Rapport Technique : Développement d'un Chatbot intelligent spécialisé sur les politiques publiques sénégalaises

Abdoulaye SALL
M2 SID UADB 2024-2025

3 mars 2025

Table des matières

1 Introduction **3**
- 1.1 Objectifs du projet . . . 3
- 1.2 Contexte . . . 3
- 1.3 Flux de données . . . 3

2 Composants techniques **4**
- 2.1 Ingestion des documents . . . 4
  - 2.1.1 Extraction de texte (document_loader.py) . . . 4
  - 2.1.2 Prétraitement du texte (text_processor.py) . . . 4
  - 2.1.3 Segmentation (chunker.py) . . . 5
  - 2.1.4 Indexation (indexer.py) . . . 5
- 2.2 Système de récupération . . . 5
  - 2.2.1 Interface avec ChromaDB (vector_store.py) . . . 5
  - 2.2.2 Retriever (retriever.py) . . . 6
- 2.3 Configuration du modèle (model_config.py) . . . 6
- 2.4 Interface utilisateur . . . 7
  - 2.4.1 Application Streamlit (streamlit_app.py) . . . 7

3 Exposition API et Configuration **9**
- 3.1 Exposition API avec FASTAPI . . . 9
- 3.2 Dépendances (requirements.txt) . . . 10

4 Processus de développement **11**
- 4.1 Étapes de développement . . . 11
- 4.2 Défis techniques rencontrés . . . 11
  - 4.2.1 Extraction de texte structuré . . . 11
  - 4.2.2 Optimisation de la qualité des réponses . . . 11

5 Évaluation des performances **12**
- 5.1 Méthodologie d'évaluation . . . 12
- 5.2 Résultats . . . 12

6 Perspectives d'évolution **13**
- 6.1 Améliorations techniques . . . 13
- 6.2 Extensions fonctionnelles . . . 13

7 Conclusion **14**

A Guide d'installation **15**
A.1 Installation 15
A.2 Utilisation 15

Chapitre 1

Introduction

Ce rapport détaille la conception et l'implémentation d'un chatbot intelligent basé sur l'architecture RAG (Retrieval-Augmented Generation) et développé avec LangChain. Le système est spécialisé dans les nouvelles politiques publiques sénégalaises, notamment l'Agenda National de Transformation Sénégal 2050, la Stratégie Nationale de Développement 2025-2029, et le New Deal Technologique, sous la présidence de Bassirou Diomaye Faye et son Premier Ministre Ousmane Sonko.

1.1 Objectifs du projet

— Développer un système de question-réponse intelligent capable d'interpréter et de répondre à des questions sur les politiques publiques sénégalaises
— Créer une base de connaissances structurée à partir des documents officiels
— Fournir une interface utilisateur intuitive via Streamlit
— Assurer la précision et la fiabilité des informations fournies

1.2 Contexte

Ce projet s'inscrit dans une démarche de démocratisation de l'accès à l'information gouvernementale. Avec l'avènement de la nouvelle administration sénégalaise, de nombreux plans stratégiques et politiques ont été mis en place. Ce chatbot permet aux citoyens, chercheurs, étudiants et professionnels d'accéder facilement à ces informations et de mieux comprendre les orientations politiques actuelles du Sénégal.

1.3 Flux de données

Le système suit un flux de données en plusieurs étapes :

1. **Ingestion** : Les documents PDF sont chargés, traités et segmentés
2. **Indexation** : Les segments sont vectorisés et stockés dans ChromaDB
3. **Requête** : L'utilisateur soumet une question via l'interface Streamlit
4. **Récupération** : Le système identifie les passages les plus pertinents
5. **Génération** : Le LLM produit une réponse en s'appuyant sur les passages récupérés
6. **Présentation** : La réponse est affichée à l'utilisateur

Chapitre 2

Composants techniques

2.1 Ingestion des documents

2.1.1 Extraction de texte (document_loader.py)

L'extraction de texte des documents PDF s'effectue à l'aide de la bibliothèque PyMuPDF (fitz) :

```
import os
import PyPDF2

class DocumentLoader:
    def __init__(self, raw_dir):
        self.raw_dir = raw_dir

    def load_pdfs(self):
        """Charge le texte des PDFs dans le dossier raw."""
        documents = {}
        for file in os.listdir(self.raw_dir):
            if file.endswith(".pdf"):
                path = os.path.join(self.raw_dir, file)
                with open(path, "rb") as f:
                    reader = PyPDF2.PdfReader(f)
                    text = "\n".join([page.extract_text() for page in
                        reader.pages if page.extract_text()])
                    documents[file] = text
        return documents
```

