

Parcours Makers - Module 2 - Introduction au RAG

Cours 2 : Mesurer et améliorer la qualité RAG avec RAGAS

Ce notebook vous guide dans l'évaluation systématique et l'optimisation d'un système RAG (Retrieval-Augmented Generation) en utilisant le framework RAGAS. Vous apprendrez à mesurer la qualité de votre système selon plusieurs métriques objectives et à optimiser ses paramètres pour améliorer ses performances.

Objectifs pédagogiques : À l'issue de ce cours, vous serez capable de :

- Mettre en œuvre un protocole d'évaluation continue avec RAGAS
- Générer automatiquement un jeu de données de validation
- Calculer et interpréter les métriques de qualité RAG (`context_precision`, `answer_faithfulness`, `answer_relevancy`)
- Ajuster les paramètres top-k et taille de chunk pour optimiser les métriques
- Analyser les compromis entre différentes configurations

Métriques d'évaluation RAG : Ce notebook se concentre sur trois métriques fondamentales pour évaluer un système RAG :

- **Context Precision :** Mesure la précision des documents récupérés par rapport à la question posée. Une valeur élevée indique que les chunks récupérés sont pertinents.
- **Answer Faithfulness :** Évalue la fidélité de la réponse générée aux documents sources. Cette métrique détecte les hallucinations en vérifiant que la réponse s'appuie bien sur le contexte fourni.
- **Answer Relevancy :** Mesure la pertinence de la réponse par rapport à la question originale. Une réponse peut être fidèle aux sources mais ne pas répondre à la question posée.

Cas d'usage : Optimisation du Service Client TechStore En continuité avec le notebook précédent, nous utiliserons le même système de service client pour TechStore. L'objectif est d'optimiser les paramètres du système RAG pour améliorer l'expérience utilisateur et réduire les réponses inappropriées.

✓ 1. Préparation de l'environnement

✓ 1.1. Installation des dépendances

```
from IPython.utils import io

# Installation complète pour RAG et évaluation avec versions fixées

!pip install \
    google-api-python-client \
    google-auth-http2 \
    google-auth-oauthlib \
    langchain==0.3.26 \
    chromadb==1.0.15 \
    sentence-transformers==4.1.0 \
    langchain-community==0.3.27 \
    openai==1.97.1 \
    ragas==0.3.0 \
    datasets==4.0.0 \
    nltk==3.9.1 \
    pandas==2.2.2 \
    matplotlib==3.10.0 \
    seaborn==0.13.2 \
    plotly==5.24.1

print("✅ Installations terminées avec versions fixées.")
```

✓ 1.2. Imports des bibliothèques

Cette cellule importe toutes les bibliothèques nécessaires : celles pour construire le système RAG (reprises du notebook précédent) et celles spécifiques à l'évaluation et à l'optimisation.

```
# Imports pour le système RAG (repris du notebook précédent)
from langchain.document_loaders import TextLoader
```

```

from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.schema import Document
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma
from langchain.llms import HuggingFacePipeline
from langchain.chains import RetrievalQA
from langchain.prompts import PromptTemplate

```

```

# Imports pour l'évaluation RAGAS (nouveaux pour ce notebook)
from ragas import evaluate
from ragas.metrics import (
    context_precision,
    faithfulness,
    answer_relevancy
)

```

```

# Imports pour la génération de données et l'analyse
from datasets import Dataset
import pandas as pd
import numpy as np
from collections import Counter

```

```

# Imports pour la visualisation des résultats d'optimisation
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

```

```

# Imports système et Google Drive (repris du notebook précédent)
import os
import shutil
import tempfile
from googleapiclient.discovery import build
from google.colab import auth
import io
from googleapiclient.http import MediaIoBaseDownload

```

```

# Imports pour le modèle de génération OpenAI
from langchain.llms import OpenAI
from langchain.chat_models import ChatOpenAI
import os
from getpass import getpass

```

```

# Configuration de l'affichage pour les graphiques
plt.style.use('default')
sns.set_palette("husl")

```

```

print("✅ Imports terminés avec succès")

```

```

🔄 ✅ Imports terminés avec succès

```

```

import os
from getpass import getpass
from dotenv import load_dotenv

```

```

load_dotenv() # charge un éventuel .env
if "OPENAI_API_KEY" not in os.environ: # invite si la variable manque
    os.environ["OPENAI_API_KEY"] = getpass("🔑 Entrez votre clé OpenAI : ")

```

```

🔄 🔑 Entrez votre clé OpenAI : .....

```

✓ 1.3. Authentification Google Drive

```

# Authentification pour l'API Google Drive
print("Configuration de l'authentification...")
print("⚠️ ATTENTION : Une popup d'autorisation Google va s'afficher")
print("Instructions d'authentification :")
print("  1. Cliquez sur 'Autoriser' dans la popup")
print("  2. Sélectionnez votre compte Google")
print("  3. Acceptez les permissions demandées")
print("  4. Attendez la confirmation d'authentification")
print("-" * 50)

```

```

# Authentification pour l'API Google Drive
auth.authenticate_user()
service = build('drive', 'v3')

```

```

def download_file_from_drive(file_id, file_name):
    """
    Télécharge un fichier depuis Google Drive en utilisant son ID.
    """

```

```

"""
try:
    # Requête pour télécharger le fichier
    request = service.files().get_media(fileId=file_id)

    # Buffer pour stocker le contenu du fichier
    file_buffer = io.BytesIO()
    downloader = MediaIoBaseDownload(file_buffer, request)

    # Téléchargement progressif
    done = False
    while done is False:
        status, done = downloader.next_chunk()

    # Décodage du contenu en UTF-8
    file_content = file_buffer.getvalue().decode('utf-8')
    return file_content

except Exception as e:
    print(f"Erreur lors du téléchargement de {file_name}: {str(e)}")
    return None

def load_documents_from_drive_ids(doc_config):
    """
    Charge tous les documents depuis Google Drive en utilisant leurs IDs.
    """
    documents = []



    for doc_config in document_config:
        print(f"Chargement de {doc_config['name']}")

        # Téléchargement du contenu du fichier
        content = download_file_from_drive(doc_config['id'], doc_config['name'])

        if content:
            # Création du document LangChain avec métadonnées enrichies
            doc = Document(
                page_content=content,
                metadata={
                    "source": doc_config['source'],
                    "name": doc_config['name'],
                    "drive_id": doc_config['id'] # Stockage de l'ID pour traçabilité
                }
            )
            documents.append(doc)
            print(f"✓ Document chargé avec succès : {doc_config['name']}")
        else:
            print(f"✗ Échec du chargement : {doc_config['name']}")

    return documents

