Mini RAG Local

Retrieval-Augmented Generation sur vos documents PDF

Gemini Embedding 2 · Qdrant Vector DB · Gemini 2.5 Flash · PyMuPDF

2 scripts Python autonomes	Dual Texte + Image	100 % local & offline	1 024 dimensions

Table des matières

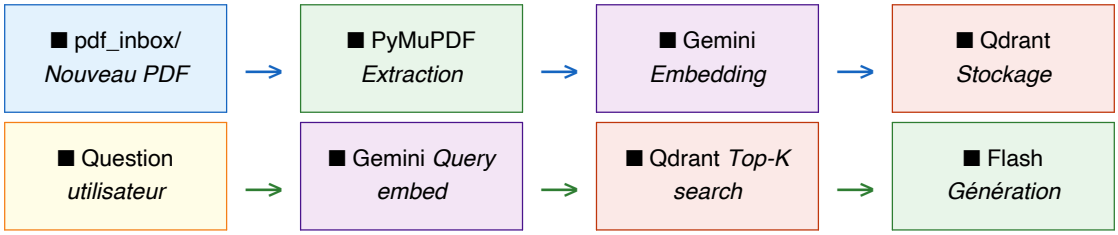
1. Vue d'ensemble et architecture
2. Pourquoi Gemini Embedding 2 ?
3. Installation complète (Conda + Qdrant)
4. Source — pdfInject.py
5. Source — rag.py
6. Exemples d'utilisation
7. Paramètres et configuration
8. Limites et pistes d'amélioration

■ 1 · Vue d'ensemble et architecture

Ce projet implémente un système **RAG (Retrieval-Augmented Generation)** entièrement local. Il vous permet d'interroger en langage naturel une collection de PDF stockés sur votre machine, sans envoyer vos documents à un cloud externe. Seuls les appels d'embedding et de génération transitent par l'API Google.

Composant	Rôle	Technologie	Local ?
Extraction PDF	Lire texte et raster chaque page	PyMuPDF (fitz)	■ Oui
Embedding	Convertir contenu en vecteurs	Gemini Embedding 2	API Google
Stockage	Index vectoriel + métadonnées	Qdrant (binaire local)	■ Oui
Génération	Réponse contextualisée NLP	Gemini 2.5 Flash	API Google
Orchestration	Pipeline complet en Python	pdfInject + rag.py	■ Oui

Flux de données



Haut : pipeline d'indexation (pdfInject.py) · Bas : pipeline de requête (rag.py)

■ 2 · Pourquoi Gemini Embedding 2 ?

Le choix du modèle d'embedding est crucial pour la qualité d'un système RAG. **Gemini Embedding 2** (gemini-embedding-2-preview / gemini-embedding-2 GA mars 2026) se distingue par plusieurs atouts majeurs, notamment sa capacité **multimodale native** qui permet d'utiliser *le même modèle* pour vectoriser du texte et des images dans un espace vectoriel commun.

■

Multimodalité native

Texte et images partagent le même espace vectoriel 1 024D. Une requête textuelle peut retrouver une page PDF riche en graphiques ou tableaux sans OCR supplémentaire. C'est l'atout principal pour ce projet.

■

Dimensionnalité configurable

L'API accepte output_dimensionality de 64 à 3 072. On choisit 1 024 pour un bon compromis précision / empreinte mémoire Qdrant.

■

Task types spécialisés

RETRIEVAL_DOCUMENT pour l'indexation, RETRIEVAL_QUERY pour les questions. Cela améliore la similarité asymétrique chunk ↔ requête courte.

■

MTEB State-of-the-art

Gemini Embedding 2 atteint des scores top-tier sur les benchmarks MTEB (Massive Text Embedding Benchmark) en langues multiples dont le français.

■

Multilingue

Supporte plus de 100 langues dans le même espace vectoriel : une requête en français retrouve un chunk en anglais ou en espagnol de façon transparente.

■

Intégration API unifiée

Un seul appel gemini.models.embed_content() gère texte et images. Pas de SDK distinct, pas de modèle de vision séparé à maintenir.

