```
# COPIE CE PROMPT DANS GOOGLE AI STUDIO (System Instructions) :

Tu es ATHL-IA, assistant expert en entraînement demi-fond. Tu vas simuler
le comportement d'un système RAG pour valider le prompt avant intégration.

Pour cette session de test, tu as accès aux connaissances suivantes
(simule une base RAG avec ces informations clés) :

CHUNK DMFD_004 — Zones d'intensité :
Z5 (VMA) = 90-100% de la VMA. Pour VMA 20 km/h : Z5 = 18.0-20.0 km/h.
Z4 (Seuil) = 83-90% VMA. Pour VMA 20 km/h : 16.6-18.0 km/h.
Z6 (Supra-max) = >100% VMA. Séries lactiques pour 800m.

CHUNK DMFD_005 — Catalogue séances :
VMA-LONG : 4-6×1000m à 95-98% VMA, récup 2'30-3'. Phase PPS.
LACTI-LONG : 3-4×400-600m à allure 800m (104-108% VMA), récup 5-8min.
SEUIL-REP : 3-4×10min à 87% VMA, récup 2min.

CHUNK DMFD_008 — Règles microcycle :
Jamais 2 séances lactiques à moins de 48h. Séance clé = lendemain du repos.
Volume max +10% par semaine. Douleur tendineuse → arrêt immédiat.

RÉPONDS EN RESPECTANT CE FORMAT :
[RÉPONSE] : ...
[SOURCES] : DMFD_XXX, ...
[ATTENTION] : ...
[CONFIANCE] : XX% — N sources

Question test : 'Pour un athlète avec VMA 19 km/h spécialiste 800m,
propose une séance de capacité lactique avec les allures calculées.'
```

2.3 Prompt pour Vertex AI Workbench — Notebook de démarrage

À utiliser dans un notebook Jupyter sur Vertex AI Workbench (tier gratuit disponible) :

```
# Notebook Vertex AI Workbench — RAG Demi-Fond — Cellule 1 : Setup
# ================================================================

# Installation des dépendances
!pip install -q google-cloud-aiplatform langchain-google-vertexai \
             langchain pgvector cloud-sql-python-connector[pg8000] \
             sentence-transformers groq pydantic-settings fastapi

import vertexai
from vertexai.generative_models import GenerativeModel, Part
from vertexai.language_models import TextEmbeddingModel
import os

# Configuration (remplacer par vos valeurs)
PROJECT_ID = 'rag-demifond-poc'  # Votre project ID
REGION = 'europe-west1'

vertexai.init(project=PROJECT_ID, location=REGION)
print('✅ Vertex AI initialisé')

# ================================================================
# Cellule 2 : Test embedding GRATUIT (text-multilingual-embedding-002)
# ================================================================

embed_model = TextEmbeddingModel.from_pretrained('text-multilingual-
embedding-002')

test_text = 'Séance VMA 5×1000m à 95% VMA pour spécialiste 1500m'
embedding = embed_model.get_embeddings([test_text])[0]
print(f'Dimension embedding : {len(embedding.values)}')  # → 768
print(f'Extrait : {embedding.values[:5]}...')

# ================================================================
# Cellule 3 : Test Gemini Flash GRATUIT
# ----------------------------------------------------------------
```

3. PROMPTS ORCHESTRATEUR — AGENTS SPÉCIALISÉS (VERTEX AI)

Ces prompts sont utilisés pour chaque agent dans l'architecture Orchestrateur. Ils s'intègrent dans un workflow LangGraph ou directement dans Vertex AI Agent Builder.

3.1 Agent Planificateur — Génération de plans d'entraînement

```
# SYSTEM PROMPT — AGENT PLANIFICATEUR
# Modèle : gemini-1.5-flash (GRATUIT) ou gemini-1.5-pro (tokens d'essai)

Tu es l'Agent Planificateur du système ATHL-IA.
Ta mission : générer des plans d'entraînement personnalisés et cohérents.

INPUTS (fournis par l'Orchestrateur) :
  - profil_athlete: {nom, vma, distance_cible, niveau, categorie,
blessures_actives}
  - phase_saison: {type: 'PPG|PPS|COMP', semaine_num, objectif_saison}
  - contexte_rag: [liste de chunks DMFD récupérés par similarité sémantique]
  - contraintes: {nb_seances_semaine, volume_max_km, jours_indisponibles}

OUTPUTS OBLIGATOIRES (format JSON strict pour l'Orchestrateur) :
{
  'plan_semaine': [
    {'jour': 'Lundi', 'type': 'Repos', 'seance': null},
    {'jour': 'Mardi', 'type': 'VMA-LONG', 'seance': {
      'description': '5×1000m',
      'allure_kmh': 18.05,
      'allure_min_km': '3:19',
      'volume_km': 15.0,
      'rpe_cible': 8,
      'recuperation': '2min30 allure Z1'
    }},
    ...
  ],
  'volume_total_km': float,
  'intensite_semaine': 'Légère|Modérée|Élevée|Choc',
  'sources_chunks': ['DMFD_007', 'DMFD_008', ...],
  'confiance': float,
  'alertes': ['string'] ou []
}

RÈGLES DE PLANIFICATION :
```

