



Université : Abdelmalek Essaâdi
Ecole Nationale des Sciences Appliquées

Rapport du Projet

Open Domain Question Answering

Module: TALN



Réalisé par :

TIDAADAR Fatima Ezzahraa

SEBBAR Asmae

SALIM Aya

YOUNES Omaima

BOUHAFA Taha

BOULAALAM Yassine

BOUSKINE Othmane

Encadré par :
Pr.Belcaid Anass

Table des matières

1 Contexte général et objectifs	7
1.1 Contexte et motivation : De la Logique Symbolique aux Réseaux de Neurones Profonds	7
1.2 Importance des systèmes de Question-Answering en intelligence artificielle	8
1.3 Problématique de l'Open-Domain Question Answering	9
1.4 Objectifs du projet	10
2 Etat de l'art	11
2.1 Définition du Question Answering	11
2.1.1 Closed-Domain vs Open-Domain Question Answering	11
2.2 Architectures des Systèmes ODQA : Une Évolution en Trois Étapes	12
2.2.1 L'Approche Traditionnelle : IR + NLP (Retriever-Reader)	12
2.2.2 L'Approche Basée sur le Deep Learning : Récupération Dense (DPR)	14
2.2.3 L'Approche Retrieval-Augmented Generation (RAG)	15
2.3 Modèles et techniques existants	17
2.3.1 Modèles basés sur des transformers	18
2.3.2 Modèles spécialisés pour le Question Answering	21
2.4 Applications réelles du Question Answering (QA)	22
3 Dataset utilisée	23
3.1 Présentation du Dataset	23
3.2 Caractéristiques du Dataset	23
3.3 Prétraitement et Filtrage	24
3.3.1 Filtrage des Exemples Utilisables	24
3.3.2 Construction du Corpus de Passages	24
3.3.3 Alignement des Positions	25
3.3.4 Division du Dataset	25
4 Architecture du Système Proposé	26
4.1 Vue d'Ensemble de l'Architecture	26
4.2 Module de traitement des questions	28
4.2.1 Nettoyage et normalisation	28
4.2.2 Tokenisation	29
4.2.3 Encodage sémantique	29

4.3	Module de Récupération d'Information (Retrieval)	29
4.3.1	Architecture Dense Passage Retrieval (DPR)	29
4.3.2	Indexation des Documents	30
4.3.3	Recherche Sémantique	31
4.3.4	Sélection des Passages Pertinents	31
4.4	Module de Lecture et Compréhension (Reader)	32
4.4.1	Architecture du Modèle	32
4.4.2	Fine-tuning Efficace avec LoRA	33
4.4.3	Optimisation de l'Entraînement avec LoRA	34
4.4.4	Extraction de la Réponse	36
4.4.5	Gestion des Réponses Longues et Courtes	37
5	Implémentation	39
5.1	Environnement de Développement et Mise en Œuvre	39
5.1.1	Environnement de Développement	39
5.2	Déroulement du Pipeline	41
5.2.1	Phase d'Indexation	41
5.2.2	Phase d'Entraînement du Reader	42
5.2.3	Phase d'Inférence	43
6	Évaluation et résultats	44
6.1	Évaluation des Performances	44
6.1.1	Métriques d'Évaluation	44
6.1.2	Scénarios de Tests et Analyse par Type de Question	49
6.1.3	Analyse des Résultats	51
6.2	Comparaison des Performances : Modèles Baselines vs. Notre Modèle Fine-tuné .	54
6.2.1	Modèles Baselines Évalués	54
6.2.2	Résultats de Performance	54
6.2.3	Analyse Comparative	55
7	Intérface Web	56
7.1	Architecture du Frontend	56
7.1.1	Stack Technologique	56
7.1.2	Optimisations CSS Avancées	58
7.2	Communication Frontend-Backend	58
	 AXIOS	
7.2.1	Client HTTP : Axios	58
7.2.2	Flux de Données Temps Réel	59
7.3	Gestion de l'Authentification et des Données	60
	 Supabase : Backend-as-a-Service	
7.3.1	Supabase : Backend-as-a-Service	60
7.3.2	Système d'Authentification	61
7.4	Architecture et flux d'interaction de l'interface web pour un système de Question-Réponse	61

7.5 Démonstration de l'Application Web	62
7.5.1 Page d'Accueil	62
7.5.2 Personnalisation de l'expérience	63
7.5.3 Interface de Chat et Interaction	63
Conclusion	65
Bibliographie	66

Table des figures

2.1	L'architecture à deux étapes de retriever-reader de DrQA	14
2.2	Récupération Dense (DPR)	15
2.3	L'Approche Retrieval-Augmented Generation (RAG)	17
2.4	Architecture de base et de grande dimension de BERT	19
2.5	T5 (Transformateur de transfert texte-à-texte)	21
4.1	Architecture globale du système ODQA	27
4.2	Architecture Dense Passage Retrievel (DPR)	32
4.3	Architecture BERT for QA	33
4.4	Fine-tuning avec LoRA	34
5.1	Pipeline de préparation d'un système de recherche DPR	41
5.2	Pipeline d'entraînement Question-Answering avec LoRA	42
5.3	Pipeline d'inférence Question-Answering	43
6.1	Performance de Valisation set	52
6.2	Performance de Test set	52
6.3	Performance de Reader par type de question	53
6.4	Modèles Baselines vs. Notre Modèle BERT for QA Fine-tuné	55
7.1	Transformation Pipeline	58
7.2	Flux de donnée	59
7.3	Flux de traitement d'une question dans l'interface web de Question–Réponse.	62
7.4	Page d'Accueil	62
7.5	Authentification par Email	63
7.6	Saisie du nom d'usage	63
7.7	Aperçu des exemples d'utilisation	63
7.8	Interface de Chat	63

Liste des tableaux

2.1	Comparaison entre Closed-Domain QA et Open-Domain QA	12
6.1	Correspondance entre rang et valeur RR	46
6.2	Résultats d'évaluation du système de retrieval	46
6.3	Analyse comparative des performances	47
6.4	Distribution des types de questions dans l'ensemble de test.	49
6.5	Performances détaillées par type de question sur l'ensemble de test.	53

Chapitre 1

Contexte général et objectifs

1.1 Contexte et motivation : De la Logique Symbolique aux Réseaux de Neurones Profonds

L'évolution historique des systèmes de réponse aux questions (QA) témoigne d'une transition progressive d'environnements clos et rigides vers une flexibilité universelle. Dès les années 1960, les premiers systèmes utilisaient deux paradigmes majeurs : la recherche d'informations et la représentation sémantique. Ces systèmes initiaux, bien qu'efficaces, étaient strictement limités par leur domaine d'application. Le système BASEBALL, par exemple, permettait d'interroger une base de données sur les statistiques de la Major League Baseball sur une période d'un an, tandis que le système LUNAR, présenté en 1971, répondait à des questions sur l'analyse géologique des roches lunaires rapportées par les missions Apollo. Bien que LUNAR ait démontré une précision de 90% sur son domaine, son architecture reposait sur des bases de données structurées et des règles manuelles incapables de s'adapter à l'imprévisibilité du langage naturel général.

Dans les années 1970 et 1980, des projets comme SHRDLU ont exploré l'interaction dans des mondes simplifiés (le "monde des blocs"), où les règles de physique et de logique étaient explicitement encodées. Cette approche "système expert" a conduit à des outils comme Unix Consultant (UC), capable d'aider les utilisateurs du système d'exploitation Unix grâce à une base de connaissances artisanale. Cependant, ces systèmes se heurtaient au "problème de la fragilité" : toute question sortant du cadre sémantique prévu entraînait un échec total.

Le véritable changement de paradigme s'est opéré avec l'avènement de l'internet et l'explosion des données numériques. L'architecture des systèmes QA est passée de pipelines modulaires complexes — comprenant le traitement des questions, la récupération de passages et l'extraction de réponses — à des modèles de réseaux de neurones profonds capables de traiter des millions d'articles de manière probabiliste. La victoire d'IBM Watson au jeu Jeopardy ! en 2011 a illustré cette transformation. Watson n'était pas seulement une base de données, mais un système hybride capable de raisonner en temps réel sur des allusions culturelles, des jeux de mots et des contextes ambigus en orchestrant des centaines de techniques de traitement du langage naturel (NLP).

- La motivation actuelle pour le développement de l'ODQA réside dans la nécessité de maîtriser cette richesse d'informations numériques croissante dans les secteurs sociaux, environnementaux et biomédicaux. L'utilisateur contemporain ne souhaite plus naviguer dans des index, mais interagir avec une IA capable de synthétiser des faits objectifs pour répondre à des interrogations complexes de manière instantanée.

1.2 Importance des systèmes de Question-Answering en intelligence artificielle

Le Question-Answering est devenu une pierre angulaire de l'IA moderne car il représente l'interface ultime entre l'intelligence computationnelle et la connaissance humaine. L'importance de ces systèmes s'articule autour de trois dimensions majeures : l'efficacité de l'accès à la connaissance, l'amélioration de l'interaction utilisateur et l'autonomisation des secteurs critiques.

Transformation de l'Accès à l'Information

Contrairement aux moteurs de recherche qui imposent à l'utilisateur une charge cognitive de lecture et de synthèse, les systèmes ODQA déterminent le contexte derrière une question, extraient l'information pertinente et la présentent de manière concise. Cette capacité réduit considérablement le temps passé à mémoriser des faits ou à effectuer des recherches manuelles laborieuses. Pour les entreprises, cela se traduit par une intégration logicielle 40 à 60% plus rapide grâce à l'utilisation de bibliothèques de prompts pré-optimisées et de bases de connaissances automatisées.

Impact dans les Domaines Spécialisés

L'importance de l'ODQA est particulièrement visible dans des secteurs à forts enjeux comme la santé, le droit et l'éducation :

1. **Santé et Biomédecine** : Les systèmes QA permettent aux professionnels de santé d'accéder rapidement à des protocoles de diagnostic ou à des études cliniques récentes, améliorant ainsi la précision des soins.
2. **Droit et Conformité** : Dans le domaine juridique, où l'ancrage factuel est vital, les modèles ODQA aident à naviguer dans la jurisprudence et à vérifier la conformité des documents, à condition qu'ils puissent citer leurs sources de manière fiable.
3. **Éducation** : L'intégration de chatbots QA dans l'enseignement supérieur permet aux étudiants d'explorer des informations au-delà du curriculum statique, favorisant un engagement actif et une recherche d'information dynamique en temps réel sur le web.

Vers une IA Agentique et Multimodale

En 2025 et 2026, l'importance de l'ODQA s'étend à la création d'IA agentiques capables de raisonner de manière délibérée (pensée de "Système 2"). Ces systèmes ne se contentent plus de texte, mais intègrent des modalités visuelles, auditives et sensorielles. Par exemple, les modèles multimodaux de grande taille (LMM) peuvent désormais analyser des graphiques financiers complexes ou des vidéos publicitaires pour répondre à des questions sur les émotions suscitées ou

les stratégies de persuasion employées. Cette convergence fait de l'ODQA un outil indispensable pour la surveillance de sécurité, la robotique collaborative et les systèmes d'infodivertissement embarqués.

1.3 Problématique de l'Open-Domain Question Answering

Malgré les avancées spectaculaires, l'ODQA fait face à des défis structurels, théoriques et pratiques qui limitent son déploiement universel et sa fiabilité absolue. Ces problématiques constituent le cœur de la recherche actuelle.

Le Dilemme du Raisonnement Multi-étapes (Multi-hop)

L'une des limites majeures des systèmes actuels de type "récupérer puis lire" (retrieve-and-read) est leur difficulté à traiter des questions complexes nécessitant plusieurs sauts logiques. Une question telle que "Dans quelle ville est né l'acteur qui a joué dans Oppenheimer ?" nécessite d'abord d'identifier l'acteur (Cillian Murphy) avant de pouvoir chercher son lieu de naissance. Si le document initial ne contient pas le lien sémantique direct avec la question finale, l'extracteur échoue à fournir le contexte nécessaire au lecteur. Ce problème de "découverte de connaissances" exige des architectures capables d'itérer entre la lecture et la génération de nouvelles requêtes, un processus coûteux en termes de calcul et complexe à superviser.

Fiabilité et Phénomène d'Hallucination

Le problème le plus critique des modèles basés sur les grands modèles de langage (LLM) est l'hallucination factuelle. Ces modèles, étant fondamentalement des moteurs de génération de motifs probabilistes, n'ont aucune compréhension inhérente de la vérité. Les hallucinations peuvent survenir pour plusieurs raisons :

- **Données obsolètes** : Les connaissances des modèles sont figées au moment de l'entraînement, nécessitant des mécanismes externes (comme le RAG) pour l'actualisation en temps réel.
- **Mélange de contextes** : Incapacité à distinguer les sources fiables des contenus satiriques ou biaisés, conduisant à une possible propagation d'erreurs factuelles.
- **Complexité computationnelle** : La complexité quadratique des Transformers ($O(N^2d)$) impose une limite théorique au-delà de laquelle les hallucinations deviennent inévitables, particulièrement pour des tâches de raisonnement complexe.

Lacunes de l'Évaluation

Les métriques traditionnelles comme l'Exact Match (EM) sont de plus en plus contestées car elles sous-estiment la performance réelle des modèles capables de paraphraser ou de fournir des réponses synonymes. De plus, assurer la "fidélité des citations" (citation faithfulness) reste un défi : un modèle peut citer une source correcte mais dont le contenu ne soutient pas réellement l'affirmation générée, ce qui compromet la confiance de l'utilisateur final.

1.4 Objectifs du projet

L'objectif général de ce projet est de concevoir et d'implémenter un système de Question-Answering en domaine ouvert capable de répondre efficacement à des questions exprimées en langage naturel.

Les objectifs spécifiques du projet sont les suivants :

- Comprendre les principes fondamentaux des systèmes de Question-Answering et des architectures Open-Domain QA.
- Mettre en place une architecture ODQA intégrant un module de récupération d'information et un module de compréhension ou de génération de réponses.
- Exploiter des techniques modernes de NLP et de deep learning pour améliorer la compréhension sémantique des questions et des documents.
- Évaluer les performances du système en termes de pertinence, de précision et de robustesse face à la diversité des questions.
- Analyser les limites du système et proposer des pistes d'amélioration pour des travaux futurs.

Chapitre 2

Etat de l'art

2.1 Définition du Question Answering

Le **Question Answering (QA)** est un domaine de recherche en intelligence artificielle qui vise à fournir automatiquement des réponses précises et concises à des questions formulées en langage naturel. Contrairement aux systèmes classiques de recherche d'information, qui retournent une liste de documents classés par pertinence, les systèmes de QA ont pour objectif principal d'extraire et de fournir directement la réponse attendue par l'utilisateur [3].