```

 Configuration de l'authentification...
 **ATTENTION** : Une popup d'autorisation Google va s'afficher
 Instructions d'authentification :
 1. Cliquez sur 'Autoriser' dans la popup
 2. Sélectionnez votre compte Google
 3. Acceptez les permissions demandées
 4. Attendez la confirmation d'authentification

✓ 1.4. Chargement des documents

```

# Configuration des documents avec leurs IDs Google Drive
document_config = [
    {
        "id": "1iwCq1qb7xNiRp_fK_jUsKhr1QHGeKamR", # ID du fichier politique_de_retour_et_remboursement.md
        "source": "politique_de_retour_et_remboursement.md",
        "name": "Politique de retour et remboursement"
    },
    {
        "id": "17dk0unV_GGlnjBQcUzsQpUTkg7sX8Z7G", # ID du fichier garantie_et_service_apres_vente.md
        "source": "garantie_et_service_apres_vente.md",
        "name": "Garantie et service après-vente"
    },
    {
        "id": "1xzEuVU_1P56b5Y0hPHeGS2zZUQ_9jJ0L", # ID du fichier livraison_et_expédition.md
        "source": "livraison_et_expédition.md",
        "name": "Livraison et expédition"
    },
    {
        "id": "13CZ5wujmKcEmFGcLqWjHNX8j7R7EM_CC", # ID du fichier compte_client_et_gestion_des_commandes.md

```

```

        "source": "compte_client_et_gestion_des_commandes.md",
        "name": "Compte client et gestion des commandes"
    },
    {
        "id": "1DPDv1P4QuRBG-KMpth6Dn3fz0v9Edw9h", # ID du fichier produits_et_compatibilite.md
        "source": "produits_et_compatibilite.md",
        "name": "Produits et compatibilité"
    }
]

```

```
documents = load_documents_from_drive_ids(document_config)
```

```

🔄 Chargement de Politique de retour et remboursement
✓ Document chargé avec succès : Politique de retour et remboursement
Chargement de Garantie et service après-vente
✓ Document chargé avec succès : Garantie et service après-vente
Chargement de Livraison et expédition
✓ Document chargé avec succès : Livraison et expédition
Chargement de Compte client et gestion des commandes
✓ Document chargé avec succès : Compte client et gestion des commandes
Chargement de Produits et compatibilité
✓ Document chargé avec succès : Produits et compatibilité

```

✓ 2. Construction du système RAG baseline

Cette section reprend le code du notebook précédent pour construire un système RAG complet. Elle sert de baseline pour nos expérimentations d'optimisation. Tous les composants (découpage, embeddings, base vectorielle, modèle de génération, chaîne RAG) sont configurés avec des paramètres par défaut raisonnables.

```

def create_baseline_rag_system(documents, chunk_size=1000, chunk_overlap=200, top_k=3):
    """
    Crée un système RAG baseline avec les paramètres par défaut.
    Cette fonction reprend et encapsule le code du notebook précédent.

    Args:
        documents (list): Documents source
        chunk_size (int): Taille des chunks (par défaut du notebook précédent)
        chunk_overlap (int): Chevauchement entre chunks
        top_k (int): Nombre de documents à récupérer

    Returns:
        tuple: (vectorstore, qa_chain, retriever, chunks, embeddings, llm)
    """

    print("Construction du système RAG baseline...")
    print("(Code repris du notebook précédent pour autonomie)")

    # 1. Configuration du text splitter (paramètres du notebook précédent)
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap,
        separators=["\n\n", "\n", ". ", " ", ""],
        length_function=len
    )

    # 2. Découpage des documents
    chunks = text_splitter.split_documents(documents)
    print(f"✓ {len(chunks)} chunks créés")

    # 3. Configuration du modèle d'embeddings (identique au notebook précédent)
    embedding_model_name = "sentence-transformers/all-MiniLM-L6-v2"
    embeddings = HuggingFaceEmbeddings(
        model_name=embedding_model_name,
        model_kwargs={'device': 'cpu'},
        encode_kwargs={'normalize_embeddings': True}
    )
    print(f"✓ Modèle d'embeddings chargé : {embedding_model_name}")

    # 4. Création de la base vectorielle (approche du notebook précédent)
    persist_directory = "./chroma_db_baseline"
    if os.path.exists(persist_directory):
        shutil.rmtree(persist_directory)

    vectorstore = Chroma.from_documents(
        documents=chunks,
        embedding=embeddings,
        persist_directory=persist_directory,
        collection_name="techstore_baseline"
    )

```

```

vectorstore.persist()
print(f"✓ Base vectorielle créée avec {len(chunks)} chunks")

# Utilisation de GPT-3.5-turbo pour un bon équilibre performance/coût
llm = ChatOpenAI(
    model_name="gpt-3.5-turbo",
    temperature=0.3,
    max_tokens=150,
    openai_api_key=os.environ["OPENAI_API_KEY"]
)
print(f"✓ Modèle de génération OpenAI configuré : gpt-3.5-turbo")

# 5. Configuration du retriever
retriever = vectorstore.as_retriever(
    search_type="similarity",
    search_kwargs={"k": top_k}
)

# 6. Template de prompt pour le service client (repris du notebook précédent)
prompt_template = """Vous êtes un assistant du service client de TechStore, une boutique d'électronique en ligne.
Votre rôle est d'aider les clients en répondant à leurs questions de manière précise et professionnelle.

Informations disponibles pour répondre :
{context}

Question du client : {question}

Instructions importantes :
- Répondez uniquement en vous basant sur les informations fournies dans le contexte
- Adoptez un ton poli, professionnel et bienveillant
- Si l'information demandée n'est pas disponible dans le contexte, dites-le clairement
- Soyez précis et donnez des détails utiles quand c'est possible
- N'inventez jamais d'informations qui ne sont pas dans le contexte

Réponse du service client :"""

PROMPT = PromptTemplate(
    template=prompt_template,
    input_variables=["context", "question"]
)

# 7. Création de la chaîne RAG complète (assemblage du notebook précédent)
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    chain_type_kwargs={"prompt": PROMPT},
    return_source_documents=True
)

print("✅ Système RAG baseline construit avec succès")
print(f"Configuration : chunk_size={chunk_size}, chunk_overlap={chunk_overlap}, top_k={top_k}")

return vectorstore, qa_chain, retriever, chunks, embeddings, llm

# Construction du système baseline
vectorstore, qa_chain, retriever, chunks, embeddings, llm = create_baseline_rag_system(documents)

# Test rapide du système pour vérifier son fonctionnement
print("\nTest rapide du système baseline :")
test_question = "Combien de temps pour retourner un produit ?"
try:
    test_result = qa_chain({"query": test_question})
    print(f"Question : {test_question}")
    print(f"Réponse : {test_result['result'][:100]}...")
    print("✅ Système RAG avec OpenAI opérationnel")
except Exception as e:
    print(f"❌ Erreur : {e}")

