



ÉCOLE CENTRALE CASABLANCA

RAPPORT

---

# Système RAG - Assistant de Codage PyTorch

---

***Élèves :***

Amine ARBAOUI  
Hamza BELAAZRI  
Oussama LAFDIL  
Haitam M'BIRIK

***Encadrant :***  
IMAD LAKIM

17 mars 2025

Table des matières

1 Introduction 2

1.1 Contexte et Problématique . . . . . 2

1.2 Objectif . . . . . 2

2 Architecture du Système RAG 2

2.1 Module d'Indexation (DocumentIndexer) . . . . . 2

2.2 Moteur de Recherche Sémantique (DocumentSearcher) . . . . . 3

2.3 Générateur de Réponses (QASystem) . . . . . 3

2.4 Module d'Évaluation (RAGEvaluator) . . . . . 3

2.5 Chatbot Interactif (ChatSystem) (Optionnel) . . . . . 3

3 Choix Techniques et Justifications 3

3.1 Framework LangChain . . . . . 3

3.2 Vector Store . . . . . 4

3.3 Modèle d'Embeddings . . . . . 4

3.4 Large Language Model (LLM) . . . . . 5

3.5 Structure de Configuration (fichier YAML) . . . . . 6

3.6 Gestion des Dépendances (requirements.txt) . . . . . 6

3.7 CLI . . . . . 6

4 Implémentation des Composants 6

4.1 Indexation des Documents . . . . . 7

4.2 Recherche Sémantique . . . . . 7

4.3 Génération de Réponses . . . . . 7

4.4 Évaluation de la Pertinence . . . . . 8

5 Résultats et Évaluation 8

5.1 Indexation . . . . . 8

5.2 Recherche . . . . . 9

5.3 Question-Réponse . . . . . 10

5.4 Évaluation . . . . . 12

5.4.1 Résultats Quantitatifs . . . . . 12

5.4.2 Analyse Qualitative . . . . . 13

5.4.3 Discussion et Améliorations . . . . . 13

6 Conclusion 13

# 1 Introduction

## 1.1 Contexte et Problématique

L'évolution rapide des modèles de traitement du langage naturel (NLP) a permis l'émergence de systèmes avancés d'accès à l'information, parmi lesquels les systèmes de génération augmentée par récupération (RAG - Retrieval-Augmented Generation). Ces systèmes combinent des techniques de recherche d'information avec des modèles génératifs afin de produire des réponses précises et contextuelles en s'appuyant sur une base de connaissances externe.

Dans le domaine de la programmation, et en particulier dans l'utilisation de frameworks complexes comme PyTorch, l'accès rapide et pertinent à la documentation technique est un enjeu majeur. Les développeurs passent souvent un temps considérable à rechercher des informations, à comprendre des concepts avancés et à intégrer correctement les différentes fonctionnalités dans leurs projets. C'est dans ce contexte que notre projet vise à développer un assistant de codage basé sur un système RAG, spécialisé pour PyTorch.

## 1.2 Objectif

L'objectif principal de ce projet est de concevoir une solution capable d'indexer la documentation officielle de PyTorch, d'effectuer une recherche efficace sur cette documentation et de fournir des réponses précises aux questions des utilisateurs, incluant des explications techniques et des exemples de code.

Le projet intègre plusieurs composants clés :

- Un indexeur de documents permettant de structurer et stocker efficacement la documentation.
- Un moteur de recherche qui identifie les passages les plus pertinents pour une requête donnée.
- Un système de génération de réponses basé sur un modèle NLP avancé.
- Un module d'évaluation pour garantir la pertinence et la qualité des réponses fournies.
- Une extension possible sous forme de chatbot permettant une interaction plus fluide avec les utilisateurs.

# 2 Architecture du Système RAG

Le système RAG mis en place repose sur une architecture modulaire qui intègre plusieurs composants essentiels travaillant de concert pour offrir une expérience utilisateur optimale. L'architecture générale comprend :

## 2.1 Module d'Indexation (DocumentIndexer)

- Charge la documentation PyTorch.
- Segmente les données en fragments pertinents.
- Génère et stocke des représentations vectorielles (embeddings) dans une base de données optimisée.

## 2.2 Moteur de Recherche Sémantique (DocumentSearcher)

- Effectue une recherche par similarité dans un espace vectoriel.
- Identifie les passages les plus pertinents en fonction des requêtes des utilisateurs.
- Permet d’aller au-delà des simples correspondances de mots-clés pour une recherche plus contextuelle.