Le **QA** représente une avancée majeure des technologies de recherche d'information, car il permet un accès plus naturel et plus efficace aux connaissances en interrogeant diverses sources de données et en produisant des réponses courtes, pertinentes et exploitables [4]. Pour atteindre cet objectif, les systèmes de Question Answering combinent plusieurs domaines de recherche complémentaires, notamment la Recherche d'Information (Information Retrieval), l'Extraction d'Information (Information Extraction) et le Traitement Automatique du Langage Naturel (Natural Language Processing) [5].

Depuis les années 1960, de nombreux systèmes de QA ont été développés en s'appuyant sur différentes approches, couvrant une grande variété de domaines, de types de questions, de sources de données et de formes de réponses [6]. Les méthodes récentes exploitent souvent plusieurs sources d'information et des techniques avancées de traitement linguistique afin de comprendre la question, identifier les informations pertinentes et générer une réponse correcte, plutôt que de laisser l'utilisateur interpréter lui-même les documents retournés [7].

2.1.1 Closed-Domain vs Open-Domain Question Answering

La classification primordiale des systèmes de QA repose sur l'étendue du domaine de connaissances couvert [8]. Cette distinction influence non seulement la conception de l'architecture mais aussi le choix des modèles d'apprentissage et des bases de données sous-jacentes.

Le **Question Answering en domaine fermé (Closed-Domain QA)** se concentre sur un ensemble spécifique de sujets ou une expertise sectorielle limitée [9]. Ces systèmes sont conçus

pour comprendre des conversations très structurées ou des corpus de données restreints, comme une base de connaissances médicale, juridique ou technique. Par exemple, un agent conversationnel dédié à la maintenance automobile exploitera des ontologies et des manuels spécifiques pour répondre à des questions sur les pannes de moteur. L'avantage principal de ces systèmes réside dans leur haute précision, car le vocabulaire et les relations entre entités sont prédéfinis. Ils utilisent souvent des modèles d'apprentissage profond complexes adaptés aux nuances sémantiques de leur domaine particulier. Cependant, leur capacité de généralisation est quasi nulle ; un système de QA spécialisé en droit ne pourra pas répondre à une question sur la physique quantique.

À l'inverse, le **Question Answering en domaine ouvert (Open-Domain QA ou ODQA)** s'attaque à des questions sur presque n'importe quel sujet, en s'appuyant sur des bases de connaissances massives et non structurées, telles que Wikipedia ou l'intégralité du Web [10]. La difficulté ici n'est pas seulement l'extraction de la réponse, mais d'abord l'identification des quelques documents pertinents parmi des millions, voire des milliards de sources potentielles [11]. Les systèmes ODQA doivent gérer une ambiguïté contextuelle élevée et des données hautement non structurées. Ils reposent sur des ontologies générales et une connaissance du monde acquise lors de phases d'entraînement massives.

Le tableau suivant synthétise les différences structurelles et opérationnelles entre ces deux approches :

TABLE 2.1 – Comparaison entre Closed-Domain QA et Open-Domain QA

Caractéristique	Closed-Domain QA (Domaine Fermé)	Open-Domain QA (Domaine Ouvert)
Portée des connaissances	Spécifique (ex : médecine, support technique)	Universelle (ex : culture générale, actualités)
Source de données	Base de données propriétaire, ontologies spécialisées	Wikipedia, Common Crawl, Web global
Complexité de recherche	Faible (espace de recherche restreint)	Très élevée (besoin de filtres massifs)
Précision sémantique	Très haute sur les termes techniques	Variable, dépend de la qualité de la récupération
Flexibilité	Rigide, limitée aux données d'entraînement	Élevée, capable de traiter des sujets variés
Exemple type	Chatbot de livraison de pizza	Google Search (avec extraits optimisés)

2.2 Architectures des Systèmes ODQA : Une Évolution en Trois Étapes

2.2.1 L'Approche Traditionnelle : IR + NLP (Retriever-Reader)

L'architecture traditionnelle, qui a dominé jusqu'au milieu des années 2010, repose sur un pipeline modulaire divisé en deux étapes distinctes : le Document Retriever (Récupérateur) et le

Document Reader (Lecteur) [10]. Cette structure est souvent désignée sous le terme de "Machine Reading at Scale" (MRS).

Le Mécanisme de Récupération Statistique

Le rôle du récupérateur est de réduire l'espace de recherche. Face à un corpus comme Wikipedia (plus de 5 millions d'articles), il est impossible d'appliquer des modèles de compréhension complexes à chaque document [12]. Le système utilise donc des méthodes statistiques de recherche d'information (IR) pour identifier les 5 à 10 documents les plus susceptibles de contenir la réponse.

Les techniques fondamentales utilisées sont le TF-IDF (Term Frequency-Inverse Document Frequency) et le BM25 (Best Matching 25) [13]. Le TF-IDF attribue un poids à chaque mot d'un document en fonction de sa fréquence locale et de sa rareté dans l'ensemble du corpus. Cependant, le TF-IDF souffre de deux défauts majeurs : il traite la fréquence des termes de manière linéaire et ne normalise pas correctement la longueur des documents.

Le BM25 a révolutionné cette approche en introduisant la saturation de la fréquence des termes (l'importance d'un mot n'augmente pas indéfiniment s'il se répète 50 fois) et la normalisation de la longueur (pour éviter de favoriser les documents longs qui contiennent naturellement plus de mots) [13].

La formule du score BM25 pour un document D et une requête Q est la suivante :

$$\text{Score}(D, Q) = \sum_{q \in Q} \text{IDF}(q) \cdot \frac{f(q, D) \cdot (k_1 + 1)}{f(q, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Ici, $f(q, D)$ est la fréquence du terme q dans D , $|D|$ est la longueur du document, avgdl est la longueur moyenne des documents du corpus, et k_1 et b sont des hyperparamètres de réglage.

Le Document Reader : Compréhension de Texte par RNN

Une fois les documents sélectionnés, le Document Reader (souvent basé sur des réseaux de neurones récurrents comme les LSTM) entre en scène [14]. Son rôle est d'analyser chaque paragraphe pour identifier le "**span**" (l'intervalle de texte) exact qui constitue la réponse.

Un exemple emblématique est le **système DrQA** développé par Facebook Research [10]. Pour fonctionner à grande échelle, DrQA utilise un hachage de bigrammes pour représenter les articles Wikipedia, permettant une récupération extrêmement rapide. Le lecteur de **DrQA** encode les paragraphes en utilisant des vecteurs de caractéristiques riches, incluant des **word embeddings (Glove)**, des **tags POS (Part-of-Speech)**, et des **entités nommées (NER)** [15].

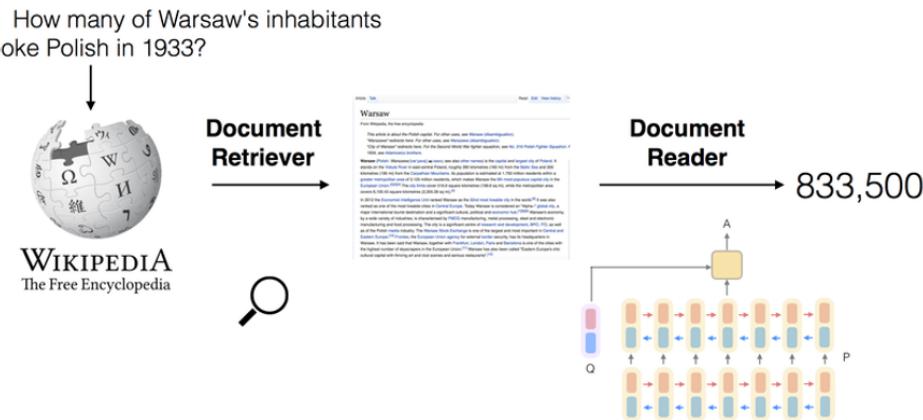


FIGURE 2.1 – L'architecture à deux étapes de retriever-reader de DrQA

2.2.2 L'Approche Basée sur le Deep Learning : Récupération Dense (DPR)

Vers 2019-2020, les limites des méthodes de recherche par mots-clés (sparse retrieval) sont devenues flagrantes. Un système BM25 ne peut pas comprendre que "voiture" et "automobile" sont sémantiquement proches s'il n'y a pas de correspondance exacte de caractères [30]. L'approche par apprentissage profond a introduit la récupération dense (Dense Retrieval).

Dense Passage Retrieval (DPR)

Le DPR remplace les vecteurs statistiques par des représentations vectorielles denses (embeddings) générées par des modèles comme BERT [30]. Le système utilise un encodeur double (dual-encoder) : un pour la question et un pour les passages de documents. Les deux sont projetés dans un espace vectoriel partagé, généralement de 768 dimensions [17].

La pertinence d'un document par rapport à une question est alors calculée par une simple opération de produit scalaire ou de similarité cosinus entre les deux vecteurs :

$$\text{sim}(q, p) = E_Q(q)^\top E_P(p)$$

Cette méthode permet de capturer les relations sémantiques profondes, les synonymes et les paraphrases. Pour effectuer cette recherche parmi des millions de vecteurs en temps réel, on utilise des bibliothèques d'indexation comme FAISS (Facebook AI Similarity Search), qui permettent des recherches de plus proches voisins extrêmement efficaces [18].

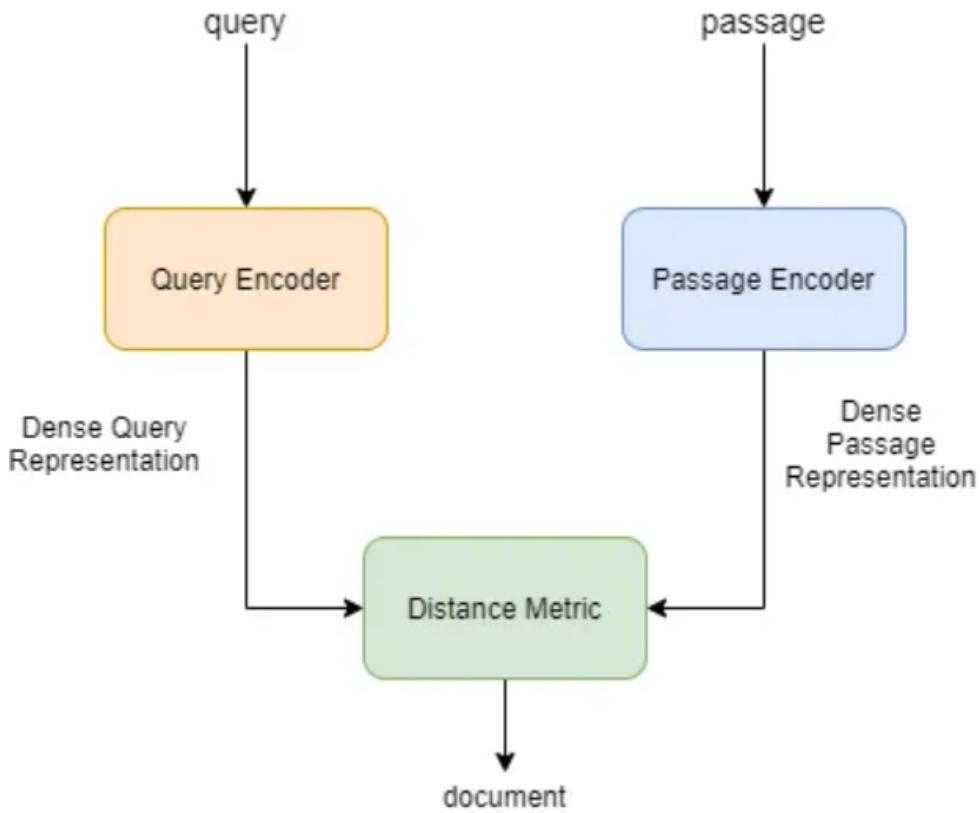


FIGURE 2.2 – Récupération Dense (DPR)

Entraînement et Échantillonnage Négatif

Contrairement au BM25, le DPR doit être entraîné sur des paires (question, passage positif). Pour que le modèle soit performant, il est crucial d'utiliser des "négatifs difficiles" (hard negatives) : des passages qui ressemblent à la question par leurs mots mais ne contiennent pas la réponse, forçant le modèle à apprendre des distinctions sémantiques fines [19].

L'architecture DPR se décompose ainsi :

1. **Encodeur de question** : Transforme la requête en vecteur.
2. **Encodeur de contexte** : Transforme les millions de passages en vecteurs (indexés au préalable).
3. **Base de données vectorielle** : Stocke et recherche les vecteurs (ex : FAISS, Qdrant, Pinecone).
4. **Lecteur (Reader)** : Toujours présent pour extraire la réponse précise des documents récupérés.

2.2.3 L'Approche Retrieval-Augmented Generation (RAG)

L'architecture la plus avancée et la plus utilisée aujourd'hui dans l'industrie est la Génération Augmentée par la Récupération (RAG), formalisée par Lewis et al. en 2020 [20].

Elle combine les forces des systèmes de récupération avec la puissance générative des Large Language Models (LLM) comme GPT-4, Llama 3 ou BART [21].

Mémoire Paramétrique vs Non-Paramétrique

Le concept central du RAG est l'hybridation de deux types de mémoire [20] :

- **Mémoire Paramétrique** : Ce sont les connaissances figées dans les poids du modèle LLM lors de son entraînement initial. Elle est excellente pour la structure du langage mais peut être obsolète ou sujette aux hallucinations.
- **Mémoire Non-Paramétrique** : C'est la base de données externe (documents PDF, sites web, fichiers JSON). Le système consulte cette base dynamiquement au moment de l'inférence.

Le Pipeline RAG

Le processus RAG se déroule en trois étapes fluides [22] :

1. **Récupération (Retrieval)** : Le système cherche les fragments de documents les plus pertinents pour la question de l'utilisateur dans une base de données vectorielle.
2. **Augmentation (Augmentation)** : Les extraits récupérés sont insérés directement dans le "prompt" envoyé au LLM, fournissant ainsi un contexte frais et vérifiable au modèle.
3. **Génération (Generation)** : Le LLM génère une réponse en langage naturel qui synthétise les informations du contexte. Contrairement aux lecteurs extractifs du passé, le RAG peut reformuler, résumer et citer ses sources.