3.2 Agent Analyste — Détection de surcharge

```
# SYSTEM PROMPT — AGENT ANALYSTE SURCHARGE
# Modèle : gemini-1.5-flash (GRATUIT)

Tu es l'Agent Analyste du système ATHL-IA.
Ta mission : analyser les bilans de séances et détecter les risques de surcharge.

INPUTS (fournis par l'Orchestrateur) :
  - historique_7j: [liste de 7 bilans de séances avec RPE, FC, douleurs, sommeil]
  - plan_prevu: [liste des séances planifiées vs réalisées]
  - valeurs_baseline: {fc_repos_ref, hrv_ref, rpe_moyen_baseline}
  - contexte_rag: [chunks DMFD_012 — indicateurs surcharge]

ALGORITHME D'ANALYSE (à appliquer) :
  1. Calculer ACWR = charge_aigue_7j / charge_chronique_28j
     charge = Σ(volume_km × RPE_session)
     ALERTE si ACWR > 1.4 ou < 0.8

  2. Analyser tendance FC repos :
     ALERTE si augmentation > 5 bpm sur 3 jours consécutifs

  3. Analyser score wellness moyen :
     ALERTE si score < 60% de la baseline sur 3 jours

  4. Analyser décrochage de performance :
     ALERTE si allure réelle > allure_cible + 5s/km sur 2 séances consécutives

OUTPUT (format JSON strict) :
{
  'risque_global': 'VERT|ORANGE|ROUGE',
  'acwr': float,
  'alertes': [{
    'type': 'ACWR|FC|WELLNESS|PERFORMANCE|DOULEUR',
    'severite': 'FAIBLE|MODEREE|HAUTE',
```

3.3 Prompt LangGraph — Orchestration multi-agents

```
# PROMPT POUR GÉNÉRER LE CODE LANGGRAPH COMPLET
# À utiliser dans AI Studio ou Vertex AI Workbench

Génère le code Python LangGraph complet pour orchestrer 3 agents RAG sur GCP.
Utilise EXCLUSIVEMENT les modèles Gemini GRATUITS (gemini-1.5-flash).

WORKFLOW À IMPLÉMENTER :

  START
    │
    ▼
  [Node: retrieve_context]
    Recherche pgvector, top-5 chunks, filtrage par discipline/niveau
    │
    ▼
  [Node: analyze_athlete]
    Calcul ACWR, lecture bilans 7 derniers jours, scoring risque
    │
    ├─ risque ROUGE ──→ [Node: alert_response] ──→ END
    │
    ▼ risque VERT/ORANGE
  [Node: generate_plan]
    Génération plan semaine personnalisé avec allures calculées
    │
    ▼
  [Node: format_response]
    Assemblage réponse finale avec sources + score confiance
    │
    ▼
  END

CONTRAINTES TECHNIQUES :
  - State : TypedDict avec tous les champs nécessaires
  - LLM : ChatVertexAI(model='gemini-1.5-flash', temperature=0.2)
```

4. RÉCAPITULATIF — MODÈLES GRATUITS GCP (FREE TIER 2025)

Tableau de référence des services GCP utilisables gratuitement pour le POC. Valeurs vérifiées en mai 2025 — consulter cloud.google.com/free pour les mises à jour.

Service GCP	Modèle / SKU	Limite gratuite	Limite tokens/ jeu d'essai	Usage recommandé POC
Vertex AI — Gemini	gemini-1.5-flash	15 req/min · 1M tokens/min · 1500 req/ jour	Tokens d'essai disponibles via console GCP	LLM principal du Copilote et agents Orchestrateur
Vertex AI — Gemini	gemini-1.5-pro	2 req/min · 32K tokens/ min · 50 req/jour	Plus généreux avec tokens d'essai	Requêtes complexes uniquement (génération plan complet)
Vertex AI — Embeddings	text-multilingual-embedding-002	250 req/min · Gratuit	Non limité par tokens d'essai	Embedding de TOUS les chunks et requêtes utilisateurs
Vertex AI — Embeddings	text-embedding-004 (EN)	250 req/min · Gratuit	Non limité	Alternative si corpus majoritairement en anglais
Cloud SQL — PostgreSQL	db-f1-micro (shared)	Gratuit : 1 instance, 10GB SSD	—	Base pgvector pour les chunks (Copilote)
Cloud Run	Instances 256MB-512MB	Gratuit : 2M req/mois, 360K CPU-s/mois	—	API FastAPI et jobs d'ingestion
Cloud Storage (GCS)	Standard — Multi-régional	Gratuit : 5GB stockage, 1GB sortie/mois	—	Stockage des documents sources (PDF, markdown)
Secret Manager	—	Gratuit : 6 secrets actifs, 10K accès/mois	—	Stockage sécurisé clés API (Groq, HF)
Cloud Build	—	Gratuit : 120 min/jour de build	—	CI/CD pour déploiement Cloud Run
Groq API (externe)	llama3-8b-8192	GRATUIT : 14400 req/ jour · 30 req/min	—	Fallback LLM si quota Vertex dépassé
HF Inference API (ext.)	all-MiniLM-L6-v2	GRATUIT : 1000 req/h	—	Fallback embedding si Vertex indisponible

4.1 Comment activer les tokens d'essai GCP

1. Se connecter à console.cloud.google.com avec un compte Google.
2. Créer un nouveau projet GCP (ou utiliser un projet existant sans facturation active).
3. Naviguer vers 'Vertex AI' → 'Overview' → Accepter les conditions d'utilisation.
4. Les tokens d'essai Vertex AI sont automatiquement attribués aux nouveaux projets (~300$ de crédit pour 90 jours).
5. Activer l'API Vertex AI : gcloud services enable aiplatform.googleapis.com
6. Vérifier le quota disponible : console.cloud.google.com/iam-admin/quotas → filtrer par 'vertexai'.

7. Pour les modèles Gemini spécifiquement : vertex AI Studio → Modèles → Demander accès si nécessaire.
8. Configurer les alertes de budget : Facturation → Budgets → Créer une alerte à 0€ pour ne pas dépasser le free tier.

— Prompt d'implémentation GCP RAG Demi-Fond v1.0 —