2.1.2 Prétraitement du texte (text_processor.py)

Le texte extrait subit plusieurs étapes de prétraitement :

```
import re

class TextProcessor:
    @staticmethod
    def clean_text(text):
        """Nettoie le texte en supprimant les caract res sp ciaux et
            les espaces inutiles."""
        text = re.sub(r'\s+', ' ', text)  # Suppression des espaces
            multiples
```

```
        text = re.sub(r'[^\w\s\.,;:!?]', '', text)  # Suppression des
            caract res sp ciaux
        return text.strip()
```

2.1.3 Segmentation (chunker.py)

La segmentation divise les documents en fragments de taille appropriée tout en préservant le contexte :

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

class Chunker:
    def __init__(self, chunk_size=500, chunk_overlap=50):
        self.splitter = RecursiveCharacterTextSplitter(
            chunk_size=chunk_size, chunk_overlap=chunk_overlap
        )

    def chunk_text(self, text):
        """Segmenter le texte en chunks plus petits."""
        return self.splitter.split_text(text)
```

2.1.4 Indexation (indexer.py)

L'indexation vectorielle est gérée par ChromaDB :

```
import chromadb

class Indexer:
    def __init__(self, db_path):
        self.client = chromadb.PersistentClient(path=db_path)
        self.collection = self.client.get_or_create_collection("
            senegal_policies")

    def index_chunks(self, chunks, doc_name):
        """Ajoute les chunks    la base vectorielle."""
        for i, chunk in enumerate(chunks):
            self.collection.add(
                ids=[f"{doc_name}_{i}"],
                documents=[chunk]
            )
```

2.2 Système de récupération

2.2.1 Interface avec ChromaDB (vector_store.py)

```
import chromadb

class VectorStore:
    def __init__(self, db_path):
        self.client = chromadb.PersistentClient(path=db_path)
        self.collection = self.client.get_collection("senegal_policies")

    def search(self, query, k=5):
```

```
        """Recherche les documents les plus pertinents."""
        results = self.collection.query(query_texts=[query], n_results=k
            )
        return results["documents"]
```

2.2.2 Retriever (retriever.py)

```
from src.retrieval.vector_store import VectorStore

class Retriever:
    def __init__(self, db_path):
        self.vector_store = VectorStore(db_path)

    def retrieve_documents(self, query):
        """R cup re les documents pertinents pour la requ te."""
        return self.vector_store.search(query)
```

2.3 Configuration du modèle (model_config.py)

```
import os
import google.generativeai as genai
import os
from dotenv import load_dotenv

load_dotenv()
# Configuration de l'API Gemini
genai.configure(api_key=os.getenv("GEMINI_API_KEY"))

class Model:
    @staticmethod
    def generate_response(context, query):
        """Utilise Gemini pour g n rer une r ponse bas e sur le
           contexte et la question"""

        prompt = f"""Tu es un assistant sp cialis  dans les politiques
            publiques du S n gal ,
                    particuli rement celles mises en place par le
                       Pr sident Bassirou Diomaye Faye
                    et son Premier Ministre Ousmane Sonko.

                    Utilisez les informations suivantes pour r pondre
                          la question de l'utilisateur.
                    Si tu ne trouvez pas l'information dans les passages
                        fournis, indique-le clairement
                    sans inventer de r ponse:

        Contexte : {context}

        Question : {query}

        R ponse :"""

        model = genai.GenerativeModel("gemini-1.5-flash-8b-exp-0827")
        response = model.generate_content(prompt)
```

```
        return response.text
```

2.4 Interface utilisateur

2.4.1 Application Streamlit (streamlit_app.py)

```
import sys
import os
from PIL import Image

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
    "..", "..")))

import streamlit as st
from src.retrieval.retriever import Retriever
from src.llm.model_config import Model