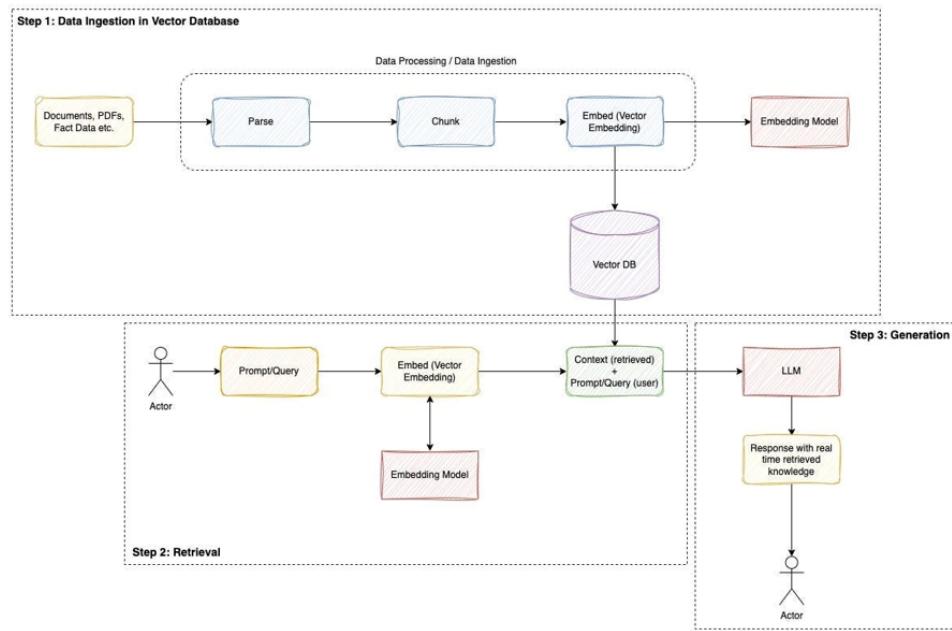


FIGURE 2.3 – L’Approche Retrieval-Augmented Generation (RAG)

Avantages Stratégiques du RAG

Le passage au RAG résout trois problèmes critiques pour les entreprises [23] :

- 1. Réduction des hallucinations** : En forçant le modèle à se baser sur des documents réels, on minimise les risques qu'il invente des faits.
- 2. Mise à jour en temps réel** : Il n'est plus nécessaire de réentraîner le modèle (processus coûteux) pour lui apprendre de nouvelles données; il suffit de mettre à jour la base de documents externe.
- 3. Transparence et Audit** : Le système peut indiquer précisément quel document a été utilisé pour générer la réponse, facilitant la vérification par l'utilisateur.

2.3 Modèles et techniques existants

Les systèmes de Question Answering ont évolué progressivement en suivant les avancées du traitement automatique du langage naturel. Les premières approches reposaient sur des modèles classiques, principalement basés sur des règles linguistiques, des mots-clés et des méthodes statistiques, dont l'objectif était d'identifier des réponses à partir de correspondances lexicales. Bien que ces modèles aient permis les premières applications du Question Answering, leurs performances restaient limitées face à la complexité et à l'ambiguïté du langage naturel. Par la suite, l'introduction de l'apprentissage automatique et des réseaux de neurones a permis d'améliorer la compréhension sémantique des questions et des textes. Cette évolution a conduit à l'émergence des modèles basés sur l'architecture Transformer, qui représentent aujourd'hui l'état de

l'art, notamment à travers des modèles pré-entraînés tels que BERT et ses variantes, largement utilisés dans les systèmes modernes de Question Answering.

2.3.1 Modèles basés sur des transformers

Les transformers constituent une architecture fondamentale de l'IA moderne, en particulier dans le Traitement Automatique du Langage Naturel (TALN) et la modélisation générative. Présentés pour la première fois dans l'article « Attention is All You Need » (Vaswani et al., 2017), les transformers abandonnent la récurrence au profit d'un mécanisme appelé auto-attention, qui permet aux modèles de prendre en compte simultanément toutes les parties de la séquence d'entrée.[24]

BERT et ses variantes

BERT, pour Bidirectional Encoder Representations from Transformers, est un LLM (Large Language Model), entraîné sur un grand corpus de données textuelles et conçu pour générer du texte en langage naturel. Google AI a développé ce modèle en 2018, en utilisant des réseaux neuronaux. Cette technique de Machine Learning permet à l'algorithme de mieux comprendre le contexte d'une requête. Contrairement aux modèles antérieurs qui utilisaient une approche unidirectionnelle pour traiter le texte, BERT est capable de comprendre le contexte bidirectionnel du langage. Il prend donc en compte à la fois les mots précédents et les mots suivants dans une phrase. Cela lui permet de mieux comprendre le sens des mots et des phrases.[25]

Le modèle améliore la compréhension automatique du texte. Il permet d'extraire des informations, d'identifier des entités nommées et de faire de l'analyse textuelle en général. Il peut également générer des résumés automatiques de documents, même longs, en identifiant les parties les plus importantes du texte.. BERT est capable de répondre à des questions posées en langage naturel. Il comprend le contexte et génère des réponses pertinentes à partir de documents ou de corpus de texte.[25]

Modèle BERT pré-entraîné

Le modèle BERT est pré-entraîné sur de grandes quantités de texte non étiqueté afin d'apprendre des représentations contextuelles[26]

. - BERT est pré-entraîné sur une grande quantité de données textuelles non étiquetées. Le modèle apprend les plongements contextuels, qui sont des représentations des mots tenant compte de leur contexte dans une phrase[26]

. - BERT effectue diverses tâches de pré-entraînement non supervisées. Par exemple, il peut apprendre à prédire les mots manquants dans une phrase (modèle de langage masqué ou MLM), à comprendre la relation entre deux phrases ou à prédire la phrase suivante dans une paire[26].

Architecture BERT

L'architecture de BERT est un encodeur transformeur bidirectionnel multicouche, très similaire au modèle transformeur. Une architecture transformeur est un réseau encodeur-décodeur qui utilise l'auto-attention côté encodeur et l'attention côté décodeur[26].

1. **BERT_{BASE}** possède 12 couches dans la pile d'encodeurs, tandis que **BERT_{LARGE}** en possède 24. Ceci est supérieur à l'architecture Transformer décrite dans l'article original, qui comporte 6 couches d'encodeur.
2. Les architectures **BERT** (BASE et LARGE) disposent également de réseaux de propagation avant plus étendus, avec respectivement 768 et 1024 unités cachées, ainsi qu'un plus grand nombre de têtes d'attention (12 et 16 respectivement), comparativement à l'architecture Transformer proposée dans l'article original, qui comporte 512 unités cachées et 8 têtes d'attention.
3. **BERT_{BASE}** contient environ 110 millions de paramètres, tandis que **BERT_{LARGE}** en compte environ 340 millions.

La figure suivante illustre les différences architecturales entre les versions **BERT_{BASE}** et **BERT_{LARGE}**.

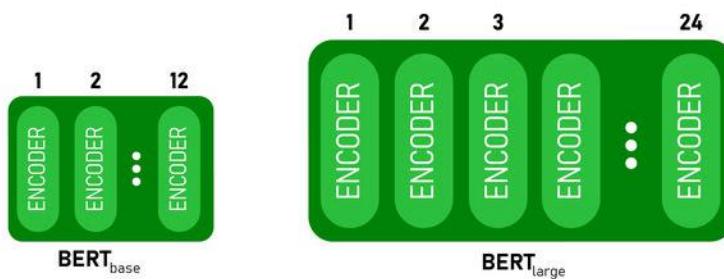


FIGURE 2.4 – Architecture de base et de grande dimension de BERT

BERT pour le Question Answering

Les modèles basés sur l'architecture BERT ont démontré une efficacité remarquable dans les tâches de Question Answering. Grâce à sa capacité à comprendre le contexte bidirectionnel du langage, BERT peut être adapté afin d'identifier avec précision les réponses à des questions formulées en langage naturel à partir d'un texte donné. Dans les systèmes de Question Answering, BERT est généralement utilisé dans une phase de fine-tuning, où le modèle pré-entraîné est ajusté pour prédire les segments de texte correspondant à la réponse, en se basant sur la relation entre la question et le contexte fourni.

Dans une tâche de Question Answering, le modèle BERT est généralement appliqué selon une approche de type *extractive*, où la réponse est extraite directement à partir d'un texte de référence. Le modèle reçoit en entrée une paire composée de la question et du passage de texte (contexte), qui sont concaténés et séparés par des tokens spéciaux. Cette représentation permet à BERT d'analyser simultanément la relation entre la question et le contexte.

Lors de la phase de fine-tuning, le modèle est entraîné à prédire les positions de début et de

fin de la réponse correcte dans le texte fourni. Pour chaque mot du passage, BERT estime la probabilité qu'il corresponde au début ou à la fin de la réponse. La séquence de mots comprise entre ces deux positions constitue alors la réponse générée par le système.

Cette approche permet au modèle de tirer pleinement parti de sa compréhension contextuelle bidirectionnelle, ce qui améliore significativement la précision des réponses, en particulier pour des questions factuelles et contextuelles. Elle est aujourd’hui largement utilisée dans de nombreux systèmes de Question Answering basés sur des modèles pré-entraînés.

Quelques Variantes de BERT

- **RoBERTa** : s'appuie sur la stratégie de masquage du langage de BERT, permettant au système d'apprendre à prédire des sections de texte intentionnellement cachées dans des exemples de langage non annotés. Implémenté en PyTorch, RoBERTa modifie des hyperparamètres clés de BERT, notamment en supprimant l'objectif de pré-entraînement « phrase suivante » et en utilisant des mini-lots et des taux d'apprentissage beaucoup plus importants. Ceci permet à RoBERTa d'améliorer la modélisation du langage masqué par rapport à BERT et d'obtenir de meilleures performances pour les tâches en aval[27].
- **FinBERT** : est un modèle de traitement automatique du langage naturel (TALN) pré-entraîné pour l'analyse des sentiments dans les textes financiers. Il est construit en perfectionnant le modèle de langage BERT dans le domaine financier, à l'aide d'un vaste corpus financier, afin de l'affiner pour la classification des sentiments financiers. Même avec un ensemble de données très réduit, il était désormais possible de tirer parti des modèles TALN de pointe. FinBERT excelle dans l'identification du sentiment positif ou négatif de phrases que d'autres algorithmes qualifient à tort de neutres, probablement grâce à son utilisation du contexte dans les textes financiers[27].
- **ALBERT** : Le modèle ALBERT a été proposé dans l'article « ALBERT : A Lite BERT for Self-supervised Learning of Language Representations » par Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma et Radu Soricut. Il présente deux techniques de réduction de paramètres afin de diminuer la consommation de mémoire et d'accélérer l'entraînement de BERT. D'abord diviser la matrice d'intégration en deux matrices plus petites et l'utilisation de couches répétitives réparties entre les groupes. L'architecture d'ALBERT repose sur la même base que celle de BERT, utilisant un encodeur Transformer avec des non-linéarités GELU. De plus, à l'instar de BERT, ALBERT a été pré-entraîné sur les jeux de données Wikipedia en anglais et Book Corpus, contenant 16 Go de données non compressées[27].

En résumé, des variantes de BERT ont été proposées pour améliorer ses performances ou l'adapter à des contextes précis. RoBERTa renforce l'efficacité de l'entraînement, FinBERT se spécialise dans le domaine financier, et ALBERT réduit la complexité du modèle tout en maintenant de bonnes performances. Ces approches montrent que le choix de la variante dépend essentiellement des objectifs et des contraintes de l'application.

2.3.2 Modèles spécialisés pour le Question Answering

Au-delà de BERT, plusieurs modèles spécialisés ou améliorés ont été proposés afin d'optimiser les performances des systèmes de Question Answering. Ces modèles s'appuient généralement sur l'architecture Transformer tout en introduisant des stratégies de pré-entraînement ou des mécanismes d'attention améliorés, leur permettant d'obtenir de meilleurs résultats sur des benchmarks standards.

Modèles génératifs

Les modèles génératifs, tels que T5 (Text-to-Text Transfer Transformer) et BART, abordent le Question Answering comme une tâche de génération de texte. Contrairement aux approches extractives, ces modèles ne se limitent pas à sélectionner un segment du texte de contexte, mais génèrent directement la réponse sous forme de séquence de mots. Cette approche permet de produire des réponses plus naturelles et plus flexibles, en particulier lorsque la réponse ne correspond pas exactement à un passage du texte fourni.

T5 (Text-to-Text Transfer Transformer) est un modèle basé sur l'architecture transformable, développé par Google Research. Contrairement aux modèles de traitement automatique du langage naturel (TALN) traditionnels, qui possèdent des architectures spécifiques à chaque tâche, T5 considère chaque tâche de TALN comme un problème de conversion de texte en texte. Ce cadre uniifié permet son application à diverses tâches telles que la traduction, la synthèse et la réponse aux questions[28].

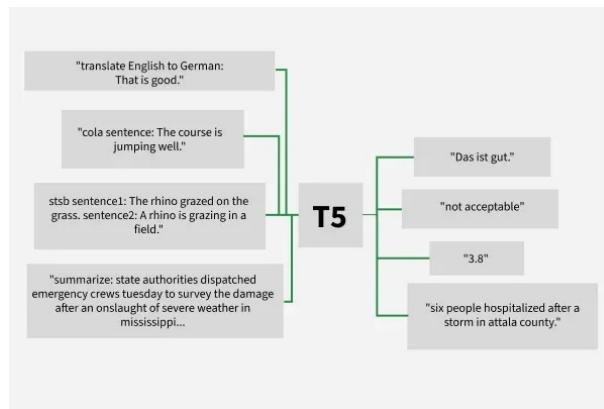


FIGURE 2.5 – T5 (Transformateur de transfert texte-à-texte)

Le modèle T5 suit un principe simple mais efficace : il convertit tous les problèmes de traitement automatique du langage naturel (TALN) en un format *texte-à-texte*. T5 utilise une architecture *encodeur-décodeur* similaire aux modèles séquence-à-séquence de type Transformer. Son fonctionnement peut être résumé comme suit[28] :

1. **Formulation des tâches en mode texte-à-texte** : Au lieu de traiter séparément les différentes tâches de TALN, chaque problème est reformulé en une entrée et une sortie textuelles.
2. **Encodage de l'entrée** : Le texte d'entrée est tokenisé à l'aide de *SentencePiece*, puis

transmis à l'encodeur qui génère une représentation contextuelle.

3. **Décodage du résultat** : Le décodeur prend la représentation encodée et génère le texte de sortie de manière *autorégressive*.
4. **Entraînement du modèle** : T5 est pré-entraîné à l'aide d'une fonction de débruitage (*denoising*) où des portions de texte sont masquées et le modèle apprend à les reconstruire. Il est ensuite affiné (*fine-tuning*) pour différentes tâches spécifiques.