```

```

🔗 Construction du système RAG baseline...
(Code repris du notebook précédent pour autonomie)
✓ 11 chunks créés
/tmp/ipython-input-8-3987851723.py:33: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in
  embeddings = HuggingFaceEmbeddings(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100% 349/349 [00:00<00:00, 13.3kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 11.0kB/s]
README.md: 10.5k/? [00:00<00:00, 590kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 5.00kB/s]
config.json: 100% 612/612 [00:00<00:00, 56.5kB/s]
model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 75.8MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 26.7kB/s]
vocab.txt: 232k/? [00:00<00:00, 6.04MB/s]
tokenizer.json: 466k/? [00:00<00:00, 14.7MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 6.90kB/s]
config.json: 100% 190/190 [00:00<00:00, 13.5kB/s]
✓ Modèle d'embeddings chargé : sentence-transformers/all-MiniLM-L6-v2
/tmp/ipython-input-8-3987851723.py:51: LangChainDeprecationWarning: Since Chroma 0.4.x the manual persistence method is
  vectorstore.persist()
/tmp/ipython-input-8-3987851723.py:55: LangChainDeprecationWarning: The class `ChatOpenAI` was deprecated in LangChain 0
  llm = ChatOpenAI(
✓ Base vectorielle créée avec 11 chunks
✓ Modèle de génération OpenAI configuré : gpt-3.5-turbo
✓ Système RAG baseline construit avec succès
Configuration : chunk_size=1000, chunk_overlap=200, top_k=3

Test rapide du système baseline :
/tmp/ipython-input-9-1438334379.py:8: LangChainDeprecationWarning: The method `Chain.__call__` was deprecated in langcha
  test_result = qa_chain({"query": test_question})
Question : Combien de temps pour retourner un produit ?
Réponse : Pour retourner un produit chez TechStore, vous disposez de 30 jours calendaires à partir de la date ...
✓ Système RAG avec OpenAI opérationnel

```

✓ 3. Génération automatique du jeu de données de validation

Au lieu de définir manuellement les questions-réponses de validation, nous utilisons un LLM pour les générer automatiquement à partir du contenu des documents. Cette approche élimine les biais de sélection manuelle et assure une couverture plus représentative du domaine.

✓ 3.1. Stratégie de génération automatique avec LLM

```

def setup_question_generation_llm():
    """
    Configure un LLM OpenAI spécialisé pour la génération de questions-réponses.
    Utilise GPT-3.5-turbo avec des paramètres optimisés pour cette tâche.
    """
    print("Configuration du LLM OpenAI pour génération de questions...")

    # Vérification de la clé API
    if "OPENAI_API_KEY" not in os.environ:
        os.environ["OPENAI_API_KEY"] = getpass("🔑 Entrez votre clé API OpenAI : ")

    # Configuration spécialisée pour la génération de questions
    generation_llm = ChatOpenAI(
        model_name="gpt-3.5-turbo",
        temperature=0.7, # Plus de créativité pour la génération
        max_tokens=200, # Suffisant pour générer des questions détaillées
        openai_api_key=os.environ["OPENAI_API_KEY"]
    )

    print("LLM OpenAI de génération configuré")
    return generation_llm

def generate_validation_dataset_with_llm(documents, generation_llm, num_samples=15):
    """
    Génère automatiquement un jeu de données de validation en utilisant OpenAI.

```

Cette approche innovante utilise GPT pour créer des questions variées et représentatives du domaine, éliminant les biais de sélection manuelle.

Args:

```
documents (list): Documents source
generation_llm: LLM OpenAI configuré pour la génération
num_samples (int): Nombre de paires question-réponse à générer
```

Returns:

```
list: Liste de dictionnaires contenant question, ground_truth, contexts
"""
```

```
print("Génération automatique du jeu de données avec OpenAI...")
print("Cette approche utilise GPT pour créer des questions représentatives")
```

```
# Préparation du contexte des documents pour le LLM
```

```
documents_summary = ""
for doc in documents:
    doc_name = doc.metadata['name']
    # Prendre les 500 premiers caractères de chaque document
    content_preview = doc.page_content[:500]
    documents_summary += f"\n--- {doc_name} ---\n{content_preview}...\n"
```

```
# Génération automatique des questions avec OpenAI
```

```
generated_questions = []
reference_answers = []
```

```
# Prompt pour générer des questions de service client
```

```
generation_prompt = f"""Basé sur les documents de service client TechStore suivants :
{documents_summary}
```

Générez des questions réalistes qu'un client pourrait poser au service client de TechStore.
Variez les types de questions : délais, processus, conditions, produits, réclamations.

Générez exactement {num_samples} questions courtes et naturelles, une par ligne, sans numérotation."""

```
try:
```

```
    print("Génération des questions ...")
    questions_response = generation_llm.predict(generation_prompt)
    generated_questions = [q.strip() for q in questions_response.split('\n') if q.strip()]
```

```
    # Limiter au nombre demandé
```

```
    generated_questions = generated_questions[:num_samples]
```

```
    print(f"✓ {len(generated_questions)} questions générées")
```

```
    # Génération des réponses de référence pour chaque question
```

```
    print("Génération des réponses de référence...")
```

```
    for i, question in enumerate(generated_questions):
```

```
        answer_prompt = f"""Basé sur les documents TechStore fournis précédemment, répondez à cette question de client :
```

```
Question : {question}
```

Répondez de manière précise et professionnelle en tant qu'assistant du service client TechStore.

Basez-vous uniquement sur les informations disponibles dans les documents.