## 2.3 Générateur de Réponses (QASystem)

- Utilise un modèle de langage avancé pour formuler des réponses précises.
- S’appuie sur les fragments de documentation récupérés.
- Génère du texte naturel incluant des extraits de code PyTorch si nécessaire.
- Optimise la structuration des prompts pour améliorer la qualité des réponses générées.

## 2.4 Module d’Évaluation (RAGEvaluator)

- Analyse la pertinence et la qualité des réponses générées.
- Évalue plusieurs critères : exactitude, clarté, correction syntaxique et logique du code, et absence d’hallucinations.
- Permet d’optimiser le système en identifiant les points d’amélioration.

## 2.5 Chatbot Interactif (ChatSystem) (Optionnel)

- Conserve l’historique des échanges pour adapter ses réponses.
- Offre une interaction utilisateur plus fluide et naturelle.
- Gère intelligemment la mémoire conversationnelle pour répondre aux questions de suivi.

Cette architecture modulaire et flexible permet de construire un assistant de codage puissant et adaptable, capable d’aider efficacement les développeurs travaillant avec PyTorch. L’intégration harmonieuse des différentes composantes assure une interaction fluide et optimisée, tout en garantissant une grande précision des réponses fournies.

# 3 Choix Techniques et Justifications

Cette section justifie les choix technologiques effectués pour la construction de notre système RAG d’assistance au codage PyTorch. Chaque choix est motivé par des considérations de performance, de facilité d’utilisation, et d’adéquation au cas d’utilisation.

## 3.1 Framework LangChain

Nous avons choisi d’utiliser le framework **LangChain** pour plusieurs raisons :

- **Simplification de l’implémentation** : LangChain fournit des abstractions de haut niveau et des composants pré-construits (chaînes, prompts, modèles, intégrations avec des vector stores) qui accélèrent considérablement le développement d’applications basées sur des LLMs.

- **Accès à des fonctionnalités avancées** : LangChain offre des outils pour la gestion de l'historique des conversations, la création de prompts complexes, et la manipulation de documents, ce qui est essentiel pour un système RAG performant.
- **Large communauté et documentation** : LangChain bénéficie d'une communauté active et d'une documentation complète, ce qui facilite la résolution de problèmes et l'apprentissage.
- **Intégration avec d'autres outils** : LangChain s'intègre facilement avec d'autres bibliothèques et services, tels que Hugging Face (pour les modèles et les embeddings) et différents vector stores.

Dans notre projet, LangChain est utilisé intensivement pour :

- Définir les chaînes de question-réponse (`create_retrieval_chain`, `create_stuff_documents_chain`).
- Créer et gérer les prompts (`ChatPromptTemplate`, `PromptTemplate`).
- Intégrer les modèles de langage (`ChatGoogleGenerativeAI`, `HuggingFaceEndpoint`).
- Gérer l'historique des conversations (dans le `ChatSystem`).

## 3.2 Vector Store

Pour le stockage des embeddings de la documentation PyTorch, nous avons considéré deux options principales : **FAISS** et **Chroma**.

- **FAISS (Facebook AI Similarity Search)** est une bibliothèque optimisée pour la recherche rapide de similarité dans des espaces de grande dimension. Elle est particulièrement efficace pour les grands ensembles de données. Ses avantages incluent :
  - Haute performance en recherche de similarité.
  - Optimisation pour la vitesse et l'utilisation de la mémoire.
  - Support de différents types d'index (plats, IVF, HNSW).
- **Chroma** est une base de données vectorielle open-source conçue pour être facile à utiliser et à déployer. Ses avantages incluent :
  - Simplicité d'utilisation et d'installation.
  - Intégration facile avec LangChain.
  - Fonctionnalités de persistance et de gestion des données intégrées.

**Notre choix** : Nous avons implémenté le code supportant les deux, configurable via le fichier `config.yaml`, le choix final dépendra de tests plus poussés de performance et de simplicité d'utilisation, mais pour le moment, nous avons laissé la possibilité de supporter les deux options.

**Justification de la persistance** : Le vector store est sauvegardé sur disque (dans le répertoire spécifié par `persist_directory` dans la configuration). Cela permet de :

- Éviter de recalculer les embeddings et de reconstruire l'index à chaque exécution du programme.
- Réduire le temps de démarrage de l'application.
- Faciliter le déploiement du système.

## 3.3 Modèle d'Embeddings

Le choix du modèle d'embeddings est crucial pour la performance du système RAG, car il détermine la qualité de la représentation vectorielle des documents et des requêtes.