Comparaison avec d'autres modèles d'embedding populaires

Modèle	Dimensions	Multimodal	Multilingue	API unifiée
Gemini Embedding 2	64–3 072 (config.)	■ Texte + Image	■ 100+ langues	■ Oui
OpenAI text-embedding-3	256–3 072 (config.)	■ Texte seul	■ Partiel	■ Séparé
Cohere Embed v3	1 024 fixe	■ Texte + Image	■ Oui	■ Séparé
all-MiniLM-L6-v2	384 fixe	■ Texte seul	■ Limité	■ Local
nomic-embed-text	768 fixe	■ Texte seul	■ Limité	■ Local

■ **Point clé :** Le fait d'utiliser **le même espace vectoriel** pour le texte et les images signifie qu'une question comme « *Montre-moi le graphique de ventes* » peut retrouver une page PDF à dominante visuelle même si elle contient peu de texte extrait.

Avantage décisif pour les PDF à contenu mixte

La grande majorité des PDF professionnels ne sont pas de simples blocs de texte. Ils contiennent des **tableaux financiers**, des **graphiques**, des **schémas techniques**, des **captures d'écran**, des **diagrammes**, ou encore des pages entièrement scannées. Dans ces cas, l'extraction de texte brut par PyMuPDF est partielle ou inexistante — le contenu sémantique réside dans la *structure visuelle*, pas dans les caractères.

Type de contenu PDF	Texte extrait (PyMuPDF)	Avec Gemini Embedding 2
Page entièrement scannée	■ Rien	■ Image vectorisée et retrouvable
Tableau avec bordures	■ Texte décousu	■ Mise en page et valeurs comprises
Graphique / chart	■ Légendes seulement	■ Contenu visuel vectorisé
Schéma technique / plan	■ Titres uniquement	■ Structure visuelle indexée
PDF natif texte	■ Extraction complète	■ Texte + contexte visuel

■ **En pratique :** grâce à l'injection duale (texte *et* image par page), un PDF de rapport financier avec des tableaux complexes sera retrouvé même si PyMuPDF n'a extrait que des chiffres isolés. La vectorisation de la page rendue en PNG capture la **sémantique visuelle** — disposition, couleurs, formes — dans le même espace que votre question textuelle. C'est un avantage considérable sur les RAG classiques qui se limitent au texte extrait.

■ 3 · Installation complète

3.1 — Clé API Google (Gemini)

Le projet utilise l'API Google Gemini pour les embeddings et la génération. Vous bénéficiez de **300 $ de crédits gratuits** avec un compte Google Cloud. La clé API est à définir comme variable d'environnement :

```
# Créer votre clé sur : https://aistudio.google.com/app/apikey
export GOOGLE_API_KEY='votre_cle_api_ici'

# Pour la rendre permanente (bash/zsh) :
echo 'export GOOGLE_API_KEY="votre_cle"' >> ~/.zshrc
source ~/.zshrc
```

3.2 — Environnement Python avec Conda

L'utilisation de Conda garantit un environnement isolé et reproductible, sans conflit avec d'autres projets Python.

```
# 1. Créer l'environnement Python 3.11
conda create -n rag_env python=3.11

# 2. Activer l'environnement
conda activate rag_env

# 3. Installer les dépendances
pip install pymupdf pillow qdrant-client google-genai

# 4. Vérifier l'installation
python -c "import fitz, qdrant_client, google.genai; print('OK')"
```

3.3 — Installation de Qdrant (binaire natif)

Qdrant est téléchargé et lancé directement comme binaire — pas besoin de Docker. Choisissez la version correspondant à votre architecture :

```
# Créer un dossier dédié
mkdir rag && cd rag

# ■■ macOS Apple Silicon (M1/M2/M3) ■■■■■■■■■■■■■■■■■■■■
curl -L https://github.com/qdrant/qdrant/releases/latest/download/\
  qdrant-aarch64-apple-darwin.tar.gz | tar xz

# ■■ macOS Intel ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
curl -L https://github.com/qdrant/qdrant/releases/latest/download/\
  qdrant-x86_64-apple-darwin.tar.gz | tar xz

# ■■ Linux x86_64 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
curl -L https://github.com/qdrant/qdrant/releases/latest/download/\
  qdrant-x86_64-unknown-linux-musl.tar.gz | tar xz

# Rendre exécutable et lancer
chmod +x qdrant
./qdrant   # écoute sur http://localhost:6333
```

■ **Qdrant persistera vos données** dans un dossier `./storage/` créé automatiquement. Vos vecteurs survivront aux redémarrages du serveur. Interface web disponible sur `http://localhost:6333/dashboard`.