Réponse courte et directe (maximum 100 mots) :"""

```
    try:
```

```
        answer = generation_llm.predict(answer_prompt)
        reference_answers.append(answer.strip())
        print(f"✓ Réponse {i+1}/{len(generated_questions)} générée")
    except Exception as e:
        print(f"✗ Erreur génération réponse {i+1}: {e}")
        reference_answers.append("Réponse non disponible - erreur de génération")
```

```
except Exception as e:
```

```
    print(f"✗ Erreur lors de la génération : {e}")
```

```
# Création du dataset de validation
```

```
validation_data = []
```

```
for i in range(min(len(generated_questions), len(reference_answers), num_samples)):
```

```
    validation_data.append({
        'question': generated_questions[i],
        'ground_truth': reference_answers[i],
        'contexts': [] # Sera rempli lors de l'évaluation
    })
```

```
print(f"✅ {len(validation_data)} paires question-réponse générées automatiquement")
```

```
print("Les questions couvrent tous les aspects du service client TechStore")
```

```
return validation_data
```

```
# Configuration du LLM de génération
generation_llm = setup_question_generation_llm()

# Génération automatique du jeu de données de validation
validation_dataset = generate_validation_dataset_with_llm(documents, generation_llm, num_samples=15)

# Affichage d'exemples de questions générées
print("\nExemples de questions générées automatiquement :")
for i, item in enumerate(validation_dataset[:3], 1):
    print(f"\n{i}. Question : {item['question']}")
    print(f"    Réponse de référence : {item['ground_truth'][:80]}...")
```

➡ Configuration du LLM OpenAI pour génération de questions...
 LLM OpenAI de génération configuré
 Génération automatique du jeu de données avec OpenAI...
 Cette approche utilise GPT pour créer des questions représentatives
 Génération des questions ...
 /tmp/ipython-input-10-3914772738.py:65: LangChainDeprecationWarning: The method `BaseChatModel.predict` was deprecated i
 questions_response = generation_llm.predict(generation_prompt)
 ✓ 12 questions générées
 Génération des réponses de référence...
 ✓ Réponse 1/12 générée
 ✓ Réponse 2/12 générée
 ✓ Réponse 3/12 générée
 ✓ Réponse 4/12 générée
 ✓ Réponse 5/12 générée
 ✓ Réponse 6/12 générée
 ✓ Réponse 7/12 générée
 ✓ Réponse 8/12 générée
 ✓ Réponse 9/12 générée
 ✓ Réponse 10/12 générée
 ✓ Réponse 11/12 générée
 ✓ Réponse 12/12 générée
 ✓ 12 paires question-réponse générées automatiquement
 Les questions couvrent tous les aspects du service client TechStore

Exemples de questions générées automatiquement :

1. Question : – Quel est le délai de retour pour un produit reconditionné chez TechStore ?
 Réponse de référence : Le délai de retour pour un produit reconditionné chez TechStore est de 14 jours ...
2. Question : – Est-ce que je peux retourner un produit sans son emballage d'origine ?
 Réponse de référence : Selon notre politique de retour, nous n'acceptons pas les retours de produits sa...
3. Question : – Quelle est la durée de la garantie pour un smartphone Apple acheté chez TechStore ?
 Réponse de référence : La durée de la garantie pour un smartphone Apple acheté chez TechStore est de 1 ...

✓ 3.2. Préparation des données pour l'évaluation RAGAS

Cette section prépare les données générées automatiquement au format requis par RAGAS. Pour chaque question, nous récupérons les contextes pertinents et générons la réponse du système RAG à évaluer.

```
def prepare_ragas_dataset(validation_data, qa_chain, retriever):
    """
    Prépare le dataset au format requis par RAGAS en récupérant les contextes
    et générant les réponses du système pour chaque question.

    Args:
        validation_data (list): Données de validation générées automatiquement
        qa_chain: Chaîne RAG à évaluer
        retriever: Système de récupération de documents

    Returns:
        Dataset: Dataset formaté pour RAGAS
    """

    print("Préparation des données pour l'évaluation RAGAS...")
    print("Récupération des contextes et génération des réponses du système...")

    questions = []
    ground_truths = []
    answers = []
    contexts = []

    for i, item in enumerate(validation_data):
        print(f"Traitement question {i+1}/{len(validation_data)}: {item['question'][:50]}...")

        question = item['question']
        ground_truth = item['ground_truth']
```



```
# Récupération des contextes pertinents par le retriever
retrieved_docs = retriever.get_relevant_documents(question)
context_list = [doc.page_content for doc in retrieved_docs]
```

```
# Génération de la réponse par le système RAG
try:
    result = qa_chain({"query": question})
    answer = result['result']
except Exception as e:
    print(f"Erreur pour la question {i+1}: {e}")
    answer = "Erreur de génération"
```

```
# Stockage des données formatées pour RAGAS
questions.append(question)
ground_truths.append(ground_truth)
answers.append(answer)
contexts.append(context_list)
```

```
# Création du dataset RAGAS
ragas_dataset = Dataset.from_dict({
    'question': questions,
    'ground_truth': ground_truths,
    'answer': answers,
    'contexts': contexts
})
```

```
print(f"✅ Dataset RAGAS préparé avec {len(questions)} échantillons")
print("Prêt pour l'évaluation des métriques de qualité")
```

```
return ragas_dataset
```

```
# Préparation du dataset initial avec la configuration baseline
ragas_dataset = prepare_ragas_dataset(validation_dataset, qa_chain, retriever)
```

```
# Aperçu du dataset préparé
print("\nAperçu du dataset préparé pour RAGAS :")
print(f"Colonnes disponibles : {ragas_dataset.column_names}")
print(f"Nombre d'échantillons : {len(ragas_dataset)}")
print("\nStructure des données :")
print("- question: Question posée au système")
print("- ground_truth: Réponse de référence attendue")
print("- answer: Réponse générée par le système RAG")
print("- contexts: Documents récupérés par le retriever")
```

```
➡ Préparation des données pour l'évaluation RAGAS...
Récupération des contextes et génération des réponses du système...
Traitement question 1/12: - Quel est le délai de retour pour un produit reco...
/tmp/ipython-input-12-3748636682.py:30: LangChainDeprecationWarning: The method `BaseRetriever.get_relevant_documents` w
    retrieved_docs = retriever.get_relevant_documents(question)
Traitement question 2/12: - Est-ce que je peux retourner un produit sans son...
Traitement question 3/12: - Quelle est la durée de la garantie pour un smart...
Traitement question 4/12: - Quelles sont les options de livraison proposées ...
Traitement question 5/12: - Comment puis-je suivre ma commande chez TechStor...
Traitement question 6/12: - Quels sont les modèles de smartphones Samsung di...
Traitement question 7/12: - Quels opérateurs sont compatibles avec les smart...
Traitement question 8/12: - Est-ce que TechStore propose une garantie étendu...
Traitement question 9/12: - Comment puis-je bénéficier de la livraison expre...
Traitement question 10/12: - Quels accessoires sont inclus avec l'achat d'un ...
Traitement question 11/12: - Quels sont les avantages d'un compte client Tech...
Traitement question 12/12: - Quelle est la politique de remb...
✅ Dataset RAGAS préparé avec 12 échantillons
Prêt pour l'évaluation des métriques de qualité
```

```
Aperçu du dataset préparé pour RAGAS :
Colonnes disponibles : ['question', 'ground_truth', 'answer', 'contexts']
Nombre d'échantillons : 12
```

```
Structure des données :
- question: Question posée au système
- ground_truth: Réponse de référence attendue
- answer: Réponse générée par le système RAG
- contexts: Documents récupérés par le retriever
```

```
ragas_dataset.to_pandas().head()
```