Nous avons utilisé **HuggingFaceEmbeddings** de LangChain, qui offre une interface simple pour utiliser les modèles de Hugging Face.

**Modèle choisi** : Nous avons configuré l'utilisation du modèle `all-mpnet-base-v2` via le fichier `config.yaml`, mais le code permet de facilement changer de modèle.. Ce modèle a été choisi pour les raisons suivantes :

- **Performance élevée** : Il obtient de bons résultats sur des tâches d'évaluation de la similarité sémantique.
- **Taille raisonnable** : Il offre un bon compromis entre performance et taille, ce qui le rend utilisable sur des machines avec des ressources limitées.
- **Polyvalence** : Il est capable de bien représenter le sens des phrases et des paragraphes, y compris dans le contexte du code et du langage technique.

### 3.4 Large Language Model (LLM)

Le LLM est le cœur du système de question-réponse. Il est responsable de la génération des réponses à partir du contexte récupéré et de la question posée. Nous avons configuré notre système pour supporter deux types de LLMs, offrant ainsi une flexibilité maximale : un modèle open-source de Hugging Face et un modèle commercial de Google. Ce choix permet de comparer les performances et les coûts, et de s'adapter à différents besoins et contraintes.

**Modèles Configurés :**

- **Hugging Face** : `mistral-7b-v0.3` (via `HuggingFaceEndpoint`) :
  - **Avantages** : Modèle open-source, accessible via l'API Hugging Face. Mistral-7B est connu pour ses bonnes performances et sa taille relativement compacte, ce qui le rend utilisable avec des ressources modérées.
  - **Considérations** : Les performances peuvent être inférieures à celles des modèles plus grands ou commerciaux, en particulier pour les tâches complexes de génération de code.
- **Google** : `gemini-2.0-pro-exp-02-05` (via `ChatGoogleGenerativeAI`) :
  - **Avantages** : Modèle commercial puissant, offrant potentiellement une meilleure qualité de réponse et une meilleure capacité à générer du code complexe. Accès via une API, ce qui simplifie l'intégration.
  - **Considérations** : Coûts d'utilisation potentiels associés à l'API Google une fois en production.

**Justification de la double configuration :**

- **Flexibilité** : Permet à l'utilisateur de choisir le modèle le plus adapté à ses besoins (performance, coût, disponibilité).
- **Comparaison** : Facilite la comparaison directe des performances des deux modèles sur notre cas d'utilisation spécifique (assistance au codage PyTorch).
- **Robustesse** : Offre une alternative en cas d'indisponibilité de l'un des modèles ou de l'une des API.

**Choix du LLM via la configuration** : Le modèle de langage à utiliser est spécifié dans le fichier de configuration `config.yaml`, sous la clé `llm.provider` (pour le fournisseur : `"huggingface"` ou `"google"`) et `llm.model` (pour le nom du modèle spécifique). Cela permet de changer de modèle sans modifier le code.

**Gestion des imports PyTorch** : Quel que soit le LLM choisi, le prompt est conçu pour encourager la génération de code PyTorch correct. Les LLMs, en particulier ceux

entraînés sur de grandes quantités de code, sont généralement capables de gérer les imports de manière appropriée.

**Alternatives et évolutions futures :**

- **Modèles spécialisés dans le code** (e.g. Qwen2.5-Coder-7B-Instruct) : Ce modèle est spécifiquement conçu pour la génération de code et pourrait être plus performants pour notre cas d'utilisation. Cependant, leur accès peut être plus limité ou nécessiter une configuration plus complexe. Nous prévoyons d'explorer ces options dans le futur.

### 3.5 Structure de Configuration (fichier YAML)

Nous avons utilisé un fichier de configuration YAML (`config.yaml`) pour stocker tous les paramètres du système RAG. Ce choix est motivé par :

- **Flexibilité** : Les paramètres (modèles, chemins, paramètres de découpage, etc.) peuvent être modifiés facilement sans avoir à modifier le code source.
- **Maintenabilité** : Le code est plus propre et plus lisible, car les paramètres sont séparés de la logique.
- **Reproductibilité** : Le fichier de configuration permet de reproduire facilement les mêmes résultats en utilisant les mêmes paramètres.
- **Séparation des préoccupations** : La configuration est distincte du code, ce qui facilite la gestion et l'évolution du projet.