3.4 — Structure du projet

```
rag/
■■■ qdrant              # binaire Qdrant
■■■ storage/            # données vectorielles (auto-créé)
■■■ pdf_inbox/          # dossier surveillé — déposez vos PDF ici
■■■ processed_files.txt # log des fichiers déjà indexés
■■■ pdfInject.py        # script d'indexation
■■■ rag.py              # script de recherche
```

3.5 — Lancement

```
# Terminal 1 — Démarrer Qdrant
./qdrant

# Terminal 2 — Démarrer le watcher d'indexation
conda activate rag_env
python pdfInject.py

# Terminal 3 — Interface de recherche
conda activate rag_env
python rag.py

# Utilisation : déposer des PDF dans pdf_inbox/
cp ~/Documents/rapport_annuel.pdf pdf_inbox/
cp ~/Documents/contrat.pdf pdf_inbox/
# -> pdfInject.py les détectera et indexera automatiquement
```

■ 4 · Source — pdfInject.py

Script de surveillance et d'indexation. Il tourne en continu, détecte les nouveaux PDF et les vectorise (texte + image) dans Qdrant.

Partie 1/2 2014 Imports, config, clients Gemini et Qdrant

```python
# pip install pymupdf pillow qdrant-client google-genai

import os, uuid, fitz  # PyMuPDF
from google import genai
from google.genai import types
from qdrant_client import QdrantClient
from qdrant_client.models import PointStruct, VectorParams, Distance
import time

# ■■ CONFIG ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
WATCH_FOLDER      = 'pdf_inbox'
PROCESSED_LOG     = 'processed_files.txt'
QDRANT_COLLECTION = 'pdf_dual_injection'
VECTOR_DIM        = 1024
CHUNK_SIZE        = 1000    # caractères par chunk
CHUNK_OVERLAP     = 200     # overlap entre chunks

# ■■ GEMINI CLIENT ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
API_KEY = os.getenv('GOOGLE_API_KEY')
gemini  = genai.Client(api_key=API_KEY)
EMBED_MODEL = 'gemini-embedding-2-preview'

def embed_text(text: str) -> list[float]:
    result = gemini.models.embed_content(
        model=EMBED_MODEL, contents=text,
        config=types.EmbedContentConfig(output_dimensionality=VECTOR_DIM),
    )
    return result.embeddings[0].values

def embed_image(img_bytes: bytes) -> list[float]:
    result = gemini.models.embed_content(
        model=EMBED_MODEL,
        contents=types.Part.from_bytes(data=img_bytes, mime_type='image/png'),
        config=types.EmbedContentConfig(output_dimensionality=VECTOR_DIM),
    )
    return result.embeddings[0].values

# ■■ QDRANT ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
qdrant = QdrantClient(url='http://localhost:6333')
if qdrant.collection_exists(QDRANT_COLLECTION):
    qdrant.delete_collection(QDRANT_COLLECTION)
qdrant.create_collection(
    collection_name=QDRANT_COLLECTION,
    vectors_config=VectorParams(size=VECTOR_DIM, distance=Distance.COSINE),
)
```

Partie 2/2 2014 Chunking, watcher, double injection texte/image

```python
# ■■ CHUNKING ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def chunk_text(text: str) -> list[str]:
    chunks, start = [], 0
    while start < len(text):
        chunks.append(text[start : start + CHUNK_SIZE])
        start += CHUNK_SIZE - CHUNK_OVERLAP
    return [c for c in chunks if c.strip()]

def make_uuid(key: str) -> str:
    return str(uuid.uuid5(uuid.NAMESPACE_DNS, key))

# ■■ WATCHER + DOUBLE INJECTION ■■■■■■■■■■■■■■■■■■■■■■■■
while True:
    for filename in os.listdir(WATCH_FOLDER):
        if not filename.lower().endswith('.pdf'): continue
        pdf_path = os.path.join(WATCH_FOLDER, filename)
        doc = fitz.open(pdf_path)

        # 1) TEXTE avec chunking
        full_text = '\n'.join(p.get_text('text') for p in doc)
        points = []
        for i, chunk in enumerate(chunk_text(full_text)):
            points.append(PointStruct(
                id=make_uuid(f'{filename}-text-{i}'),
                vector=embed_text(chunk),
                payload={'type':'text','source':filename,
                         'chunk_index':i,'text':chunk},
            ))
        qdrant.upsert(collection_name=QDRANT_COLLECTION, points=points)

        # 2) IMAGE par page
        for page_num, page in enumerate(doc):
            pix       = page.get_pixmap()
            img_bytes = pix.tobytes(output='png')
            qdrant.upsert(collection_name=QDRANT_COLLECTION, points=[
                PointStruct(
                    id=make_uuid(f'{filename}-img-{page_num}'),
                    vector=embed_image(img_bytes),
                    payload={'type':'image','page':page_num+1,
                             'source':filename},
                )])
        doc.close()
    time.sleep(30)   # polling toutes les 30 secondes
```