Modèles à récupération d'information

Dans les scénarios de Question Answering à grande échelle, le contexte pertinent n'est pas toujours disponible. Les modèles à récupération d'information, tels que RAG (Retrieval-Augmented Generation) -déjà mentionné-, combinent des techniques de recherche de documents avec des modèles génératifs. Le système commence par identifier les passages les plus pertinents à partir d'une large base de connaissances, puis génère une réponse en s'appuyant sur les informations récupérées. Cette approche est particulièrement adaptée aux systèmes de Question Answering ouverts, où les connaissances nécessaires à la réponse ne sont pas limitées à un corpus restreint.

2.4 Applications réelles du Question Answering (QA)

Les systèmes de Question Answering (QA) sont largement utilisés dans de nombreux contextes grâce à leur capacité à fournir des réponses précises à partir de grandes quantités de données. Voici quelques exemples réels :

- **ChatGPT** : un modèle conversationnel développé par OpenAI qui peut répondre à des questions générales, expliquer des concepts ou aider à la rédaction de textes.
- **Google Search et Google Assistant** : utilisent des techniques de QA pour fournir des réponses directes aux questions des utilisateurs, en extrayant l'information depuis le web ou des bases de données structurées.
- **IBM Watson** : célèbre pour avoir participé au jeu télévisé *Jeopardy!*, il est utilisé dans le domaine médical et commercial pour répondre à des questions complexes en langage naturel.
- **Siri et Alexa** : assistants vocaux d'Apple et Amazon qui répondent aux questions des utilisateurs, donnent des recommandations et exécutent des commandes vocales.
- **Systèmes de support client automatisé** : de nombreuses entreprises intègrent des chatbots basés sur le QA pour répondre instantanément aux questions fréquentes des clients, par exemple *Zendesk Answer Bot*.

Chapitre 3

Dataset utilisée

3.1 Présentation du Dataset

Le dataset principal utilisé dans ce projet est le **Natural Questions (NQ) dataset**, dans sa version simplifiée et prétraitée : simplified-nq-train.jsonl. Ce dataset provient de la compétition et des travaux de recherche associés à la tâche Open-Domain Question Answering, initialement publié par Google en 2019. Il est largement considéré comme l'un des benchmarks les plus réalistes et challengants pour les systèmes ODQA, car les questions sont issues de recherches réelles effectuées par des utilisateurs sur le moteur de recherche Google. La version simplifiée utilisée ici est disponible publiquement sur Kaggle sous le nom « The Natural Questions Dataset » et contient uniquement le fichier d'entraînement (train) au format JSON Lines (.jsonl).

3.2 Caractéristiques du Dataset

Nombre d'exemples : Environ 307 373 exemples dans l'ensemble d'entraînement (*simplified-nq-train.jsonl*).

Origine des questions : Questions réelles anonymisées issues des requêtes des utilisateurs du moteur de recherche Google. Contrairement à d'autres jeux de données de questions-réponses, les questions ne sont pas rédigées par des annotateurs, mais proviennent de recherches authentiques.

Source des documents : Pages Wikipedia (version anglaise). Chaque exemple est associé à une page Wikipedia complète, fournie sous forme de texte brut enrichi de balises HTML légères (par exemple <P>, <Table>).

Structure des annotations :

- **Long answer candidates** : Ensemble de segments de texte définis par des indices de

tokens (*start_token*, *end_token*), correspondant généralement à des paragraphes, sections ou tableaux de la page Wikipedia. Certains de ces segments sont marqués par l'attribut *top_level*, indiquant des passages de haut niveau (paragraphes principaux).

- **Annotations humaines** : Dans la version *simplified* du dataset, chaque exemple d'entraînement est associé à **une seule annotation humaine**. Cette annotation contient :

- Un *long answer* : passage long (souvent un paragraphe) contenant la réponse, ou une valeur nulle si aucune réponse n'est présente sur la page.
- **short answers** : réponses courtes et précises.
- Un champ *yes_no_answer* indiquant si la réponse est de type oui/non (YES, NO, ou NONE).

3.3 Prétraitement et Filtrage

Le prétraitement suit une approche rigoureuse en plusieurs étapes visant à garantir la qualité et la cohérence des données d'entraînement. Ces étapes sont essentielles pour optimiser les performances des modèles de traitement du langage naturel.

3.3.1 Filtrage des Exemples Utilisables

La première étape consiste en un filtrage minutieux des exemples. Seuls sont retenus les exemples possédant à la fois une réponse longue (*long_answer*) et une réponse courte (*short_answer*) valides. Cette condition est cruciale pour les tâches d'extraction de réponses. Les exemples dépourvus d'annotations complètes ou présentant des réponses manquantes sont systématiquement exclus. Enfin, une validation de la cohérence entre les positions des réponses courtes et longues est effectuée pour s'assurer que les réponses courtes se situent bien à l'intérieur des segments de texte identifiés comme réponses longues, éliminant ainsi les incohérences potentielles dans les annotations.

3.3.2 Construction du Corpus de Passages

Pour entraîner efficacement le module de récupération (*retrieval*), un corpus structuré de passages est construit. Deux catégories de passages sont extraites de chaque document source. Les **Passages Gold** correspondent aux segments de texte contenant la réponse correcte annotée (*long_answer*). En parallèle, des **Passages Distracteurs** sont échantillonnés de manière aléatoire, généralement entre 4 et 5 par document. Ces distracteurs sont choisis parmi les autres candidats de haut niveau (*top_level*) du même document qui, eux, ne contiennent pas la réponse. Cette stratégie de construction crée un corpus équilibré, permettant au système d'apprendre à discriminer et à retrouver les passages pertinents parmi un ensemble qui inclut des passages non pertinents, simulant ainsi une tâche de recherche réelle.

L’application de cette méthodologie sur l’ensemble du jeu de données d’entraînement a permis de constituer un corpus substantiel de **1 368 573 passages** au total. Cette échelle importante garantit une diversité suffisante pour l’apprentissage du modèle de récupération, couvrant un large éventail de sujets et de contextes présents dans la collection documentaire source.

3.3.3 Alignement des Positions

La dernière étape concerne l’alignement précis des positions des réponses dans le texte. Pour chaque exemple retenu, les positions relatives des réponses courtes sont recalculées par rapport au début du passage long qui les contient. Ensuite, afin de préparer les données pour un modèle comme BERT, les indices de tokens bruts sont convertis en indices de caractères compatibles avec le tokenizer spécifique utilisé. Cet alignement nécessite un *mapping* précis entre les *offsets* produits par le tokenizer et les positions originales des réponses dans le texte, garantissant que les spans de réponse annotés correspondent exactement aux tokens d’entrée du modèle, ce qui est fondamental pour la tâche d’extraction de réponse précise.

3.3.4 Division du Dataset

La séparation des données en ensembles distincts est effectuée de manière à garantir une évaluation robuste et reproductible des modèles. La répartition adoptée suit les proportions standards dans le domaine de l’apprentissage automatique.

- **Ensemble d’entraînement** : 90% des exemples après prétraitement, soit **96 233 exemples**. Cet ensemble, le plus volumineux, est utilisé pour l’apprentissage des paramètres du modèle.
- **Ensemble de validation** : 5% des exemples, correspondant à **5 346 exemples**. Cet ensemble sert au réglage des hyperparamètres (comme le taux d’apprentissage), au suivi de l’apprentissage pour détecter le surapprentissage (*overfitting*), et à la sélection des meilleurs modèles au cours de l’entraînement.
- **Ensemble de test** : 5% des exemples, soit **5 347 exemples**. Cet ensemble est strictement réservé à l’évaluation finale des performances du modèle. Il n’est utilisé à aucune étape de l’entraînement ou du réglage, fournissant ainsi une estimation non biaisée de la capacité de généralisation du modèle.

Pour assurer la reproductibilité des expériences, une graine aléatoire (*random seed*) fixée à **42** est utilisée lors de la division aléatoire stratifiée. Cela garantit que la même partition des données est obtenue à chaque exécution du script de prétraitement.

Chapitre 4

Architecture du Système Proposé

4.1 Vue d'Ensemble de l'Architecture

Le système implémente une architecture classique de Question-Réponse sur Documents Ouverts (ODQA – Open-Domain Question Answering), organisée en un pipeline séquentiel à deux étages (*Two-Stage Pipeline*). Cette décomposition permet de traiter efficacement le problème en le divisant en deux sous-tâches spécialisées : la recherche rapide de documents pertinents, suivie d'une analyse précise du contenu.[29]

- **Retriever (Module de récupération)** : Ce premier module a pour objectif de filtrer rapidement l'immense corpus de documents (ici, les pages Wikipedia) pour ne conserver qu'un petit ensemble de passages les plus susceptibles de contenir la réponse à la question posée. Notre implémentation utilise une approche de **Dense Passage Retrieval (DPR)**. Contrairement aux méthodes traditionnelles de recherche lexicale (comme BM25), DPR représente à la fois les questions et les passages dans un espace vectoriel continu et dense à l'aide de modèles de langage encodeurs. La pertinence est ensuite calculée par similarité cosinus entre ces représentations vectorielles, permettant une recherche sémantique plus nuancée.
- **Reader (Module de compréhension)** : Ce second module reçoit la question de l'utilisateur ainsi que le petit nombre de passages (généralement entre 5 et 100) pré-sélectionnés par le *Retriever*. Sa tâche est d'analyser finement chaque passage pour déterminer s'il contient une réponse et, le cas échéant, d'en extraire le segment textuel exact. Notre implémentation est basée sur un modèle **BERT for Question Answering** fine-tuné. Le modèle est entraîné à traiter la paire (question, passage) et à prédire simultanément un score de pertinence pour le passage ainsi que les positions de début et de fin de la réponse courte à l'intérieur de celui-ci.

4.1. Vue d'Ensemble de l'Architecture

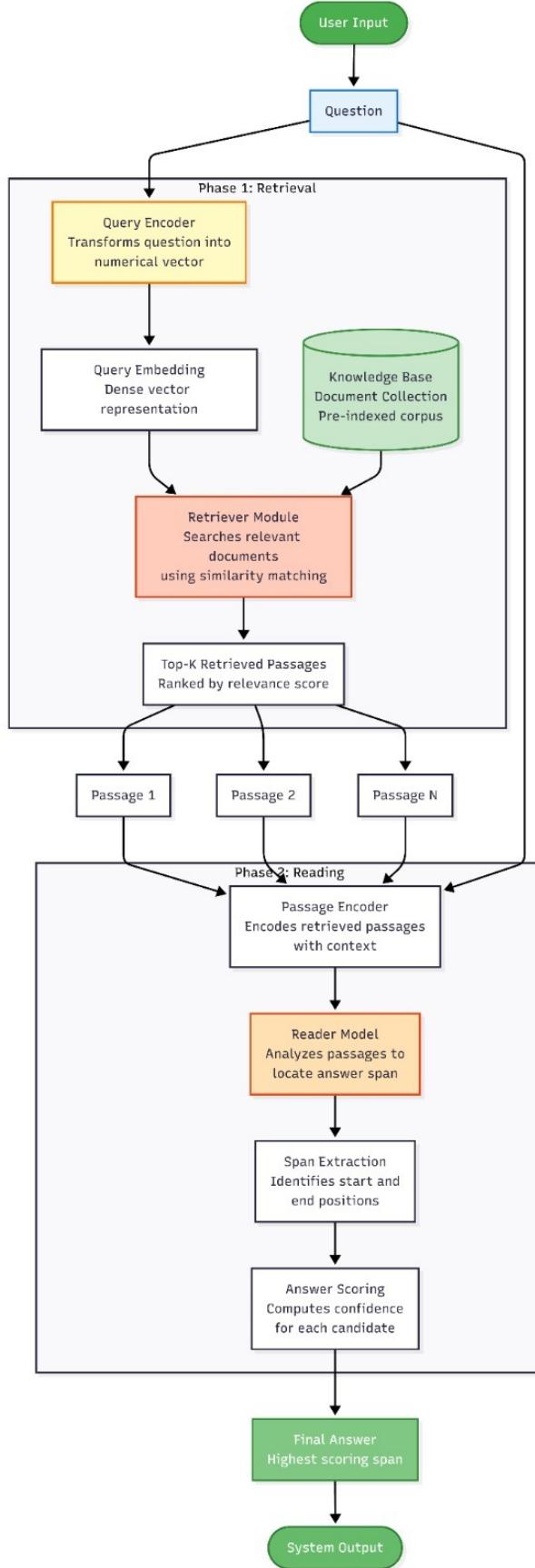


FIGURE 4.1 – Architecture globale du système ODQA

La force de cette architecture réside dans la complémentarité des deux modules : le *Retriever* réduit la complexité du problème en fournissant un contexte ciblé, permettant ainsi au *Reader*, plus coûteux en calcul, de se concentrer sur une analyse approfondie d'un nombre restreint de documents.

4.2 Module de traitement des questions

Le traitement des questions constitue la première étape du pipeline, où la requête utilisateur est préparée pour être utilisée par les deux composants principaux du système (*Retriever* et *Reader*). Ce module assure la transformation du texte brut en une représentation adaptée à chaque modèle.

4.2.1 Nettoyage et normalisation

Le traitement des questions suit un processus unifié pour les deux modules :

Conversion en minuscules

Toutes les questions sont converties en minuscules pour assurer une insensibilité à la casse (*case-insensitivity*). Cette étape permet de traiter de manière équivalente les mêmes mots qu'ils soient écrits en majuscules ou en minuscules, réduisant ainsi la redondance et améliorant la généralisation du modèle.

Suppression des articles

Pour le calcul de métriques spécifiques, notamment lors de l'évaluation, les articles définis et indéfinis courants ("a", "an", "the") sont supprimés. Cette normalisation vise à focaliser l'évaluation sur les termes porteurs de sens (*content words*) et à réduire le biais potentiel introduit par des différences stylistiques mineures dans la formulation des questions.

Normalisation des espaces blancs

Les espaces blancs excédentaires (espaces multiples, tabulations, sauts de ligne) sont uniformisés en un espace simple. Cette opération garantit une représentation cohérente du texte, évitant que des variations d'espacement n'affectent le processus de tokenisation ou la représentation sémantique.

Élimination de la ponctuation lors de l'évaluation

Dans la phase d'évaluation finale, la ponctuation est supprimée des prédictions du modèle et des réponses de référence (*ground truth*). Cette normalisation permet une comparaison plus équitable et robuste, en se concentrant sur la correspondance sémantique du contenu textuel plutôt que sur des détails de formatage.