### 3.6 Gestion des Dépendances (`requirements.txt`)

Le fichier `requirements.txt` liste toutes les dépendances Python du projet, avec leurs versions spécifiques. Cela garantit :

- **Reproductibilité** : N'importe qui peut installer exactement les mêmes versions des bibliothèques et exécuter le code sans problème de compatibilité.
- **Déploiement simplifié** : Le fichier `requirements.txt` facilite la création d'environnements virtuels et le déploiement du système sur d'autres machines.

### 3.7 CLI

Nous avons mis en place une interface en ligne de commande (CLI) grâce au script `cli.py` et à la bibliothèque `argparse`.

- **Facilité d'utilisation** : L'interface en ligne de commande rend l'interaction avec le programme plus simple pour l'utilisateur.
- **Automatisation** : Le CLI permet une utilisation scriptée et donc une automatisation de l'interaction.

## 4 Implémentation des Composants

Dans cette section, nous détaillons l'implémentation des différents composants du système RAG, en mettant l'accent sur leur intégration et leur fonctionnement.

## 4.1 Indexation des Documents

L'indexation des documents PyTorch est réalisée en plusieurs étapes :

- Chargement et prétraitement des documents en supprimant le bruit et en segmentant en passages pertinents.
- Génération des embeddings via un modèle de transformation tel que `sentence-transformers`.
- Stockage des vecteurs dans un vector store (`FAISS` ou `Chroma`).

L'implémentation de l'indexeur est la suivante :

```
from langchain.document_loaders import PyPDFLoader

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores import FAISS


# Chargement des documents

loader = PyPDFLoader("pytorch_docs.pdf")

documents = loader.load()


# Génération des embeddings

embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

vector_store = FAISS.from_documents(documents, embedding_model)
```

Listing 1 – Implémentation de l'indexation des documents

## 4.2 Recherche Sémantique

Une fois les documents indexés, nous devons être capables de récupérer les passages les plus pertinents.

```
query = "Comment utiliser torch.nn.Linear ?"

retrieved_docs = vector_store.similarity_search(query, k=3)

for doc in retrieved_docs:

    print(doc.page_content)
```

Listing 2 – Recherche sémantique des documents

## 4.3 Génération de Réponses

Le module de génération exploite un LLM pour fournir des réponses contextuelles.



```
from langchain.llms import OpenAI

from langchain.chains import RetrievalQA

llm = OpenAI(model_name="gpt-4")

qa_chain = RetrievalQA(llm=llm, retriever=vector_store.as_retriever())

query = "Comment fonctionne torch.nn.Linear ?"

response = qa_chain.run(query)

print(response)
```

Listing 3 – Génération de réponses avec LangChain

## 4.4 Évaluation de la Pertinence

Enfin, nous intégrons un module permettant d'évaluer la pertinence des réponses générées.

```
from langchain.evaluation import load_evaluator

evaluator = load_evaluator("bertscore")

score = evaluator.evaluate(prediction=response, reference="D finition  
de torch.nn.Linear")

print("Score de pertinence:", score)
```

Listing 4 – Évaluation des réponses

L'ensemble de ces composants interagissent pour offrir un assistant de codage performant et précis, optimisant l'accès à la documentation PyTorch.

## 5 Résultats et Évaluation

Cette section présente les résultats obtenus pour chaque composant du système RAG, ainsi qu'une évaluation quantitative et qualitative de ses performances.

### 5.1 Indexation

- Nombre de documents PyTorch indexés : 12
- Taille du vector store (sur disque) : 130 Ko

## 5.2 Recherche

Nous présentons ici quelques exemples de requêtes liées à PyTorch, ainsi que les résultats renvoyés par le `DocumentSearcher`. Pour chaque requête, nous affichons les 3 premiers résultats (chunks de documentation), avec leur score de similarité (plus le score est faible, plus le document est similaire à la requête).

— **Requête 1** : How to install Pytorch ?

```
Result 1: [Score: 0.5173]
Source: data\pytorch_org (13).pdf
Content: GET STARTED
Select preferences and run the command to install PyTorch locally, or
get started quickly with one
of the supported cloud platforms.
Start Locally
PyTorch 2.x
Start via Cloud Partners
Previous PyTorch Versions
ExecuTorch
START LOCALLY
Installing on Linux
PyTorch can be installed and used on various Linux distributions. Depending on your system and compute
requirements, your experience ...
-----
Result 2: [Score: 0.5173]
Source: data\pytorch_org (14).pdf
Content: GET STARTED
Select preferences and run the command to install PyTorch locally, or
get started quickly with one
of the supported cloud platforms.
Start Locally
PyTorch 2.x
Start via Cloud Partners
Previous PyTorch Versions
ExecuTorch
START LOCALLY
Installing on Linux
PyTorch can be installed and used on various Linux distributions. Depending on your system and compute
```

FIGURE 1 – Historique de l'optimisation Optuna.