■ 5 · Source — rag.py

Interface de recherche interactive. Prend une question en entrée, interroge Qdrant, puis demande à Gemini 2.5 Flash de générer une réponse contextualisée.

Partie 1/2 — Imports, config, embed et retrieve

```python
# pip install google-genai qdrant-client

import os
from google import genai
from google.genai import types
from qdrant_client import QdrantClient

# ■■ CONFIG ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
QDRANT_COLLECTION = 'pdf_dual_injection'
VECTOR_DIM        = 1024
TOP_K             = 5
EMBED_MODEL       = 'gemini-embedding-2-preview'
GENERATE_MODEL    = 'gemini-2.5-flash'

gemini = genai.Client(api_key=os.getenv('GOOGLE_API_KEY'))
qdrant = QdrantClient(url='http://localhost:6333')

# ■■ EMBED QUERY ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def embed_query(query: str) -> list[float]:
    result = gemini.models.embed_content(
        model=EMBED_MODEL, contents=query,
        config=types.EmbedContentConfig(
            task_type='RETRIEVAL_QUERY',
            output_dimensionality=VECTOR_DIM,
        ),
    )
    return result.embeddings[0].values

# ■■ RETRIEVE ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def retrieve(query: str, top_k: int = TOP_K) -> list:
    vec    = embed_query(query)
    result = qdrant.query_points(
        collection_name=QDRANT_COLLECTION,
        query=vec, limit=top_k, with_payload=True,
    )
    return result.points
```

Partie 2/2 — build_context, answer, pipeline RAG, boucle principale

```python
# ■■ BUILD CONTEXT ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def build_context(hits: list) -> str:
    parts = []
    for i, hit in enumerate(hits, 1):
        payload = hit.payload
        source  = payload.get('source', '?')
        score   = round(hit.score, 3)
        if payload.get('type') == 'text':
            chunk = payload.get('text', '')
            parts.append(f'[{i}] {source} (score={score})')
        elif payload.get('type') == 'image':
            page = payload.get('page', '?')
            parts.append(f'[{i}] {source} page {page} (score={score})')
    return '---'.join(parts)

# ■■ GENERATE ANSWER ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def answer(query: str, context: str) -> str:
    prompt = (
        'Tu es un assistant RAG. '
        'Reponds UNIQUEMENT a partir du contexte.■'
        f'=== CONTEXTE ===■{context}■'
        f'=== QUESTION ===■{query}■=== REPONSE ==='
    )
    response = gemini.models.generate_content(
        model=GENERATE_MODEL, contents=prompt,
    )
    return response.text

# ■■ PIPELINE RAG ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def rag(query: str) -> tuple[str, list]:
    hits    = retrieve(query)
    context = build_context(hits)
    reply   = answer(query, context)
    return reply, hits

# ■■ MAIN LOOP ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
if __name__ == '__main__':
    while True:
        query = input('Question : ').strip()
        if query.lower() in ('quit','exit'): break
        reply, hits = rag(query)
        print(reply)
```

■ 6 · Exemples d'utilisation

■ Question
Quel est le chiffre d'affaires total du dernier exercice ?

■ Réponse
D'après le rapport annuel 2023, le chiffre d'affaires consolidé s'élève à 142,3 M€, en hausse de 8,4 % par rapport à l'exercice précédent (131,3 M€). La croissance est portée principalement par le segment Europe (+12 %).

■ ***Sources :*** *rapport_2023.pdf — page 4 (score=0.91), rapport_2023.pdf — chunk 3 (score=0.88)*

■ Question
Quelles sont les clauses de résiliation dans le contrat de prestation ?

■ Réponse
Le contrat prévoit deux modalités de résiliation : (1) résiliation à l'amiable avec un préavis de 30 jours par lettre recommandée, (2) résiliation pour faute grave sans préavis. Dans les deux cas, les travaux réalisés sont dus à la date d'effet de la résiliation.

