Parcours de la baseline

Milliman × Université Gustave Eiffel
4–5 juin 2026 · 36h · 2 jours

Milliman

Université Gustave Eiffel

Parcourons la baseline ensemble

Comprendre le code pour mieux l'optimiser

OBJECTIF DES 30 PROCHAINES MINUTES

- Comprendre l'architecture de la baseline fournie
- Identifier les points d'entrée pour vos modifications
- Suivre le pipeline complet en traçant une question
- Repérer vos leviers d'optimisation pour les 36h

L'architecture de la baseline

4 composants, 5 briques, 1 endpoint critique

1. **Question reçue**
   endpoint POST /query
2. **Recherche**
   passages dans le corpus
3. **Génération LLM**
   réponse + sources
4. **Réponse formatée**
   format strict imposé

Hugging Face Space

C'est l'endpoint que notre système d'évaluation appelle.

Le format de réponse est strict.

Le moindre écart = score 0.

Ce que vous trouvez dans le repo

6 fichiers à connaître, et lesquels modifier

`app.py`
Le cœur : API + interface + pipeline RAG

`llm.py`
Wrapper LLM avec mesure tokens + CO_2

`prompts/rag_prompt.txt`
Prompt RAG, modifiable sans toucher au code

`config.json`
Endpoints et modèles LLM

`ingest_train_data.py`
Ingestion des 150+ mémoires

`requirements.txt`
Vos dépendances Python

CONSEIL

Modification ciblée, pas au global.

Changer le prompt ?
→ `prompts/rag_prompt.txt`

Le routing LLM ?
→ `llm.py`

Le retrieval ?
→ `app.py`

Que se passe-t-il quand on pose une question ?

Suivons une requête de bout en bout

1. **Réception** : POST /query reçoit la question
2. **Embedding** : la question est embeddée (Azure OpenAI)
3. **Retrieval** : ChromaDB renvoie les top-3 chunks
4. **Prompt** : contexte + question assemblés
5. **Génération** : appel LLM → réponse
6. **Mesure** : parsing JSON + tokens + CO_2
7. **Retour** : réponse renvoyée au client

À RETENIR

Chaque étape est un point où vous pouvez optimiser.

Connaître ce flux = savoir où injecter vos améliorations.

`rag_query()`

dans app.py

C'est la fonction qui orchestre tout le pipeline.

Étape clé : retrouver le bon contexte

La qualité du retrieval conditionne la qualité de la réponse

```
retrieve_relevant_context(query, top_k=3)
```

Comment ça marche

La question est embeddée avec `text-embedding-3-small`

Le **MÊME modèle** que pour l'ingestion (essentiel !)

Recherche par similarité cosinus dans ChromaDB

Renvoie les top-3 chunks + leur score de relevance

Paramètres par défaut (tous modifiables)

```
CHUNK_SIZE = 512
CHUNK_OVERLAP = 50
TOP_K_RESULTS = 3
```

PREMIER LEVIER

Top-3, c'est peu ?

Vous pouvez essayer :

- Top-5, top-10
- Top-K adaptatif selon la question
- Retrieval hybride (BM25 + dense)
- Reranker après le retrieval

Construction du prompt & appel LLM

Deux étapes, deux fichiers à connaître

1. CONSTRUCTION DU PROMPT

`build_rag_prompt(query, contexts)`

- Colle les chunks récupérés dans un template
- Template externalisé dans `prompts/rag_prompt.txt`
- Modifiable sans toucher au code Python

2. APPEL LLM AVEC MESURES

`call_llm_with_metrics()`

dans llm.py

- Appel Azure OpenAI (GPT-5.1 par défaut)
- Récupère : prompt_tokens, completion_tokens, cached_tokens
- Calcule l'empreinte CO_2 via **`ecologits`**

Le contrat d'API à respecter absolument

La slide la plus importante de toutes

⚠ Si l'évaluation ne peut pas parser votre réponse → SCORE = 0

POST /query — format de réponse

```
{
  "answer": "...",              // votre réponse
  "sources": [...],            // chunks utilisés
  "explanation": "...",         // raisonnement
  "total_token": 1200,          // tokens consommés
  "prompt_tokens": 1120,
  "completion_tokens": 80,
  "cached_tokens": 0,
  "co2_grams": 0.04,            // via ecologits
  "energy_kwh": 0.0001,
  "run_time_in_ms": 1800
}
```

LES 3 RÈGLES

1. **Ne renommez pas les champs**
2. **Ne supprimez pas les champs**
3. **Vous pouvez en ajouter, jamais en retirer**

Vos 30 premières minutes

Mini-checklist pour démarrer

1. Forkez le Space template
2. Remplissez team.yml avec votre équipe
3. Ajoutez AZURE_API_KEY comme Space Secret
4. Lancez hf sync pour récupérer les 120 mémoires
5. Vérifiez que GET /health répond
6. Posez une question dans la Gradio UI
7. Vérifiez : réponse + tokens + CO_2

✓ SI ÇA MARCHE

Vous êtes opérationnels.

Vous pouvez attaquer l'optimisation sereinement.

AVANT TOUTE CHOSE

Lisez le README en entier.

Où trouver cette présentation : https://curiousml.github.io/

Tout en bas de la page