— **Requête 2** : How to activate CUDA ?

```

Search Results for: 'How to activate CUDA?'

Found 2 results

Result 1: [Score: 0.9540]
Source: data\pytorch_org (5).pdf
Content: Table of Contents
torch.cuda
This package adds support for CUDA tensor types.
It implements the same function as CPU tensors, but they utilize
GPUs for computation.
It is lazily initialized, so you can always import it, and use
is_available() to determine if your system supports CUDA.
CUDA semantics has more details about working with CUDA.
StreamContext
Context-manager that selects a given stre...

-----
Result 2: [Score: 1.1860]
Source: data\pytorch_org (5).pdf
Content: synchronize
Wait for all kernels in all streams on a CUDA device to complete.
utilization
Return the percent of time over the past sample period during
which one or more kernels was executing on the GPU as given by
nvidia-smi.
temperature
Return the average temperature of the GPU sensor in Degrees C
(Centigrades).
power_draw
Return the average power draw of the GPU sensor in mW
(MilliWatts)
clock...

```

FIGURE 2 – Historique de l'optimisation Optuna.

### 5.3 Question-Réponse

Voici des exemples de questions posées au **QASystem**, accompagnées des réponses générées. Nous incluons des questions conceptuelles, des demandes de code et des questions sur des fonctions spécifiques de PyTorch. Les questions et réponses sont en anglais, comme dans les données fournies.

— **Question 1 :** How to transform a tensor to numpy array?

**Réponse générée :**

Tensors on the CPU and NumPy arrays can share their underlying memory locations. Changing one will change the other. To transform a tensor to a NumPy array, use the `‘.numpy()’` method on the tensor.

```

t = torch.ones(5)
print(f"t: {t}")
n = t.numpy()
print(f"n: {n}")

t.add_(1) # In-place addition to the tensor
print(f"t: {t}")
print(f"n: {n}") # The numpy array reflects the change

Sources:

pytorch_org (2).pdf
pytorch_org (11).pdf

```

## Listing 5 – Code pour transformer un tensor en numpy array

— **Question 2 :** How to create a neural network with three hidden layers ?

**Réponse générée :**

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class ThreeLayerNet(nn.Module):
def init(self, input_size, hidden_size1, hidden_size2,
        hidden_size3, output_size):
super(ThreeLayerNet, self).init()
self.fc1 = nn.Linear(input_size, hidden_size1)
self.fc2 = nn.Linear(hidden_size1, hidden_size2)
self.fc3 = nn.Linear(hidden_size2, hidden_size3)
self.fc4 = nn.Linear(hidden_size3, output_size)

def forward(self, x):
    x = F.relu(self.fc1(x)) # Apply ReLU activation after each
        hidden layer
    x = F.relu(self.fc2(x))
    x = F.relu(self.fc3(x))
    x = self.fc4(x) # Output layer (no activation for
        regression, or softmax for classification)
    return x

Example usage:
input_size = 784 # Example: 28x28 image input
hidden_size1 = 256
hidden_size2 = 128
hidden_size3 = 64
output_size = 10 # Example: 10 classes

Create an instance of the network
net = ThreeLayerNet(input_size, hidden_size1, hidden_size2,
        hidden_size3, output_size)

Example input (batch size of 1)
input_tensor = torch.randn(1, input_size)

Forward pass
output = net(input_tensor)