4.2.2 Tokenisation

La tokenisation diffère selon le composant cible, utilisant les tokenizers spécifiques aux modèles pré-entraînés employés.

- **Pour le Retriever :** La tokenisation est effectuée à l'aide du `DPRQuestionEncoderTokenizerFast`. Ce tokenizer suit le schéma de tokenisation *subword* (par exemple, WordPiece) optimisé pour l'encodeur de questions DPR.
- **Pour le Reader :** La tokenisation utilise le `BertTokenizerFast` (version `bert-base-uncased`). Ce tokenizer transforme également le texte en tokens subword, garantissant une compatibilité parfaite avec l'architecture BERT sous-jacente du module de lecture.

4.2.3 Encodage sémantique

- **Pour le Retriever :** La représentation sémantique de la question est cruciale pour l'étape de récupération dense. La question tokenisée est passée au modèle `DPRQuestionEncoder` pré-entraîné (spécifiquement `facebook/dpr-question_encoder-single-nq-base`). Ce dernier produit un vecteur de représentation (*embedding*) dense de **768 dimensions**. Le vecteur utilisé est généralement la sortie de la couche de *pooling* (`pooler_output`). Enfin, cet embedding est normalisé en norme L2, c'est-à-dire que le vecteur est ramené à une longueur unitaire. Cette normalisation est essentielle car la métrique de similarité utilisée pour la recherche est le produit scalaire (ou le cosinus), qui devient alors équivalent au cosinus entre les vecteurs lorsque ceux-ci sont normalisés.

4.3 Module de Récupération d'Information (Retrieval)

Le module de récupération a pour objectif de sélectionner rapidement, dans un vaste corpus de documents, les passages les plus pertinents pour répondre à une question donnée.

4.3.1 Architecture Dense Passage Retrieval (DPR)

Le système utilise l'architecture **Dense Passage Retrieval (DPR)**, une approche basée sur l'apprentissage profond qui repose sur des représentations vectorielles denses (embeddings) plutôt que sur des correspondances lexicales traditionnelles (comme TF-IDF ou BM25).[30]

- **Question Encoder :** `facebook/dpr-question_encoder-single-nq-base`
 - Transforme la question de l'utilisateur en un vecteur dense de 768 dimensions.
 - Basé sur une architecture BERT qui a été spécifiquement pré-entraînée et fine-tunée

sur le dataset Natural Questions.

- **Context Encoder** : `facebook/dpr-ctx_encoder-single-nq-base`

- Encode chaque passage ou paragraphe du corpus de documents en un vecteur dense de la même dimension (768 dimensions).
- Il partage la même architecture sous-jacente que le Question Encoder, formant une paire d'encodeurs siamois optimisés conjointement.

4.3.2 Indexation des Documents

Avant toute recherche, l'ensemble du corpus doit être indexé pour permettre des requêtes rapides.

Construction de l'Index FAISS :

- **Type d'index** : `IndexFlatIP` (Index pour Produit Scalaire Interne - *Inner Product*). Cet index calcule de manière exacte (et non approximative) la similarité, garantissant la précision des résultats.
- **Métrique de similarité** : Le produit scalaire entre les vecteurs question et passage. Comme les vecteurs sont préalablement normalisés en norme L2, cela est mathématiquement équivalent au cosinus de l'angle entre eux (cosinus similarity).

Processus d'indexation :

1. Encodage des passages par lots (taille de lot, *batch size*, de 32) à l'aide du Context Encoder.
2. Normalisation L2 de chaque embedding de passage généré.
3. Ajout séquentiel de ces embeddings normalisés dans l'index FAISS.
4. Sauvegarde de l'index vectoriel sur disque (par exemple sous le nom `passage.index`) pour une réutilisation ultérieure.

Optimisations :

- Utilisation du GPU pour accélérer massivement le processus d'encodage des passages, avec support du multi-GPU via `DataParallel`.
- Troncature des passages à un maximum de 512 tokens, limite d'entrée du modèle BERT sous-jacent.
- Stockage séparé des textes bruts des passages (via sérialisation `pickle`) et de l'index vectoriel FAISS, permettant une gestion efficace de la mémoire et un accès rapide.

4.3.3 Recherche Sémantique

Lorsqu'une nouvelle question est posée, le processus de recherche s'active.

Algorithme de Récupération :

1. Encodage de la question en un vecteur dense via le Question Encoder.
2. Normalisation L2 de ce vecteur question.
3. Recherche dans l'index FAISS.
4. Retour des identifiants des passages correspondants et de leurs scores de similarité, triés par ordre décroissant.

Configuration :

- **K (nombre de passages)** : Paramètre ajustable. Une valeur typique de 5 passages est utilisée pour fournir un contexte suffisant au module Reader.
- **Temps de recherche** : Quasi-instantané (inférieur à 10 ms) même pour des corpus de plusieurs millions de passages, grâce à l'optimisation de FAISS.

4.3.4 Sélection des Passages Pertinents

La sélection finale des passages à transmettre au Reader repose sur une combinaison de facteurs :

- **Similarité sémantique** : Le score de produit scalaire (cosinus) est l'indicateur principal de pertinence sémantique.
- **Classement** : Les passages sont naturellement ordonnés par ce score de pertinence décroissante.
- **Diversité** : La stratégie d'indexation qui inclut des passages distracteurs contribue à ce que l'ensemble de passages retournés offre une couverture variée du document source, évitant ainsi une redondance excessive.

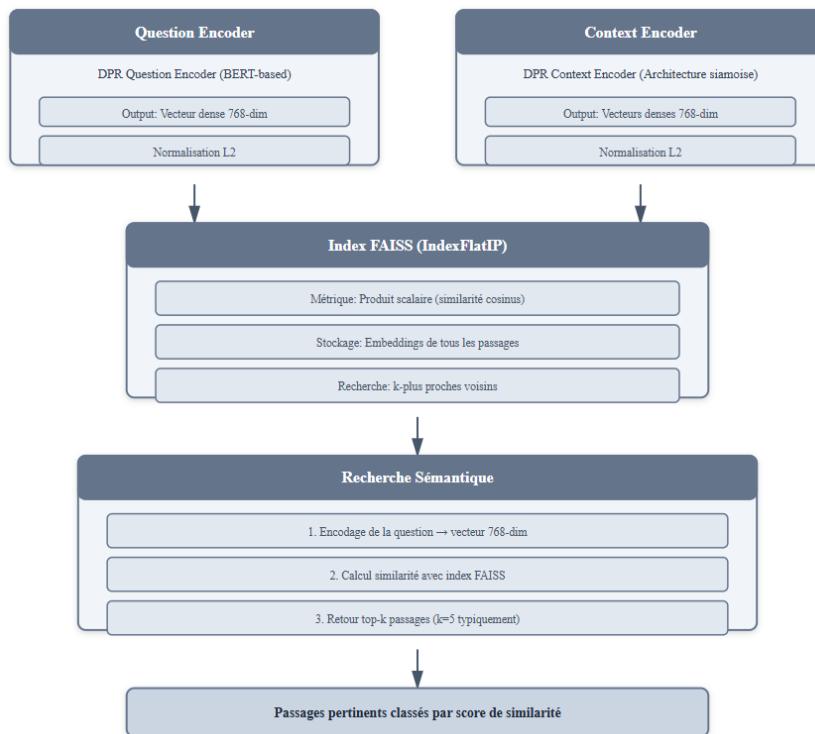


FIGURE 4.2 – Architecture Dense Passage Retrievel (DPR)

4.4 Module de Lecture et Compréhension (Reader)

Le module de lecture a pour responsabilité d'analyser les passages récupérés par le Retriever afin d'y localiser la réponse précise à la question de l'utilisateur.

4.4.1 Architecture du Modèle

Modèle de Base : *BertForQuestionAnswering* basé sur BERT-base-uncased[31]

Architecture : 12 couches de Transformer avec 12 têtes d'attention par couche.

Paramètres : Environ 110 millions de paramètres.

Vocabulaire : 30 000 tokens WordPiece (version non sensible à la casse).

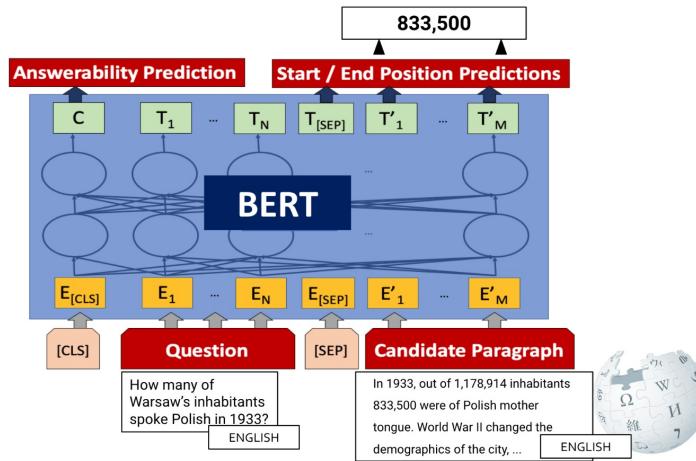


FIGURE 4.3 – Architecture BERT for QA

Adaptation pour la tâche de Question-Réponse :

- Le modèle de base est utilisé via la classe `BertForQuestionAnswering`.
 - Couche de classification à deux têtes :
 - **Tête de début (Start Head)** : Une couche linéaire qui prédit, pour chaque token du passage, la probabilité qu'il marque le début de la réponse.
 - **Tête de fin (End Head)** : Une couche linéaire similaire qui prédit la probabilité que chaque token marque la fin de la réponse.
 - Format d'entrée : [CLS] Question [SEP] Passage [SEP]
 - Format de sortie : Deux vecteurs de logits, un pour la position de début et un pour la position de fin, de longueur égale au nombre total de tokens de la séquence d'entrée.

4.4.2 Fine-tuning Efficace avec LoRA

Pour réduire de manière significative les coûts computationnels et mémoire tout en conservant les performances, le système utilise la technique d'adaptation par bas rang (Low-Rank Adaptation - LoRA)[32].

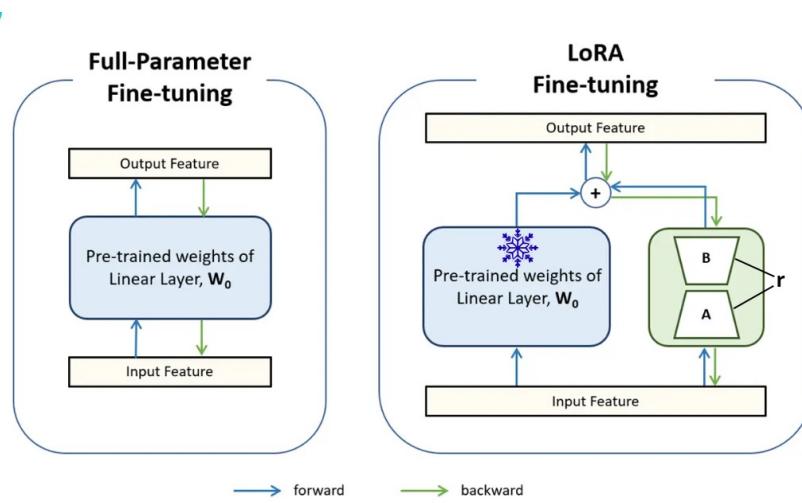


FIGURE 4.4 – Fine-tuning avec LoRA

Configuration LoRA appliquée au modèle BERT for Question Answering :

- **Rang (r)** : 8. C'est la dimension intrinsèque des matrices de mise à jour de low-rank injectées.
- **Alpha** : 16. Facteur d'échelle pour la stabilité de l'apprentissage.
- **Dropout** : 0.1, appliqué aux couches LoRA pour la régularisation.
- **Modules ciblés** : Les matrices de projection de requête (**query**) et de valeur (**value**) dans chaque mécanisme d'attention de chaque couche Transformer.
- **Paramètres entraînables** : Seules les petites matrices de bas rang de LoRA sont mises à jour, ce qui représente environ 0.3% des paramètres totaux du modèle (soit environ 300 000 paramètres au lieu des 110 millions du modèle complet).

4.4.3 Optimisation de l'Entraînement avec LoRA

Pour réduire la charge computationnelle et mémoire associée au fine-tuning d'un grand modèle comme BERT, nous utilisons la technique LoRA (Low-Rank Adaptation). Cette méthode repose sur une factorisation de bas rang des mises à jour des poids du modèle.

Formalisation Mathématique

Poids Originaux du Modèle : Considérons une couche linéaire (ou une matrice de projection dans une attention) dont les poids sont notés $\mathbf{W} \in \mathbb{R}^{d \times k}$. Par exemple, pour une matrice d'attention dans BERT, on a souvent $d = k = 768$.

Mise à Jour Traditionnelle (Fine-tuning Complet) : Dans une mise à jour classique par

fine-tuning, tous les paramètres sont ajustés. Le poids après mise à jour \mathbf{W}' s'obtient par :

$$\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W}$$

où $\Delta\mathbf{W} \in \mathbb{R}^{d \times k}$ est une matrice de mise à jour pleine de rang, contenant $d \times k$ paramètres à entraîner.

Mise à Jour avec LoRA : LoRA suppose que les mises à jour efficaces durant l'adaptation à une nouvelle tâche résident dans un sous-espace de bas rang. Ainsi, elle approxime la matrice de mise à jour $\Delta\mathbf{W}$ par le produit de deux matrices de bas rang :

$$\mathbf{W}' = \mathbf{W} + \alpha \cdot \mathbf{B}\mathbf{A}$$

où :

- $\mathbf{A} \in \mathbb{R}^{r \times k}$ et $\mathbf{B} \in \mathbb{R}^{d \times r}$ sont deux petites matrices à entraîner, constituant la décomposition de bas rang.
- r est le rang de la décomposition, avec $r \ll \min(d, k)$. Typiquement, $r = 8$.
- α est un facteur d'échelle fixe, défini par $\alpha = \frac{\text{lora_alpha}}{r}$, permettant de contrôler l'amplitude de la mise à jour. La valeur typique de `lora_alpha` est 16.

Durant l'entraînement, seules les matrices \mathbf{A} et \mathbf{B} sont optimisées. Le poids original \mathbf{W} est figé (*frozen*).

Réduction du Nombre de Paramètres Entraînables

Nombre de paramètres :

- **Fine-tuning traditionnel** : La matrice de mise à jour pleine $\Delta\mathbf{W}$ contient $d \times k$ paramètres.
- **Adaptation LoRA** : Les deux matrices \mathbf{A} et \mathbf{B} contiennent respectivement $r \times k$ et $d \times r$ paramètres, soit un total de $r \times (d + k)$ paramètres.