# Charger l'image du S n gal
image = Image.open('src/app/vsn2050.jpg')

# Configuration de la page Streamlit
st.set_page_config(page_title="Senegal BOT", page_icon="            ",
    layout="wide")
st.title("Les Nouvelles Politiques Publiques S n galaises")
st.markdown(f"Ce chatbot sp cialis  est con u pour fournir des
    informations pr cises sur les nouvelles politiques publiques du
    S n gal .")
# Disposition de l'image et du th me  c te    c te
col1, col2 = st.columns([2, 4])

with col1:
    st.image(image, width=300)

with col2:
    # S lectionner le th me de discussion
    theme = st.selectbox(
        "Choisissez un th me de discussion",
        ["Souverainet  Economique", "Technologique Num rique", "
            Justice sociale"]
    )
    st.subheader(f"Vous avez choisi le th me : {theme}")

# Liste de mots-cl s pour chaque th me
keywords = {
    "Souverainet  Economique": [" conomie ", "vision","souverainet ",
        "ressources", "industries", "ind pendance  conomique "],
    "Technologique Num rique": ["technologie", "num rique", "digital",
        "innovation", "internet", "technologique"],
    "Justice sociale": ["justice", "  galit  ", "droits humains", "
        discrimination", "justice sociale", "in galit "]
}

# Champ de saisie pour la question
question = st.text_input(" ", key="question_input", placeholder="Entrez
    ici votre question ...")
```

```
# Fonction pour v rifier si la question est li e au th me
def is_question_relevant(question, theme):
    # Convertir la question en minuscule pour une comparaison non
       sensible    la casse
    question = question.lower()
    # V rifier si un mot-cl  du th me est pr sent dans la question
    for keyword in keywords.get(theme, []):
        if keyword.lower() in question:
            return True
    return False

# Initialiser un espace pour la r ponse
response = None

# V rifier si une question a   t   pos e et si elle est li e au
   th me choisi
if st.button("Soumettre") and question:
    if not is_question_relevant(question, theme):
        st.warning("Veuillez poser une question en lien avec le th me
           choisi.")
    else:
        # Utilisation du Retriever pour r cup rer les documents en
           fonction de la question
        retriever = Retriever(db_path="data/vector_store/chroma_db")
        documents = retriever.retrieve_documents(question)

        # Aplatir la liste de documents si n cessaire
        if isinstance(documents, list):
            # Si des sous-listes existent, les aplatir
            documents = [item for sublist in documents for item in (
               sublist if isinstance(sublist, list) else [sublist])]

        # Joindre les documents dans une seule cha ne de caract res
        context = "\n".join(documents)

        # Utilisation du mod le Gemini pour g n rer la r ponse
        response = Model.generate_response(context, question)

# Affichage de la r ponse
if response:
    st.markdown(f"       {response}")
```

Chapitre 3

Exposition API et Configuration

3.1 Exposition API avec FASTAPI

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from src.retrieval.retriever import Retriever
from src.llm.model_config import Model

app = FastAPI()

# D finition du mod le pour recevoir les requ tes
class QueryRequest(BaseModel):
    question: str

@app.post("/api")
def query_model(request: QueryRequest):
    """Endpoint pour r cup rer une r ponse bas e sur la question de
       l'utilisateur."""

    question = request.question
    retriever = Retriever(db_path="data/vector_store/chroma_db")

    # R cup rer les documents pertinents
    documents = retriever.retrieve_documents(question)

    # Aplatir la liste de documents si n cessaire
    if isinstance(documents, list):
        documents = [item for sublist in documents for item in (sublist
           if isinstance(sublist, list) else [sublist])]

    # Joindre les documents en une seule cha ne pour le contexte
    context = "\n".join(documents)

    # G n rer la r ponse avec Gemini
    response = Model.generate_response(context, question)

    if not response:
        raise HTTPException(status_code=500, detail="Erreur dans la
           g n ration de r ponse.")

    return {"question": question, "response": response}
```

3.2 Dépendances (requirements.txt)

```
langchain
chromadb
pypdf
streamlit
faiss-cpu
tiktoken
python-dotenv
PyYAML
google-generativeai
python-dotenv
fastapi
uvicorn
pydantic
```