	question	ground_truth	answer	contexts
0	- Quel est le délai de retour pour un produit ...	Le délai de retour pour un produit recondition...	Bonjour,\n\nLe délai de retour pour un produit...	[[# Produits non retournables\nLogiciels dém...
1	- Est-ce que je peux retourner un produit sans...	Selon notre politique de retour, nous n'accept...	Bonjour,\n\nSelon notre politique de retour et...	[# Politique de Retour et Remboursement - Tech...
2	- Quelle est la durée de la garantie pour un s...	La durée de la garantie pour un smartphone App...	La durée de la garantie pour un smartphone App...	[# Garantie et Service Après-Vente - TechStore...
3	- Quelles sont les options de livraison propos...	TechStore propose trois options de livraison :...	Bonjour,\n\nMerci pour votre question. Chez Te...	[# Livraison et Expédition - TechStore\n\n## O...
4	- Comment puis-je suivre ma commande chez Tech...	Vous pouvez suivre votre commande en vous conn...	Pour suivre votre commande chez TechStore, voi...	[# Garantie et Service Après-Vente - TechStore...

4.Évaluation RAGAS : Calcul des métriques de base

Cette section introduit l'utilisation du framework RAGAS pour évaluer objectivement la qualité du système RAG. Nous calculons trois métriques fondamentales qui mesurent différents aspects de la performance : la précision de la récupération, la fidélité de la génération, et la pertinence des réponses.

```
def evaluate_rag_with_ragas(dataset):
    """
    Évalue un système RAG avec les métriques RAGAS.

    Cette fonction utilise le framework RAGAS pour calculer des métriques
    objectives de qualité d'un système RAG.

    Args:
        dataset: Dataset préparé pour RAGAS

    Returns:
        dict: Résultats d'évaluation avec scores pour chaque métrique
    """

    print("Exécution de l'évaluation RAGAS...")
    print("Calcul des métriques : context_precision, faithfulness, answer_relevancy")
    print("Cette évaluation peut prendre quelques minutes...")

    # Configuration des métriques à évaluer
    metrics = [
        context_precision,      # Précision du contexte récupéré
        faithfulness,           # Fidélité de la réponse aux sources
        answer_relevancy        # Pertinence de la réponse à la question
    ]

    try:
        # Exécution de l'évaluation RAGAS
        print("Démarrage de l'évaluation...")
        result = evaluate(
            dataset=dataset,
            metrics=metrics,
        )

        print("✅ Évaluation RAGAS terminée avec succès")
        return result

    except Exception as e:
        print(f"❌ Erreur lors de l'évaluation RAGAS : {e}")
        print("Vérifiez que toutes les dépendances sont installées correctement")
        return None

# Évaluation baseline avec la configuration par défaut
print("ÉVALUATION BASELINE – Configuration par défaut")
print("-" * 50)
print("Cette évaluation établit la performance de référence avant optimisation")
```

```
baseline_scores = evaluate_rag_with_ragas(ragas_dataset)
```

```

ÉVALUATION BASELINE – Configuration par défaut
-----
Cette évaluation établit la performance de référence avant optimisation
Exécution de l'évaluation RAGAS...
Calcul des métriques : context_precision, faithfulness, answer_relevancy
Cette évaluation peut prendre quelques minutes...
Démarrage de l'évaluation...

Evaluating: 100%
36/36 [00:18<00:00, 1.04s/it]
✅ Évaluation RAGAS terminée avec succès

```

```
baseline_scores
```

```
➦ {'context_precision': 0.6597, 'faithfulness': 0.6827, 'answer_relevancy': 0.8451}
```

```
baseline_results_df = baseline_scores.to_pandas()
baseline_results_df.head()
```

➦

	user_input	retrieved_contexts	response	reference	context_precision	faithfulness	answer_relevancy
0	- Quel est le délai de retour pour un produit ...	[[# Produits non retournables\n- Logiciels dém...	Bonjour,\n\nLe délai de retour pour un produit...	Le délai de retour pour un produit recondition...	0.500000	0.800000	0.987056
1	- Est-ce que je peux retourner un produit sans...	[# Politique de Retour et Remboursement - Tech...	Bonjour,\n\nSelon notre politique de retour et...	Selon notre politique de retour, nous n'accept...	1.000000	0.428571	0.851891
2	- Quelle est la durée de la garantie pour un s...	[# Garantie et Service Après-Vente - TechStore...	La durée de la garantie pour un smartphone App...	La durée de la garantie pour un smartphone App...	1.000000	0.750000	0.986061
	Quelles sont	TechStore					

```
metrics = ["context_precision", "faithfulness", "answer_relevancy"]
baseline_results_df[metrics].describe()
```

➦

	context_precision	faithfulness	answer_relevancy
count	12.000000	12.000000	12.000000
mean	0.659722	0.682738	0.845055
std	0.396414	0.224091	0.275728
min	0.000000	0.333333	0.000000
25%	0.458333	0.557143	0.888198
50%	0.791667	0.732143	0.933941
75%	1.000000	0.818750	0.971202
max	1.000000	1.000000	0.987056

✓ 5. Optimisation systématique des paramètres

Cette section constitue le cœur de l'optimisation du système RAG. Nous testons systématiquement différentes combinaisons de paramètres (top-k et chunk_size) pour identifier la configuration optimale. Cette approche méthodique permet de comprendre l'impact de chaque paramètre sur les performances.