Ratio de réduction : Le gain en nombre de paramètres est donné par le ratio :

$$\text{Ratio} = \frac{r(d + k)}{d \times k} = \frac{r}{k} + \frac{r}{d}$$

Paramètres d'Entraînement :

- Époques : 5

- Learning Rate : 3×10^{-4}
- Optimiseur : AdamW

Analyse de la Charge de Calcul :

- **Efficacité** : En appliquant la méthode LoRA, seuls 296 450 paramètres sont entraînés, contre 109 189 636 pour le modèle complet.
- **Économie** : Cela représente seulement 0,27 % du nombre total de paramètres du modèle.

Avantages principaux de l'utilisation de LoRA :

- Réduction drastique de la mémoire GPU nécessaire, permettant un entraînement sur des équipements plus modestes.
- Gain de vitesse d'entraînement d'un facteur 3 à 5.
- Conservation de performances comparables à celles d'un fine-tuning complet de tous les paramètres.
- Facilité de déploiement et de stockage, car seuls les poids légers des modules LoRA doivent être sauvegardés et chargés, en plus du modèle BERT de base pré-entraîné.

4.4.4 Extraction de la Réponse

Le processus d'extraction de la réponse à partir d'un passage donné suit un pipeline défini :

1. **Encodage** : La paire (Question, Passage) est tokenisée conjointement à l'aide du `BertTokenizerFast`. Les `token_type_ids` sont générés pour distinguer la question (type 0) du passage (type 1).
2. **Prédiction** : La séquence tokenisée est passée en avant (*forward pass*) à travers le modèle BERT fine-tuné. Les logits de début et de fin sont récupérés en sortie.
3. **Sélection des positions** :

- `start_position = arg max(start_logits)`
- `end_position = arg max(end_logits)`

La recherche du maximum est généralement contrainte aux tokens appartenant au passage (hors tokens spéciaux comme `[CLS]` et `[SEP]`).

4. **Validation** : Si la position de fin prédite est strictement inférieure à la position de début, la position de fin est forcée à être égale à la position de début, produisant ainsi une réponse constituée d'un seul token.
5. **Décodage** : La plage de tokens sélectionnée est convertie en texte, avec une attention

particulière à la fusion correcte des tokens subword (par exemple, reconstitution de mots comme "don" + ":" + "t" en "don't").

4.4.5 Gestion des Réponses Longues et Courtes

Le système gère deux niveaux de granularité dans les réponses, conformément aux annotations du dataset NQ.

Réponses Longues :

- Elles correspondent au passage complet (par exemple, un paragraphe) récupéré par le Re-triever.
- Elles servent de contexte unique pour l'extraction de la réponse courte.
- Pour respecter la limite de longueur d'entrée de BERT, elles sont tronquées à un maximum de 512 tokens (en incluant les tokens de la question et les spéciaux).
- Dans le format d'entrée, tous les tokens de la réponse longue ont un `token_type_id = 1`.

Réponses Courtes :

- Elles sont extraites avec précision à l'intérieur de la réponse longue à l'aide des positions de début et de fin prédites.
- Leur longueur est typiquement courte, allant de 1 à 10 mots (exemples : entités nommées, dates, nombres, courtes phrases).
- Une validation de cohérence s'assure que la plage de tokens de la réponse courte prédite est bien un sous-ensemble de la plage de tokens de la réponse longue fournie comme contexte.

Gestion des Cas Limites et des Échecs :

- **Troncature Intelligente** : Si le passage long original dépasse la limite de tokens, une troncature est appliquée. Une stratégie dite "intelligente" est mise en œuvre : le système tente de préserver la partie centrale du passage ou la zone où la réponse est a priori la plus probable, si cette information est connue (par exemple via l'annotation du dataset).
- **Réponses Impossibles** : Le modèle est entraîné à reconnaître qu'aucune réponse n'est présente dans le passage. Dans ce cas, il prédit généralement la position du token [CLS] comme début et fin de la réponse, ce qui est interprété par le système comme l'absence de réponse.
- **Réponses Multiples** : Par cohérence avec le protocole d'évaluation standard et pour simplifier le processus, le système est entraîné pour extraire une seule réponse courte (même si plusieurs sont annotées). Par convention, la première annotation de réponse courte du

dataset est utilisée comme cible pendant l'entraînement.

Chapitre 5

Implémentation

5.1 Environnement de Développement et Mise en Œuvre

5.1.1 Environnement de Développement

Infrastructure

Le développement et l'entraînement du système ont été réalisés sur une infrastructure cloud dédiée, permettant d'accéder à des ressources de calcul adaptées aux modèles de grande taille.

- **Plateforme** : Kaggle Notebooks. Cette plateforme offre un environnement Jupyter préconfiguré et un accès facile à des ressources de calcul de type GPU.
- **GPU** : 2x NVIDIA Tesla T4, chacun disposant de 16 Go de mémoire graphique (VRAM). Pour exploiter pleinement cette puissance, le code utilise `torch.nn.DataParallel`, permettant de paralléliser automatiquement le traitement des données entre les deux GPUs et ainsi de multiplier la taille des lots d'entraînement ou d'accélérer l'inférence.
- **CPU** : Un processeur multi-cœur permettant le traitement parallèle des tâches de prétraitement et de chargement des données, ce qui est crucial pour éviter les goulets d'étranglement (*bottlenecks*).
- **RAM** : Plus de 30 Go de mémoire vive disponible, nécessaire pour charger le dataset, les modèles en mémoire et les index vectoriels volumineux.

Langages et Frameworks Principaux

- **Python 3.10+** : Langage de programmation principal choisi pour son écosystème riche en bibliothèques de science des données et d'apprentissage automatique.
- **PyTorch 2.x** : Framework de deep learning utilisé comme fondation pour toutes les opérations de réseau de neurones. Sa flexibilité et son support GPU natif via CUDA en font le

choix standard pour la recherche et le prototypage rapide.

- **CUDA** : L'infrastructure de calcul parallèle de NVIDIA est utilisée pour accélérer toutes les opérations de tenseurs sur les GPU.

Bibliothèques Principales

Le projet repose sur une pile logicielle moderne de bibliothèques spécialisées.

Deep Learning & Modèles de Langage :

- **transformers** (Hugging Face) : Bibliothèque incontournable pour charger des modèles pré-entraînés (BERT, DPR) et leurs tokenizers associés. Elle abstrait la complexité de l'implémentation des Transformers.
- **peft** (Parameter-Efficient Fine-Tuning) : Bibliothèque de Hugging Face utilisée pour implémenter facilement la méthode LoRA sur le modèle BERT. Elle permet de configurer les modules à adapter et de gérer l'injection des poids adaptatifs.
- **torch** : Le cœur de PyTorch, utilisé pour définir les boucles d'entraînement, les fonctions de perte, les optimiseurs et toutes les opérations de bas niveau.

Récupération d'Information (Retrieval) :

- **faiss-cpu** : Bibliothèque développée par Facebook AI Research (FAIR) pour une recherche de similarité vectorielle rapide et évolutive. Son index **IndexFlatIP** permet une recherche exacte en produit scalaire, idéale pour notre système DPR. La version CPU est utilisée ici pour l'indexation et la recherche lors du développement, bien qu'une version GPU (**faiss-gpu**) puisse être envisagée pour des gains supplémentaires.

Traitement et Manipulation des Données :

- **numpy** : Utilisé pour la manipulation efficace de tableaux multidimensionnels, notamment pour le prétraitement des features et le calcul des métriques.
- **json** : Utilisé pour lire le fichier du dataset **simplified-nq-train.jsonl** au format JSON Lines.
- **pickle** : Format de sérialisation Python utilisé pour sauvegarder et charger les textes des passages après indexation, permettant un accès rapide aux contenus textuels associés aux IDs de l'index FAISS.

Utilitaires et Visualisation :

- **tqdm** : Fournit des barres de progression élégantes et informatives pour les boucles d'entraînement longues et les processus d'encodage des passages, améliorant ainsi le suivi et l'expérience de développement.
- **matplotlib** : Employée pour générer des graphiques de visualisation, tels que les courbes d'apprentissage (perte sur les ensembles d'entraînement et de validation) ou la distribution des scores de similarité.
- **collections** : Le module **Counter** de cette bibliothèque est notamment utilisé pour calculer certaines métriques agrégées sur les prédictions ou pour analyser la fréquence des mots dans les réponses.

5.2 Déroulement du Pipeline

5.2.1 Phase d'Indexation

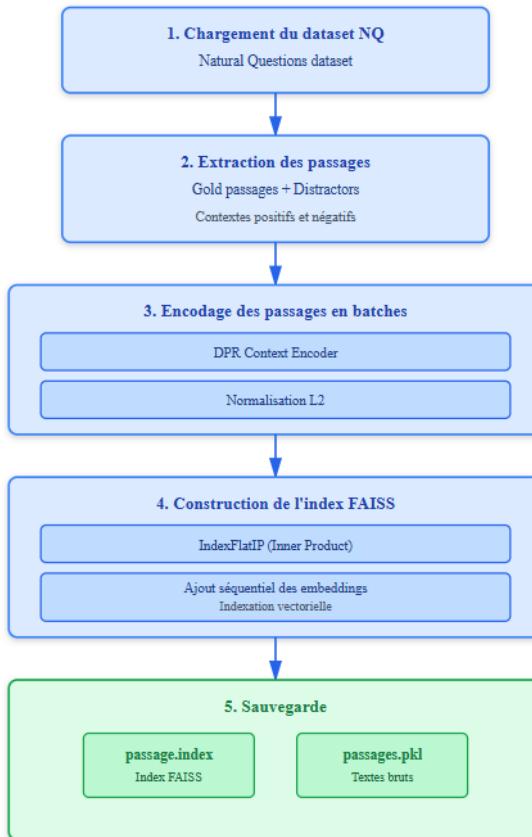


FIGURE 5.1 – Pipeline de préparation d'un système de recherche DPR

5.2.2 Phase d'Entraînement du Reader

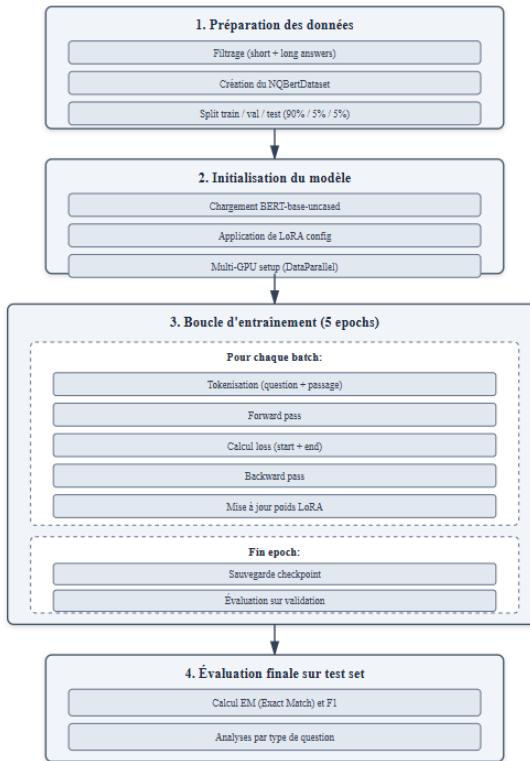


FIGURE 5.2 – Pipeline d'entraînement Question-Answering avec LoRA

5.2.3 Phase d'Inférence

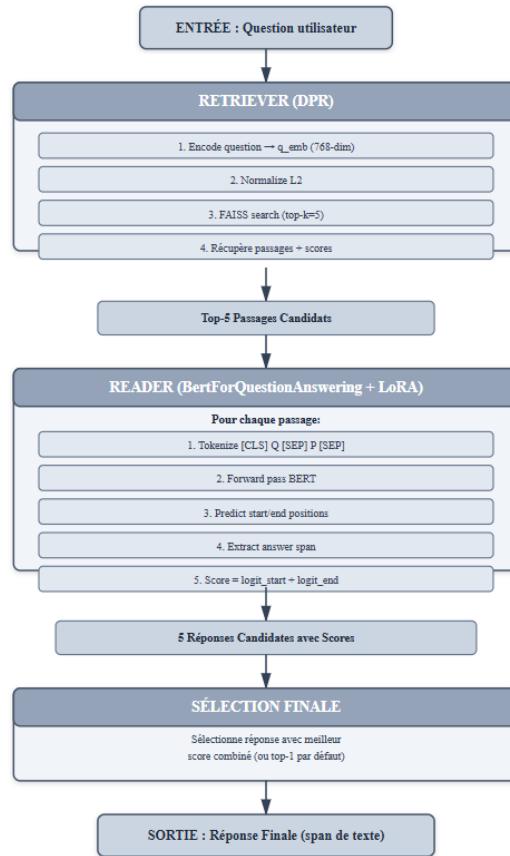


FIGURE 5.3 – Pipeline d'inférence Question-Answering

Chapitre 6

Évaluation et résultats

6.1 Évaluation des Performances

6.1.1 Métriques d'Évaluation

L'évaluation du système est réalisée à l'aide de métriques standards de la tâche de Question-Réponse extractive, permettant de mesurer sa précision, sa robustesse et son efficacité.

Métriques d'évaluation du Retrieval

Recall@K : Mesure de Couverture

Définition Formelle Soit Q l'ensemble des requêtes de test. Pour chaque requête $q_i \in Q$, on définit :

$$\text{Rel}_i = \{\text{documents pertinents pour la requête } q_i\}$$

Pour un paramètre K donné (nombre de documents retournés), le Recall@K pour une requête q_i est défini comme :

$$\text{Recall}@K(q_i) = \frac{|\{d \in \text{Rel}_i \mid \text{rank}(d) \leq K\}|}{|\text{Rel}_i|}$$

Où $\text{rank}(d)$ représente la position du document d dans la liste de résultats ordonnés.