Chapitre 4

Processus de développement

4.1 Étapes de développement

1. **Analyse des besoins** : Définition des exigences et du périmètre fonctionnel
2. **Conception** : Architecture du système et des différents modules
3. **Développement itératif** :
   — Implémentation du module d'ingestion
   — Intégration de la base vectorielle ChromaDB
   — Configuration du LLM et des RAG
   — Développement de l'interface Streamlit
   — Exposition du model sur FASTAPI
4. **Tests** : Validation des fonctionnalités et ajustements

4.2 Défis techniques rencontrés

4.2.1 Extraction de texte structuré

L'extraction d'informations structurées à partir des PDF a nécessité un prétraitement approfondi pour préserver la hiérarchie des sections et éliminer les éléments parasites (en-têtes, numéros de page).

4.2.2 Optimisation de la qualité des réponses

L'ajustement des prompts et des paramètres de récupération a été un processus itératif pour garantir des réponses à la fois précises et informatives.

Chapitre 5

Évaluation des performances

5.1 Méthodologie d'évaluation

L'évaluation du système a été réalisée selon trois critères principaux :

1. **Précision** : Exactitude factuelle des réponses
2. **Pertinence** : Adéquation des réponses aux questions posées
3. **Complétude** : Exhaustivité des informations fournies

5.2 Résultats

Le système a démontré une excellente capacité à répondre précisément aux questions factuelles sur les politiques publiques sénégalaises. Les performances sont particulièrement bonnes pour :

— Les questions sur la structure et les objectifs des plans stratégiques
— Les informations chronologiques sur les initiatives
— Les données chiffrées (budgets, objectifs quantifiés)

Des améliorations restent possibles pour :

— Les questions complexes nécessitant une analyse comparative entre politiques
— Les requêtes très spécifiques sur des aspects techniques détaillés

Chapitre 6

Perspectives d'évolution

6.1 Améliorations techniques

— **Intégration de modèles locaux** : Permettre l'utilisation de LLM open-source exécutés localement
— **Système d'évaluation des réponses** : Mécanisme de feedback utilisateur pour améliorer la qualité
— **Optimisation de la segmentation** : Utilisation d'algorithmes plus sophistiqués préservant mieux le contexte

6.2 Extensions fonctionnelles

— **Mise à jour automatique** : Système de veille pour intégrer automatiquement les nouvelles publications
— **Visualisation de données** : Intégration de graphiques pour représenter les indicateurs de performance des politiques
— **Support multilingue** : Extension aux langues nationales du Sénégal

Chapitre 7

Conclusion

Ce projet démontre l'efficacité de l'architecture RAG pour développer un chatbot spécialisé dans un domaine précis, en l'occurrence les politiques publiques sénégalaises. L'utilisation de LangChain a permis une implémentation modulaire et extensible, tandis que ChromaDB a fourni une base vectorielle performante pour la récupération d'informations.

Le résultat est un outil pratique qui facilite l'accès à l'information gouvernementale et contribue à une meilleure compréhension des orientations stratégiques du Sénégal. Les perspectives d'évolution ouvrent la voie à des améliorations continues tant sur le plan technique que fonctionnel.

Annexe A

Guide d'installation

A.1 Installation

1. Cloner le dépôt :

```
git clone https://github.com/tokosel/Visionsenegal2050.git
cd Visionsenegal2050
```

2. Créer un environnement :

```
# Avec python
# Cr ation
python -m venv env
# Activation
.\env\Scripts\activate
```

```
#Avec Conda
# Cr ation
conda create --name env python==3.10
# Activation
conda activate env
```

3. Installer les dépendances :

```
pip install -r requirements.txt
```

4. Configurer les variables d'environnement :
   — Créer un fichier `.env`
   — Ajouter la clé API Google GEMINI

A.2 Utilisation

1. Lancer le pipeline de l'ingestion des documents :

```
python pipeline.py
```

2. Démarrer l'application :

```
streamlit run src/app/streamlit_app.py
```

3. Accéder à l'interface via le navigateur :

```
http://localhost:8501
```

4. Utilisation de l'API :

```
# Lancement de l'API
uvicorn src.app.api:app --host 0.0.0.0 --port 8000 --reload
#Test de l'API
http://localhost:8000/docs
```