✓ 5.1. Configuration de l'espace de recherche des paramètres

```
def create_optimization_grid():
    """
    Définit l'espace de recherche pour l'optimisation des paramètres.

    Nous testons les combinaisons spécifiées dans les objectifs pédagogiques :
    - top_k ∈ {3, 5, 8} : nombre de documents récupérés
    - chunk_size ∈ {256, 512} : taille des segments de texte

    Returns:
        list: Liste des configurations à tester
    """

    print("Configuration de la grille d'optimisation...")

    # Paramètres à optimiser selon les spécifications du cours
    top_k_values = [3, 5, 8]          # Nombre de documents récupérés
    chunk_sizes = [256, 512]          # Taille des chunks en caractères

    configurations = []

    # Génération de toutes les combinaisons possibles
    for top_k in top_k_values:
        for chunk_size in chunk_sizes:
            config = {
                'top_k': top_k,
                'chunk_size': chunk_size,
                'chunk_overlap': int(chunk_size * 0.2) # 20% de chevauchement (règle empirique)
            }
```

```

        configurations.append(config)

    print(f"Grille d'optimisation créée : {len(configurations)} configurations à tester")
    print("\nConfigurations planifiées :")
    for i, config in enumerate(configurations, 1):
        print(f"{i}. top_k={config['top_k']}, chunk_size={config['chunk_size']}, "
              f"overlap={config['chunk_overlap']}")

    print(f"\nLogique d'optimisation :")
    print(f"-- Top-K faible (3) : contexte précis mais potentiellement incomplet")
    print(f"-- Top-K élevé (8) : contexte riche mais potentiellement bruité")
    print(f"-- Chunks petits (256) : précision élevée mais contexte fragmenté")
    print(f"-- Chunks grands (512) : contexte cohérent mais précision réduite")

    return configurations

# Création de la grille d'optimisation
optimization_grid = create_optimization_grid()

```

➡ Configuration de la grille d'optimisation...
Grille d'optimisation créée : 6 configurations à tester

```

Configurations planifiées :
1. top_k=3, chunk_size=256, overlap=51
2. top_k=3, chunk_size=512, overlap=102
3. top_k=5, chunk_size=256, overlap=51
4. top_k=5, chunk_size=512, overlap=102
5. top_k=8, chunk_size=256, overlap=51
6. top_k=8, chunk_size=512, overlap=102

Logique d'optimisation :
- Top-K faible (3) : contexte précis mais potentiellement incomplet
- Top-K élevé (8) : contexte riche mais potentiellement bruité
- Chunks petits (256) : précision élevée mais contexte fragmenté
- Chunks grands (512) : contexte cohérent mais précision réduite

```

✓ 5.2. Fonction de reconfiguration dynamique du système RAG

```

def reconfigure_rag_system(config, documents, embeddings, llm):
    """
    Reconfigure le système RAG avec de nouveaux paramètres.

    Cette fonction permet de tester différentes configurations sans
    reconstruire entièrement le système depuis zéro.

    Args:
        config (dict): Configuration des paramètres à appliquer
        documents (list): Documents source (constants)
        embeddings: Modèle d'embeddings (réutilisé)
        llm: Modèle de langage (réutilisé)

    Returns:
        tuple: (nouveau_vectorstore, nouveau_qa_chain, nouveau_retriever, temp_dir)
    """

    print(f"Reconfiguration avec : top_k={config['top_k']}, chunk_size={config['chunk_size']}")

    # 1. Reconfiguration du text splitter avec la nouvelle taille de chunk
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=config['chunk_size'],
        chunk_overlap=config['chunk_overlap'],
        separators=["\n\n", "\n", ". ", " ", ""], # Même stratégie que la baseline
        length_function=len
    )

    # 2. Redécoupage des documents avec les nouveaux paramètres
    new_chunks = text_splitter.split_documents(documents)
    print(f" → {len(new_chunks)} chunks créés (était {len(chunks)} en baseline)")

    # 3. Création d'une base vectorielle temporaire
    # Utilisation d'un répertoire temporaire pour éviter les conflits
    temp_dir = tempfile.mkdtemp()

    try:
        # 4. Construction de la nouvelle base vectorielle
        new_vectorstore = Chroma.from_documents(
            documents=new_chunks,
            embedding=embeddings, # Réutilisation du modèle d'embeddings
            persist_directory=temp_dir,
            collection_name=f"temp_kb_{config['top_k']}_{config['chunk_size']}"
        )
    
```

```

# 5. Configuration du nouveau retriever avec le top_k spécifié
new_retriever = new_vectorstore.as_retriever(
    search_type="similarity",
    search_kwargs={"k": config['top_k']}
)

# 6. Template de prompt (réutilisation du template validé)
prompt_template = """Vous êtes un assistant du service client de TechStore, une boutique d'électronique en ligne.
Votre rôle est d'aider les clients en répondant à leurs questions de manière précise et professionnelle.

```

Informations disponibles pour répondre :

```
{context}
```

Question du client : {question}

Instructions importantes :

- Répondez uniquement en vous basant sur les informations fournies dans le contexte
- Adoptez un ton poli, professionnel et bienveillant
- Si l'information demandée n'est pas disponible dans le contexte, dites-le clairement
- Soyez précis et donnez des détails utiles quand c'est possible
- N'inventez jamais d'informations qui ne sont pas dans le contexte

Réponse du service client : ""

```

PROMPT = PromptTemplate(
    template=prompt_template,
    input_variables=["context", "question"]
)

# 7. Création de la nouvelle chaîne RAG
new_qa_chain = RetrievalQA.from_chain_type(
    llm=llm, # Réutilisation du modèle de génération
    chain_type="stuff",
    retriever=new_retriever,
    chain_type_kwargs={"prompt": PROMPT},
    return_source_documents=True
)

print(f" 🟢 Système reconfiguré avec succès")
return new_vectorstore, new_qa_chain, new_retriever, temp_dir

```

```

except Exception as e:
    print(f" 🚫 Erreur lors de la reconfiguration : {e}")
    # Nettoyage en cas d'erreur
    shutil.rmtree(temp_dir, ignore_errors=True)
    return None, None, None, None

```

✓ 5.3. Boucle d'optimisation complète

```
def run_optimization_experiment(optimization_grid, validation_dataset, documents, embeddings, llm):
    """
```

Exécute l'expérimentation complète d'optimisation des paramètres.

Cette fonction teste systématiquement toutes les configurations et mesure leur performance avec RAGAS.