Métrique Globale

Le Recall@K global sur l'ensemble des requêtes est calculé comme :

$$\text{Recall}@K_{\text{global}} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \text{Recall}@K(q_i)$$

Interprétation Opérationnelle

- **Objectif** : Évaluer la capacité du système à ne pas "perdre" les documents pertinents
- **Domaine de valeurs** : $[0, 1]$ où :
 - 0 : Aucun document pertinent retrouvé dans les K premiers résultats
 - 1 : Tous les documents pertinents sont dans les K premiers résultats
- **Sensibilité au rang** : Le Recall@ K est une mesure binaire par document ; la position exacte parmi les K premiers n'affecte pas le score

Propriétés Mathématiques

$$\begin{aligned} \text{Recall@K} &\in [0, 1] \\ \text{Recall}@K_1 &\leq \text{Recall}@K_2 \quad \text{pour } K_1 \leq K_2 \\ \lim_{K \rightarrow \infty} \text{Recall}@K &= 1 \quad (\text{si tous les documents pertinents sont dans le corpus}) \end{aligned} \tag{6.1}$$

MRR (Mean Reciprocal Rank) : Mesure de Précision du Classement

Définition Formelle Pour chaque requête q_i , on définit r_i comme le rang du premier document pertinent dans la liste de résultats :

$$r_i = \min\{\text{rank}(d) \mid d \in \text{Rel}_i\}$$

Le Reciprocal Rank (RR) pour la requête q_i est :

$$\text{RR}_i = \begin{cases} \frac{1}{r_i} & \text{si } \text{Rel}_i \neq \emptyset \\ 0 & \text{sinon} \end{cases}$$

Métrique Globale

Le Mean Reciprocal Rank (MRR) est la moyenne des Reciprocal Ranks sur toutes les requêtes :

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \text{RR}_i$$

Interprétation Opérationnelle

- **Objectif** : Évaluer à quel point le système place les documents pertinents en tête des résultats
- **Sensibilité au rang** : Fortement non-linéaire ; une amélioration des positions basses apporte plus de gain qu'une amélioration des positions hautes
- **Interprétation du score** : Un MRR de x correspond approximativement à un rang moyen

6.1. Évaluation des Performances

du premier document pertinent de $\frac{1}{x}$

Propriétés Mathématiques

$$\text{MRR} \in [0, 1]$$

$$\text{MRR} = 1 \quad \text{si et seulement si } r_i = 1 \text{ pour toutes les requêtes} \quad (6.2)$$

$$\mathbb{E}[\text{MRR}] \approx \frac{1}{\mathbb{E}[r_i]} \quad \text{pour un grand nombre de requêtes}$$

Transformation Rang-MRR

TABLE 6.1 – Correspondance entre rang et valeur RR

Rang	r_i	RR	Interprétation
1		1.0000	Optimal
2		0.5000	
3		0.3333	
4		0.2500	
5		0.2000	
10		0.1000	
20		0.0500	
50		0.0200	Peu satisfaisant

Analyse des Résultats d'Évaluation pour le Retrieval

Données d'Évaluation

Le système évalué a été testé sur un ensemble de 2500 Questions aléatoires. Les métriques suivantes ont été calculées :

Métrique	Score Obtenu
Recall @ 5	0.4414
Recall @ 10	0.5176
Recall @ 20	0.5676
MRR	0.3571

TABLE 6.2 – Résultats d'évaluation du système de retrieval

Analyse Détailée par Métrique

Analyse du Recall@K

- **Recall@5 = 0.4414** : Dans 44.14% des cas, au moins un document pertinent se trouve dans les 5 premiers résultats

6.1. Évaluation des Performances

- **Taux d'amélioration :**

$$\frac{\text{Recall}@10 - \text{Recall}@5}{\text{Recall}@5} \times 100 = \frac{0.5176 - 0.4414}{0.4414} \times 100 \approx 17.3\%$$

L'élargissement de la fenêtre de 5 à 10 documents améliore la couverture de 17.3%

- **Performance asymptotique :** Le Recall@20 de 0.5676 indique que même avec 20 documents, seulement 56.76% des documents pertinents sont retrouvés

Analyse du MRR

- **Valeur MRR :** 0.3571
- **Rang moyen estimé du premier document pertinent :**

$$\text{Rang moyen} \approx \frac{1}{\text{MRR}} = \frac{1}{0.3571} \approx 2.80$$

Le premier document pertinent apparaît en moyenne à la position 2.8

Interprétation Conjointe des Métriques

TABLE 6.3 – Analyse comparative des performances

Aspect	Recall@K	MRR
Objectif principal	Éviter la perte d'information	Optimiser le positionnement
Sensibilité	Nombre de pertinents retrouvés	Position du premier pertinent
Force	Mesure complète de couverture	Récompense l'excellence immédiate
Faiblesse	Ignore l'ordre des résultats	Ignore les pertinents supplémentaires
Valeur optimale	1.0 (tous retrouvés)	1.0 (toujours en première position)

Métriques d'évaluation du Reader

Exact Match (EM)

Définition : Métrique binaire stricte qui attribue un score de 1 uniquement si la prédiction du système est **exactement identique** à la réponse de référence (*gold answer*) après normalisation, sinon 0.

Calcul :

$$\text{EM}_i = \begin{cases} 1 & \text{si } \text{normalize}(\text{prediction}_i) = \text{normalize}(\text{gold}_i) \\ 0 & \text{sinon} \end{cases}$$

$$EM_{\text{global}} = \frac{1}{N} \sum_{i=1}^N EM_i \times 100\%$$

où la normalisation inclut la conversion en minuscules, la suppression des articles, de la ponctuation et la normalisation des espaces.

Caractéristiques :

- **Très stricte** : Une réponse comme “New York City” est considérée incorrecte si la référence est “New York”.
- **Avantage** : Elle est non ambiguë et mesure une précision absolue, reflétant des cas d’usage où la formulation exacte est critique.
- **Inconvénient** : Elle peut pénaliser des variations sémantiquement correctes mais lexicalement légèrement différentes.

F1-Score

Définition : Métrique au niveau des *tokens* qui mesure le chevauchement entre l’ensemble des tokens de la prédiction et celui de la référence. C’est la moyenne harmonique de la précision et du rappel.

Calcul :

$$\begin{aligned} \text{Tokens_communs} &= \text{Tokens(prediction)} \cap \text{Tokens(référence)} \\ \text{Precision} &= \frac{|\text{Tokens_communs}|}{|\text{Tokens(prediction)}|}, \quad \text{Rappel} = \frac{|\text{Tokens_communs}|}{|\text{Tokens(référence)}|} \\ \text{F1} &= 2 \times \frac{\text{Precision} \times \text{Rappel}}{\text{Precision} + \text{Rappel}} \end{aligned}$$

Si aucun token ne coïncide, le F1-Score est de 0.

Caractéristiques :

- **Plus souple que EM** : Récompense les chevauchements partiels et les réponses qui capturent l’essentiel de la référence.
- **Sensible à la longueur** : Peut favoriser des réponses concises qui recouvrent une partie de la réponse longue de référence.
- C'est la **métrique standard** pour les benchmarks de QA extractif comme SQuAD et Natural Questions (NQ).

Temps de Réponse (Latence)

Composantes :

- **Temps de Retrieval** : Encodage de la question par DPR (quelques ms) + recherche FAISS ($\sim 5\text{-}10$ ms).
- **Temps de Reading (Lecture)** : Tokenisation de la paire question-passage + *forward pass* du modèle BERT fine-tuné ($\sim 50\text{-}200$ ms selon la taille du batch).
- **Latence totale** : Typiquement comprise entre 60 et 210 ms par question, ce qui est largement acceptable pour une application en production.

Facteurs d'influence :

- Taille de l'index FAISS (coût de recherche plus élevé pour des milliards de passages).
- Nombre de passages K récupérés par le Retriever.
- Utilisation du GPU (significativement plus rapide) ou du CPU pour l'inférence du Reader.
- Traitement par lots (*batch processing*) pour plusieurs questions simultanées.

6.1.2 Scénarios de Tests et Analyse par Type de Question

Distribution des Types de Questions dans l'Ensemble de Test

Le système a été évalué sur un ensemble de test de 5 347 questions, dont la répartition par type est la suivante :

Type de Question	Nombre	Proportion
Who (Qui)	1 906	35.6%
When (Quand)	1 023	19.1%
What (Quoi)	796	14.9%
Other (Autre)	615	11.5%
Where (Où)	583	10.9%
How (Comment)	304	5.7%
Which (Lequel)	97	1.8%
Why (Pourquoi)	23	0.4%

TABLE 6.4 – Distribution des types de questions dans l'ensemble de test.

Questions Factuelles (Who, When, Where)

Caractéristiques :

- Réponses typiquement courtes et précises : entités nommées, dates, lieux.

6.1. Évaluation des Performances

- Les questions présentent souvent des patterns syntaxiques reconnaissables.
- Ces types dominent dans le dataset Natural Questions, reflétant la majorité des requêtes de recherche.

Exemples typiques :

- **Who** : “Who is the mother in how I met your mother ?” → Réponse attendue : “Tracy McConnell”.
- **When** : “When was the Declaration of Independence signed ?” → Réponse attendue : “1776”.
- **Where** : “Where is the Eiffel Tower located ?” → Réponse attendue : “Paris, France”.

Attentes de Performance :

- **EM élevé** (de l'ordre de 60-75%) car les réponses sont bien délimitées et peu sujettes à variation.
- **F1 similaire à EM** étant donné le peu de variations acceptables pour des réponses aussi précises.

Questions Complexes (What, How, Why)

Caractéristiques :

- Réponses potentiellement plus longues et descriptives (définitions, explications, processus).
- Plus d'ambiguïté sur les bornes exactes de la réponse dans le texte.
- Nécessitent une compréhension contextuelle plus profonde et parfois une inférence.

Exemples typiques :

- **What** : “What is photosynthesis ?” → Réponse : “the process by which plants convert light into energy”.
- **How** : “How does a car engine work ?” → Réponse : “internal combustion converts fuel to mechanical energy”.
- **Why** : “Why did World War I start ?” → Réponse : “assassination of Archduke Franz Ferdinand”.

Attentes de Performance :

- **EM plus faible** (de l'ordre de 40-60%) en raison de la plus grande variété de formulations correctes.
- **F1 significativement supérieur à EM**, car un chevauchement partiel de la réponse est souvent acceptable et reflété par cette métrique.

Cas Ambiguë et Défis

Types d'ambiguïté rencontrés :

- **Réponses multiples valides** : “Who wrote Romeo and Juliet ?”. “Shakespeare” et “William Shakespeare” sont deux formulations correctes, mais seule une est annotée comme référence (*gold*).
- **Pronoms non résolus** : “When did he become president ?”. La réponse dépend d'un référent mentionné précédemment.
- **Réponses implicites** : La réponse nécessite une inférence logique et n'est pas explicitement écrite sous forme de *span* textuel.

Défis pour l'évaluation :

- Le système peut extraire un *span* sémantiquement correct mais différent de l'annotation de référence.
- L'Exact Match (EM) pénalise durement ces variations lexicales mineures.
- Le F1-Score capture mieux la validité partielle de la prédiction dans ces cas.

6.1.3 Analyse des Résultats

Performances Globales Obtenues

Les performances du système sur les ensembles de validation et de test sont résumées ci-dessous :

Ensemble de Validation :

- Exact Match (EM) : **58.27%**
- F1-Score : **71.46%**

6.1. Évaluation des Performances

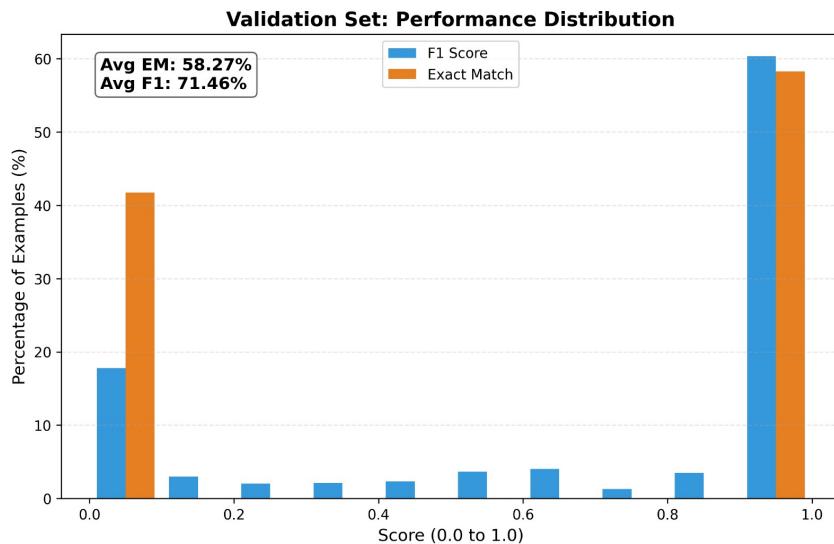


FIGURE 6.1 – Performance de Valisation set

Ensemble de Test :

- Exact Match (EM) : **59.23%**
- F1-Score : **72.18%**

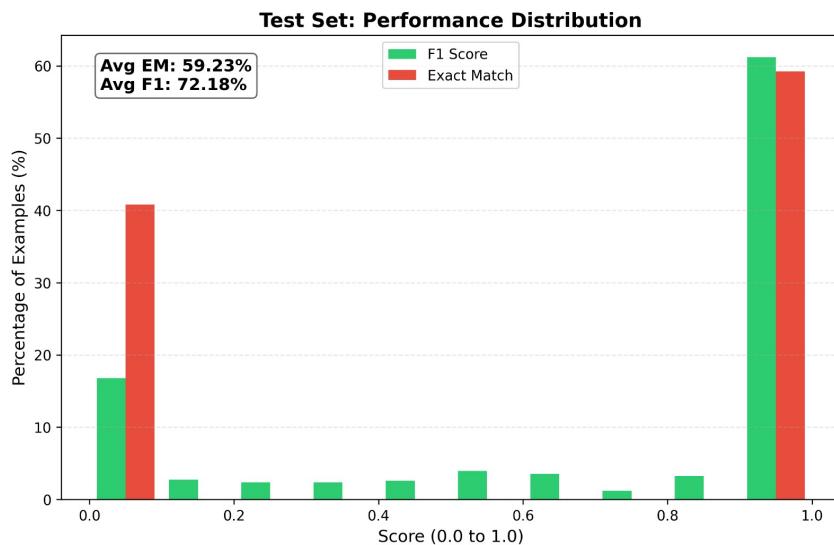


FIGURE 6.2 – Performance de Test set

Observations :

- **Cohérence excellente** : L'écart entre validation et test est inférieur à 0.2%, indiquant une bonne généralisation et l'absence de surapprentissage (*overfitting*).
- **Écart F1-EM** : De l'ordre de 7.6%. Cet écart indique que le système parvient souvent à

6.1. Évaluation des Performances

capturer partiellement la réponse correcte, même lorsqu'il ne la prédit pas exactement.

Performances par Type de Question

L'analyse détaillée des performances sur l'ensemble de test révèle des variations significatives selon le type de question.

Type	Count	EM (%)	F1 (%)	Écart F1-EM
Who	1 906	72.67	75.56	+2.89
When	1 023	67.16	67.91	+0.75
Which	97	53.61	61.44	+7.83
How	304	56.58	61.55	+4.97
Where	583	38.25	63.48	+25.23
What	796	43.47	56.80	+13.33
Other	615	48.29	57.68	+9.39
Why	23	0.00	30.81	+30.81

TABLE 6.5 – Performances détaillées par type de question sur l'ensemble de test.