print(output)
```

## Listing 6 – Code pour un réseau de neurones à trois couches cachées

**Explanation and Key Improvements :**

- **Clearer Class Definition :** The code defines a `ThreeLayerNet` class that inherits from `nn.Module`, making it a proper PyTorch module.
- **Flexible Hidden Layer Sizes :** The constructor (`__init__`) takes arguments for the input size, the sizes of all three hidden layers, and the output size, allowing you to customize the network architecture.

- **ReLU Activation** : ReLU (Rectified Linear Unit) activation function is applied after each hidden layer. ReLU is a common choice and often improves training performance.
  - **Output Layer Flexibility** : The output layer (`self.fc4`) doesn't have a fixed activation function. This is important :
    - For **regression tasks**, you typically don't want an activation function on the output layer.
    - For **classification tasks**, you would add a softmax function (or other appropriate activation) after the output layer (often handled during loss calculation).
  - **Example Usage** : The code demonstrates how to create an instance of the network, create a sample input tensor, and perform a forward pass to get the output.
  - **Comments** : Added comments to explain the purpose of each part of the code.
- Sources :

`pytorch_org (3).pdf`

`pytorch_org (1).pdf`

## 5.4 Évaluation

Nous avons évalué le système à l'aide d'un ensemble de 5 questions de test (listées dans le fichier d'évaluation), couvrant des fonctionnalités de base de PyTorch. Le **RAGEvaluator** a évalué chaque réponse en fonction de : la pertinence (1-10), l'exactitude (1-10), la complétude (1-10), l'hallucination (Oui/Non), la correction du code (syntaxe - Oui/Non), la correction du code (logique - Oui/Non), la pertinence PyTorch (Oui/Non) et le style du code (1-10).

### 5.4.1 Résultats Quantitatifs

Le tableau 1 résume les scores moyens.

Critère	Score Moyen
Pertinence	10
Exactitude	10
Complétude	10
Hallucination (Pourcentage de "Non")	100%
Correction du code (Syntaxe) (Pourcentage de "Oui")	100%
Correction du code (Logique) (Pourcentage de "Oui")	100%
Pertinence PyTorch (Pourcentage de "Oui")	100%
Style et bonnes pratiques	10

TABLE 1 – Résultats quantitatifs de l'évaluation.

### 5.4.2 Analyse Qualitative

Les évaluations indiquent une excellente performance. Le système a fourni de manière constante des réponses pertinentes, exactes et complètes, sans aucune hallucination. Le code généré était correct du point de vue de la syntaxe et de la logique, et respectait les conventions de PyTorch. Toutes les réponses ont obtenu des scores parfaits pour tous les critères. Cela suggère que le système récupère et utilise efficacement les informations de la documentation PyTorch indexée. Étant donné que toutes les évaluations étaient parfaites, la présentation d'exemples d'évaluation individuels est redondante. Le système a systématiquement satisfait aux normes les plus élevées pour ces requêtes de base.

### 5.4.3 Discussion et Améliorations

Le système fonctionne exceptionnellement bien sur ces questions fondamentales. Bien qu'il s'agisse d'un résultat positif, il représente une base de référence. Les travaux futurs devraient se concentrer sur :

- **Questions plus difficiles** : Évaluer avec des questions plus complexes impliquant un raisonnement en plusieurs étapes, des fonctionnalités avancées de PyTorch et des tâches de génération de code plus complexes.
- **Ensemble de test plus grand** : Élargir l'ensemble d'évaluation pour couvrir un plus large éventail de sujets et de types de questions.
- **Cas limites** : Tester la robustesse du système avec des questions ambiguës ou mal formulées, et des questions qui se situent aux limites des connaissances documentées.

## 6 Conclusion

Ce projet a permis de mettre en place avec succès un système RAG (*Retrieval-Augmented Generation*) conçu comme un assistant de codage spécifiquement pour PyTorch. En intégrant un mécanisme d'indexation robuste, un moteur de recherche sémantique et un modèle de langage étendu (LLM) puissant, nous avons créé un outil capable de récupérer efficacement des informations pertinentes à partir de la documentation PyTorch et de générer des réponses précises et contextuelles aux requêtes des utilisateurs. Le système démontre de solides performances dans la gestion des questions de base à intermédiaires liées aux concepts PyTorch, aux extraits de code et à l'utilisation de l'API. L'architecture modulaire, utilisant LangChain, permet la flexibilité et les extensions futures. L'utilisation d'un fichier de configuration (YAML) et d'un fichier `requirements.txt` clair améliore la maintenabilité, la reproductibilité et la facilité de déploiement. Le CLI simplifie l'interaction pour l'utilisateur final.

Bien que les résultats de l'évaluation soient très encourageants, en particulier les scores parfaits sur notre ensemble de tests initial, il est crucial de reconnaître qu'il s'agit d'une base solide, et non d'un point final. L'évaluation actuelle se concentre principalement sur les concepts fondamentaux de PyTorch. Les travaux futurs devraient donner la priorité à l'extension de l'évaluation pour englober des scénarios plus complexes, des cas limites et des fonctionnalités avancées de PyTorch.

## Annexe : Dépôt GitHub

Le code source, la documentation et les fichiers associés à ce projet sont disponibles sur GitHub dans le dépôt suivant :

[https://github.com/OussamaLafdil/NLP\\_RagProject](https://github.com/OussamaLafdil/NLP_RagProject)