Args:

```

optimization_grid (list): Configurations à tester
validation_dataset (list): Données de validation générées
documents (list): Documents source
embeddings: Modèle d'embeddings
llm: Modèle de langage

```

Returns:

```

pd.DataFrame: Résultats de toutes les expérimentations
"""

```

```

print("DÉMARRAGE DE L'OPTIMISATION SYSTÉMATIQUE")
print("=" * 50)
print(f"Nombre de configurations à tester : {len(optimization_grid)}")
print("Chaque configuration sera évaluée avec RAGAS sur 15 questions")

```

```

results = []
temp_directories = [] # Pour nettoyage ultérieur

```

Boucle principale d'expérimentation

```

for i, config in enumerate(optimization_grid, 1):
    print(f"\n{'='*20} Expérimentation {i}/{len(optimization_grid)} {'='*20}")
    print(f"Configuration testée : {config}")

```

```

print("-" * 60)

# Reconfiguration du système avec les nouveaux paramètres
new_vectorstore, new_qa_chain, new_retriever, temp_dir = reconfigure_rag_system(
    config, documents, embeddings, llm
)

if new_qa_chain is None:
    print("❌ Échec de la reconfiguration, passage à la configuration suivante")
    continue

temp_directories.append(temp_dir)

# Préparation et évaluation pour cette configuration
try:
    print("Préparation du dataset pour évaluation...")
    ragas_dataset_config = prepare_ragas_dataset(
        validation_dataset, new_qa_chain, new_retriever
    )

    print("Lancement de l'évaluation RAGAS...")
    evaluation_results = evaluate_rag_with_ragas(ragas_dataset_config)

    if evaluation_results:
        # Conversion des résultats RAGAS en DataFrame pandas
        results_df_temp = evaluation_results.to_pandas()

        # Extraction des métriques moyennes
        context_precision_score = results_df_temp['context_precision'].mean() if 'context_precision' in results_df_temp.columns else 0
        faithfulness_score = results_df_temp['faithfulness'].mean() if 'faithfulness' in results_df_temp.columns else 0
        answer_relevancy_score = results_df_temp['answer_relevancy'].mean() if 'answer_relevancy' in results_df_temp.columns else 0

        # Stockage des résultats structurés
        result_entry = {
            'top_k': config['top_k'],
            'chunk_size': config['chunk_size'],
            'chunk_overlap': config['chunk_overlap'],
            'context_precision': context_precision_score,
            'faithfulness': faithfulness_score,
            'answer_relevancy': answer_relevancy_score
        }

        # Calcul du score global (moyenne des trois métriques)
        result_entry['overall_score'] = np.mean([
            result_entry['context_precision'],
            result_entry['faithfulness'],
            result_entry['answer_relevancy']
        ])

        results.append(result_entry)

        # Affichage des résultats de cette configuration
        print(f"✅ Résultats de la configuration {i} :")
        print(f"    Context Precision: {result_entry['context_precision']:.3f}")
        print(f"    Faithfulness: {result_entry['faithfulness']:.3f}")
        print(f"    Answer Relevancy: {result_entry['answer_relevancy']:.3f}")
        print(f"    Score Global: {result_entry['overall_score']:.3f}")
    else:
        print("❌ Échec de l'évaluation RAGAS pour cette configuration")

except Exception as e:
    print(f"❌ Erreur durant l'expérimentation {i} : {e}")

print("-" * 60)

# Nettoyage des répertoires temporaires
print("\nNettoyage des fichiers temporaires...")
for temp_dir in temp_directories:
    shutil.rmtree(temp_dir, ignore_errors=True)

# Création et tri du DataFrame des résultats
results_df = pd.DataFrame(results)

if not results_df.empty:
    # Tri par score global décroissant
    results_df = results_df.sort_values('overall_score', ascending=False)

    print(f"🎉 OPTIMISATION TERMINÉE")
    print(f"Configurations testées avec succès : {len(results_df)}")

    # Affichage du meilleur résultat
    best_config = results_df.iloc[0]

```

```

print(f"🏆 MEILLEURE CONFIGURATION IDENTIFIÉE :")
print(f"    top_k = {best_config['top_k']}")
print(f"    chunk_size = {best_config['chunk_size']}")
print(f"    Score optimal = {best_config['overall_score']:.3f}")

else:
    print("❌ Aucune configuration n'a pu être testée avec succès")

return results_df

# Lancement de l'expérimentation d'optimisation
print("Préparation du lancement de l'optimisation...")
print("Cette expérimentation peut prendre 10-15 minutes selon les performances")

optimization_results = run_optimization_experiment(
    optimization_grid, validation_dataset, documents, embeddings, llm
)

```

➡ Préparation du lancement de l'optimisation...
 Cette expérimentation peut prendre 10-15 minutes selon les performances
 DÉMARRAGE DE L'OPTIMISATION SYSTÉMATIQUE

```

=====
Nombre de configurations à tester : 6
Chaque configuration sera évaluée avec RAGAS sur 15 questions

===== Expérimentation 1/6 =====
Configuration testée : {'top_k': 3, 'chunk_size': 256, 'chunk_overlap': 51}
=====
Reconfiguration avec : top_k=3, chunk_size=256
→ 43 chunks créés (était 11 en baseline)
✅ Système reconfiguré avec succès
Préparation du dataset pour évaluation...
Préparation des données pour l'évaluation RAGAS...
Récupération des contextes et génération des réponses du système...
Traitement question 1/12: - Quel est le délai de retour pour un produit reco...
Traitement question 2/12: - Est-ce que je peux retourner un produit sans son...
Traitement question 3/12: - Quelle est la durée de la garantie pour un smart...
Traitement question 4/12: - Quelles sont les options de livraison proposées ...
Traitement question 5/12: - Comment puis-je suivre ma commande chez TechStor...
Traitement question 6/12: - Quels sont les modèles de smartphones Samsung di...
Traitement question 7/12: - Quels opérateurs sont compatibles avec les smart...
Traitement question 8/12: - Est-ce que TechStore propose une garantie étendu...
Traitement question 9/12: - Comment puis-je bénéficier de la livraison expre...
Traitement question 10/12: - Quels accessoires sont inclus avec l'achat d'un ...
Traitement question 11/12: - Quels sont les avantages d'un compte client Tech...
Traitement question 12/12: - Quelle est la politique de remb...
✅ Dataset RAGAS préparé avec 12 échantillons
Prêt pour l'évaluation des métriques de qualité
Lancement de l'évaluation RAGAS...
Exécution de l'évaluation RAGAS...
Calcul des métriques : context_precision, faithfulness, answer_relevancy
Cette évaluation peut prendre quelques minutes...
Démarrage de l'évaluation...

Evaluating: 100%                                     36/36 [00:20<00:00, 1.18it/s]

✅ Évaluation RAGAS terminée avec succès
✅ Résultats de la configuration 1 :
Context Precision: 0.472
Faithfulness: 0.583
Answer Relevancy: 0.760
Score Global: 0.605
=====

===== Expérimentation 2/6 =====
Configuration testée : {'top_k': 3, 'chunk_size': 512, 'chunk_overlap': 102}
=====
Reconfiguration avec : top_k=3, chunk_size=512
→ 20 chunks créés (était 11 en baseline)
✅ Système reconfiguré avec succès
Préparation du dataset pour évaluation...
Préparation des données pour l'évaluation RAGAS...
Récupération des contextes et génération des réponses du système...
Traitement question 1/12: - Quel est le délai de retour pour un produit reco...
Traitement question 2/12: - Est-ce que je peux retourner un produit sans son...
Traitement question 3/12: - Quelle est la durée de la garantie pour un smart...
Traitement question 4/12: - Quelles sont les options de livraison proposées ...
Traitement question 5/12: - Comment puis-je suivre ma commande chez TechStor...
Traitement question 6/12: - Quels sont les modèles de smartphones Samsung di...
Traitement question 7/12: - Quels opérateurs sont compatibles avec les smart...
Traitement question 8/12: - Est-ce que TechStore propose une garantie étendu...
Traitement question 9/12: - Comment puis-je bénéficier de la livraison expre...
Traitement question 10/12: - Quels accessoires sont inclus avec l'achat d'un ...
Traitement question 11/12: - Quels sont les avantages d'un compte client Tech...
Traitement question 12/12: - Quelle est la politique de remb...
✅ Dataset RAGAS préparé avec 12 échantillons