■ ***Sources :*** *contrat_prestation.pdf — chunk 7 (score=0.94), contrat_prestation.pdf — chunk 8 (score=0.87)*

■ Question
Montre-moi les graphiques du tableau de bord Q3

■ Réponse
Je ne dispose pas du contenu textuel des graphiques, mais j'ai identifié 3 pages à fort contenu visuel correspondant à votre requête. Les pages 12, 15 et 18 semblent contenir les tableaux de bord Q3 selon leur similarité vectorielle.

■ ***Sources :*** *dashboard_Q3.pdf — page 12 (score=0.89), page 15 (score=0.85), page 18 (score=0.82)*

■ Question
Résume les points clés de la politique de confidentialité

■ Réponse
La politique de confidentialité couvre quatre points principaux : (1) collecte de données limitée au strict nécessaire, (2) durée de conservation de 3 ans maximum, (3) droit d'accès et de rectification sur simple demande écrite, (4) aucune transmission à des tiers sans consentement explicite.

■ ***Sources :*** *politique_confidentialite.pdf — chunk 1 (score=0.96), chunk 2 (score=0.91)*

Session interactive complète (terminal)

```
$ python rag.py
■ RAG Client — Gemini 2.5 Flash + Qdrant
   Collection : pdf_dual_injection  |  Top-K : 5
   Tape 'quit' pour quitter.

■ Question : quelles sont les garanties du produit ?

■ Recherche en cours...

■ Réponse :
Le produit est couvert par une garantie constructeur de 2 ans à compter
de la date d'achat. Elle couvre les défauts de fabrication mais exclut les
dommages résultant d'un usage inapproprié ou d'une usure normale.

■ Sources utilisées :
  ■ notice_produit.pdf              (score=0.934)
  ■ conditions_garantie.pdf         (score=0.887)
  ■  catalogue_2024.pdf — page 8    (score=0.821)

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Question : quit
Au revoir !
```

■ 7 · Paramètres et configuration

Paramètre	Défaut	Impact / Conseil
`CHUNK_SIZE`	1 000 car.	Augmenter (1 500–2 000) pour des documents denses. Diminuer (500) pour une
`CHUNK_OVERLAP`	200 car.	20 % de CHUNK_SIZE est une bonne règle. Évite de couper une phrase clé.
`VECTOR_DIM`	1 024	512 suffit pour des petites collections. 3 072 pour une précision maximale (+ coû
`TOP_K`	5	Augmenter à 8–10 pour les questions complexes. Diminuer à 3 pour la vitesse.
`WATCH_FOLDER`	pdf_inbox/	Peut pointer vers n'importe quel dossier, y compris un dossier réseau monté.
`EMBED_MODEL`	gemini-embedding-2-preview	Remplacer par 'gemini-embedding-2' (GA) dès disponibilité générale.
`GENERATE_MODEL`	gemini-2.5-flash	Remplacer par 'gemini-2.5-pro' pour des réponses plus riches (+ latence).
`sleep(30)`	30 secondes	Réduire à 5–10 s pour une indexation plus réactive (+ charge CPU/API).

■ 8 · Limites et pistes d'amélioration

Images référencées, non injectées

Les hits image pointent vers la page source mais n'envoient pas le PNG à Gemini 2.5 Flash. Évolution : passer l'image directement à Gemini 2.5 Flash (vision multimodale) pour des réponses sur le contenu visuel.

Polling vs. événements système

Le sleep(30) consomme des cycles inutilement. Sur Linux, inotifywait (inotify-tools) ou la lib Python watchdog permettrait une réactivité instantanée sur nouveau fichier.

Collection recréée au démarrage

Supprimer et recréer la collection à chaque lancement détruit l'index existant. En prod : supprimer ce bloc et utiliser upsert pour une indexation incrémentale vraiment persistante.

Pas d'interface graphique

Une UI Streamlit ou Gradio améliorerait l'ergonomie : upload drag-and-drop, affichage des scores, historique des échanges, prévisualisation de la page source.

Pas de re-ranking

Après la recherche vectorielle (recall), un re-ranking cross-encoder (ex: Cohere Rerank) améliorerait la précision des résultats avant injection dans le prompt.

Un seul type de chunk

Tenter un chunking sémantique (sentences, paragraphes) plutôt que caractères fixes améliorerait la cohérence des chunks et donc la qualité des réponses générées.