```

❖ 6. Analyse et visualisation des résultats

Cette section analyse en profondeur les résultats de l'optimisation pour identifier les tendances, comprendre l'impact de chaque paramètre, et fournir des recommandations concrètes. Les visualisations permettent d'explorer les données de manière intuitive.

```

import plotly.express as px
import plotly.graph_objects as go

```

```

from plotly.subplots import make_subplots

# Vérification que nous avons des résultats
if not optimization_results.empty:
    print("Création des visualisations des résultats d'optimisation...")

    metrics_data = []
    for _, row in optimization_results.iterrows():
        config_name = f"top_k={row['top_k']}, chunk={row['chunk_size']}"
        metrics_data.append({
            'Configuration': config_name,
            'Context Precision': row['context_precision'],
            'Faithfulness': row['faithfulness'],
            'Answer Relevancy': row['answer_relevancy']
        })

    metrics_df = pd.DataFrame(metrics_data)

    fig = px.bar(
        metrics_df,
        x='Configuration',
        y=['Context Precision', 'Faithfulness', 'Answer Relevancy'],
        title='Détail des Métriques par Configuration',
        barmode='group'
    )
    fig.update_layout(
        xaxis_tickangle=-45,
        width=1000,
        height=500,
        yaxis_title="Score des Métriques"
    )
    fig.show()

    # Tableau de bord récapitulatif
    print("\nRÉSUMÉ DES RÉSULTATS :")
    print(f"Nombre de configurations testées : {len(optimization_results)}")
    print(f"Meilleur score global : {optimization_results['overall_score'].max():.3f}")
    print(f"Score moyen : {optimization_results['overall_score'].mean():.3f}")
    print(f"Écart-type : {optimization_results['overall_score'].std():.3f}")

    print("\n🏆 TOP 3 DES CONFIGURATIONS :")
    for i, (_, row) in enumerate(optimization_results.head(3).iterrows(), 1):
        print(f"{i}. top_k={row['top_k']}, chunk_size={row['chunk_size']} → Score: {row['overall_score']:.3f}")

else:
    print("❌ Aucun résultat à visualiser")

    Experimentation 5/6
    Configuration testée : {'top_k': 5, 'chunk_size': 512, 'chunk_overlap': 102}
    -----
    Reconfiguration avec : top_k=5, chunk_size=512
    → 20 chunks créés (était 11 en baseline)
    ✅ Système reconfiguré avec succès
    Préparation du dataset pour évaluation...
    Préparation des données pour l'évaluation RAGAS...
    Récupération des contextes et génération des réponses du système...
    Traitement question 1/12: - Quel est le délai de retour pour un produit reco...
    Traitement question 2/12: - Est-ce que je peux retourner un produit sans son...
    Traitement question 3/12: - Quelle est la durée de la garantie pour un smart...
    Traitement question 4/12: - Quelles sont les options de livraison proposées ...
    Traitement question 5/12: - Comment puis-je suivre ma commande chez TechStor...
    Traitement question 6/12: - Quels sont les modèles de smartphones Samsung di...
    Traitement question 7/12: - Quels opérateurs sont compatibles avec les smart...
    Traitement question 8/12: - Est-ce que TechStore propose une garantie étendu...
    Traitement question 9/12: - Comment puis-je bénéficier de la livraison expre...
    Traitement question 10/12: - Quels accessoires sont inclus avec l'achat d'un ...
    Traitement question 11/12: - Quels sont les avantages d'un compte client Tech...
    Traitement question 12/12: - Quelle est la politique de remb...
    ✅ Dataset RAGAS préparé avec 12 échantillons
    Prêt pour l'évaluation des métriques de qualité
    Lancement de l'évaluation RAGAS...
    Exécution de l'évaluation RAGAS...
    Calcul des métriques : context_precision, faithfulness, answer_relevancy
    Cette évaluation peut prendre quelques minutes...
    Démarrage de l'évaluation...

    Evaluating: 100%                                     36/36 [00:21<00:00, 1.15it/s]

    ✅ Évaluation RAGAS terminée avec succès
    ✅ Résultats de la configuration 4 :
    Context Precision: 0.489
    Faithfulness: 0.682
    Answer Relevancy: 0.851
    Score Global: 0.674
    -----

    ===== Experimentation 5/6 =====
    Configuration testée : {'top_k': 8, 'chunk_size': 256, 'chunk_overlap': 51}

```


✓ Pour aller plus loin

Après avoir implémenté un chatbot RAG minimal avec LangChain, plusieurs pistes permettent d'approfondir vos compétences et d'enrichir le système :

- **Intégrer un index hybride** : combinez recherche vectorielle et recherche full-text (BM25 ou Lucene) pour améliorer la précision et la couverture des réponses.
- **Configurer un agent multi-outils** : utilisez un agent LangChain capable d'appeler plusieurs outils (base vectorielle, requêtes SQL, API externes) en fonction du contexte de la requête.
- **Gérer les coûts et la latence** : mettez en place une logique de caching, une gestion du contexte par fenêtre dynamique, ou des stratégies de fallback en cas de panne de l'API.

✓ Conclusion

Ce notebook constitue une première étape concrète vers la mise en place de systèmes de génération augmentée par récupération. En