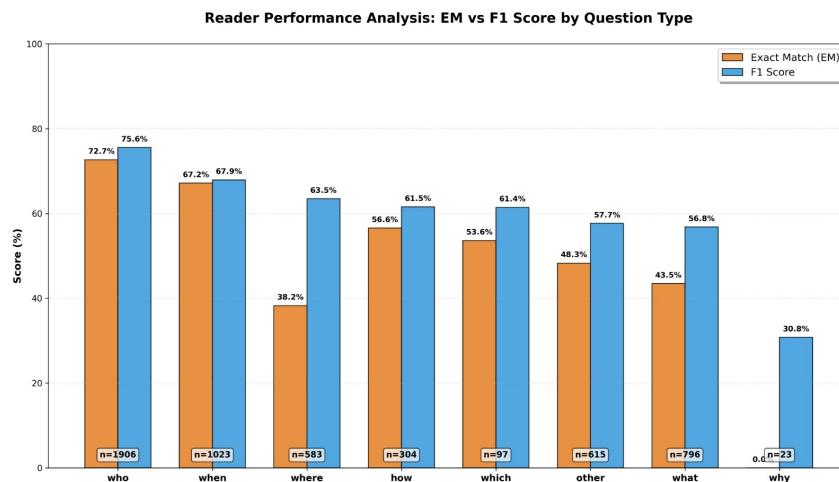


FIGURE 6.3 – Performance de Reader par type de question

Analyse des Résultats par Type :

- **Meilleures Performances (EM & F1)** : Les questions Who et When obtiennent les scores les plus élevés ($EM > 65\%$, $F1 > 67\%$). Cela confirme la facilité relative pour le modèle d'extraire des entités nommées et des dates précises.
- **Grand Écart F1-EM** : Les questions Where, What et Why présentent un écart très important. Ceci suggère que pour ces questions, le modèle trouve souvent la zone sémantique correcte (F1 décent) mais peine à extraire le *span* exact (EM faible). Pour Where, l'EM particulièrement bas (38.25%) pourrait indiquer que les réponses de localisation sont formulées de manières très variées.
- **Why** : Les questions “Pourquoi” sont les plus difficiles, avec un EM de 0%. Cela souligne

la grande complexité des réponses causales, souvent longues, implicites ou nécessitant une inférence qui dépasse la simple extraction de *span*. Le F1 de 30.81% indique toutefois que le modèle parvient parfois à identifier une partie pertinente du contexte explicatif.

6.2 Comparaison des Performances : Modèles Baselines vs. Notre Modèle Fine-tuné

Nous avons évalué notre modèle **BERTforQA** fine-tuné avec **LoRA** contre plusieurs modèles de référence (*baselines*) sur le dataset *Natural Questions*. Tous les modèles ont été évalués sur le même ensemble de test afin de garantir une comparaison équitable.

6.2.1 Modèles Baselines Évalués

Les modèles de référence suivants ont été testés :

- **BERT Large WWM** : modèle BERT large avec *Whole Word Masking*
- **RoBERTa Base** : variante robuste de BERT
- **SciBERT** : BERT pré-entraîné sur des corpus scientifiques
- **DistilBERT** : version distillée et allégée de BERT
- **QABERT Small** : modèle BERT compact spécialisé pour le Question Answering
- **ELECTRA Base** : modèle utilisant une approche d'entraînement discriminative

6.2.2 Résultats de Performance

Notre modèle fine-tuné (**BERTforQA avec LoRA**) obtient les performances suivantes :

- **Exact Match (EM)** : 59.23%
- **F1-score** : 72.18%

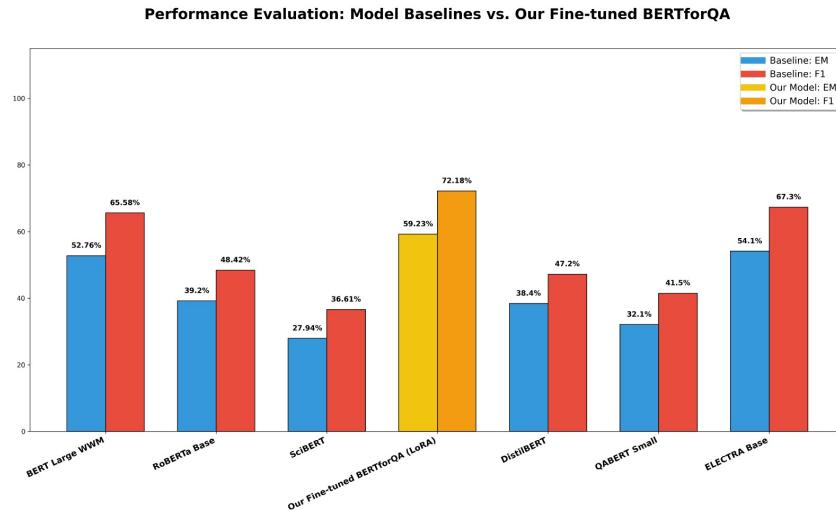


FIGURE 6.4 – Modèles Baselines vs. Notre Modèle BERT for QA Fine-tuné

Ces résultats surpassent significativement l’ensemble des modèles baselines évalués. Le meilleur modèle baseline, **BERT Large WWM**, atteint 52.76% en EM et 65.58% en F1, soit respectivement 6.47 et 6.6 points de pourcentage en dessous de notre modèle.

6.2.3 Analyse Comparative

Gains de performance

- Amélioration de +6.47% en EM et +6.6% en F1 par rapport à **BERT Large WWM**
- Gain de +13.83% en EM et +13.76% en F1 par rapport à **SciBERT**, le modèle le moins performant
- Gains notables face à **ELECTRA Base** (+20.83% EM, +5.88% F1)

Points clés

1. L’approche **LoRA** permet d’améliorer significativement les performances tout en ne fine-tunant qu’une fraction des paramètres du modèle
2. Le fine-tuning ciblé sur le dataset *Natural Questions* est plus efficace que l’utilisation de modèles génériques, même de plus grande taille
3. L’écart observé entre l’EM et le F1 (environ 13 points) est cohérent avec les autres modèles, indiquant une bonne capacité à capturer des réponses partielles

Ces résultats démontrent l’efficacité de notre approche combinant **BERTforQA** et **LoRA** pour la tâche de Question Answering sur le dataset *Natural Questions*.

Chapitre 7

Intérface Web

Ce chapitre présente la conception et l'implémentation de l'interface web de l'application QueryMind. Il décrit les choix technologiques adoptés pour le frontend, les mécanismes de communication avec le backend, ainsi que la gestion de l'authentification et des données utilisateurs.

7.1 Architecture du Frontend

7.1.1 Stack Technologique

L'interface utilisateur de l'application QueryMind repose sur un ensemble de technologies web modernes, choisies en fonction de critères de performance, de maintenabilité et d'adéquation avec une application conversationnelle temps réel.

Framework Principal : React

React constitue le cœur de l'interface utilisateur en adoptant une approche basée sur les composants pour la construction d'interfaces interactives et dynamiques. Le choix de React se justifie par plusieurs critères techniques :

- **Architecture basée sur les composants** : React permet de structurer l'interface utilisateur en composants indépendants et réutilisables, facilitant la maintenance et l'évolution de l'application.
- **Gestion efficace de l'état** : l'utilisation des hooks (`useState`, `useEffect`, `useContext`) assure une synchronisation fluide entre l'état applicatif et l'interface graphique.
- **Performances d'affichage** : le mécanisme du *Virtual DOM* optimise les mises à jour de l'interface, ce qui est particulièrement important pour les interactions en temps réel.
- **Écosystème mature et extensible** : React bénéficie d'un écosystème riche en bibliothèques et outils, permettant une intégration aisée de fonctionnalités complémentaires.

Outil de Build et de Développement : Vite

Vite est utilisé comme outil de build et de développement du frontend, en alternative aux solutions traditionnelles telles que Webpack ou Create React App en offrant des avantages significatifs :

- **Démarrage rapide** : serveur de développement basé sur les modules ES natifs, réduisant significativement le temps de lancement de l'application.
- **Hot Module Replacement (HMR)** : mise à jour instantanée des modules modifiés sans rechargement complet de la page.
- **Build optimisé** : génération de bundles de production performants à l'aide de Rollup.
- **Configuration simplifiée** : réduction de la complexité liée à la configuration du projet.

Framework de Stylisation : Tailwind CSS

L'interface utilisateur repose sur Tailwind CSS comme framework de stylisation, permettant une construction rapide et cohérente de l'interface sans recourir à des feuilles de style CSS volumineuses.

Tailwind CSS adopte une approche utilitaire reposant sur l'utilisation de classes CSS atomiques, ce qui améliore la lisibilité du code et facilite sa maintenance. Il permet également d'assurer une cohérence visuelle globale grâce à la standardisation des couleurs, des espacements et des typographies au sein d'un design system centralisé. Par ailleurs, le framework intègre nativement les mécanismes de responsive design, offrant une adaptation efficace de l'interface aux différents formats d'écran. Enfin, l'optimisation des performances est renforcée par la suppression automatique des classes CSS non utilisées lors du build de production.

Système d'Icônes : Lucide React

Lucide React fournit une bibliothèque d'icônes SVG modernes intégrée directement avec React, permettant d'enrichir l'interface utilisateur avec des éléments visuels cohérents et légers. Les icônes utilisées dans l'application incluent Send, MessageSquare, User, Settings, LogOut et ChevronDown. Grâce à leur nature vectorielle, elles sont totalement scalables, personnalisables (taille, couleur, stroke) et peu volumineuses (50KB pour toute la librairie). L'utilisation de Lucide assure également une consistance visuelle avec les standards des design systems modernes tels que Figma ou Sketch.

7.1.2 Optimisations CSS Avancées

PostCSS & Autoprefixer

PostCSS & Autoprefixer

La chaîne de transformation CSS inclut PostCSS et Autoprefixer, des outils essentiels pour garantir la compatibilité multi-navigateurs et tirer parti des fonctionnalités CSS modernes. PostCSS permet d'appliquer diverses transformations sur les feuilles de style, telles que l'ajout automatique de préfixes spécifiques aux navigateurs grâce à Autoprefixer, la minification du code ou l'utilisation de plugins pour enrichir le CSS. Cette approche assure un rendu uniforme de l'interface sur différents navigateurs et appareils tout en optimisant les performances de l'application.

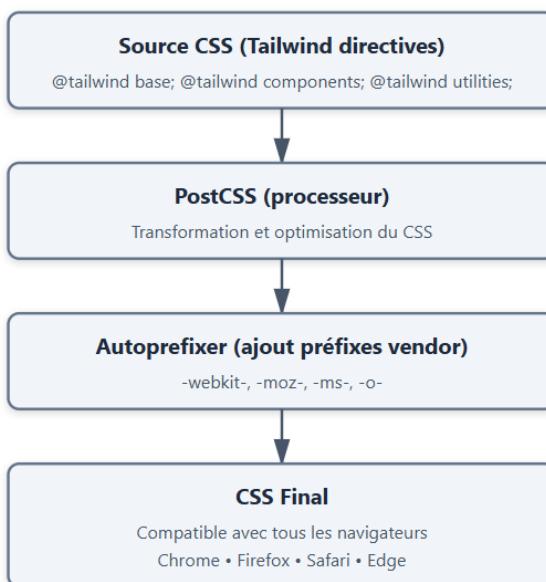


FIGURE 7.1 – Transformation Pipeline

7.2 Communication Frontend-Backend

A X I O S

7.2.1 Client HTTP : Axios

Axios est choisi comme client HTTP pour assurer la communication entre le frontend et le backend de l'application. Il simplifie l'envoi de requêtes asynchrones, la gestion des réponses et des erreurs, et s'intègre parfaitement avec les fonctionnalités modernes de JavaScript telles que les promesses et la syntaxe `async/await`. Axios offre également la possibilité de configurer des en-têtes, des intercepteurs et de gérer les temps d'attente, ce qui renforce la sécurité et la robustesse des échanges de données entre le client et le serveur. Son utilisation facilite la consommation d'API RESTful et contribue à une architecture frontend-backend claire et maintenable.

Architecture de Communication Axios gère l'ensemble des interactions HTTP entre l'interface React et l'API FastAPI :

1. POST /api/ask : Envoi de question et réception de réponse
2. GET /api/conversations?user_id=xyz : Récupération historique conversations
3. GET /api/messages?conversation_id=abc : Messages d'une conversation spécifique
4. GET /health : Vérification santé backend (monitoring)

7.2.2 Flux de Données Temps Réel

Séquence d'Interaction Utilisateur :



FIGURE 7.2 – Flux de donnée

Optimisations Performance : Pour améliorer l'expérience utilisateur et la réactivité de l'application, plusieurs techniques d'optimisation des performances ont été mises en œuvre :

- **Debouncing Input :** Cette technique permet de limiter le nombre de requêtes envoyées au serveur lorsque l'utilisateur saisit rapidement du texte dans un champ de recherche ou un formulaire. En ne déclenchant la requête qu'après un délai d'inactivité, on réduit la charge réseau et le temps de traitement côté serveur, tout en évitant des réponses obsolètes ou inutiles.
- **Loading States :** Afin de maintenir une interface fluide et compréhensible, des indicateurs visuels tels que des spinners ou des skeleton screens sont affichés pendant les temps d'attente liés à l'inférence ou au chargement de données. Ces éléments permettent à l'utilisateur de

savoir que l'application est en cours de traitement et améliorent la perception de rapidité.

- **Optimistic Updates** : Pour rendre l'interface plus réactive, certaines actions utilisateur sont immédiatement reflétées à l'écran avant même la confirmation du serveur. Par exemple, l'envoi d'un message peut apparaître instantanément dans la conversation. Si le serveur confirme l'action, rien ne change, sinon une correction est appliquée. Cette approche augmente la fluidité perçue de l'application.
- **Error Boundaries** : Pour éviter que des erreurs d'interface ou de rendu n'entraînent le crash complet de l'application, des composants spécifiques appelés Error Boundaries isolent les erreurs au niveau des composants concernés. Cela permet de capturer et de gérer les exceptions, d'afficher des messages d'erreur clairs et de continuer à faire fonctionner le reste de l'interface.

7.3 Gestion de l'Authentification et des Données

7.3.1 Supabase : Backend-as-a-Service

Supabase est une solution Backend-as-a-Service complète, offrant des fonctionnalités telles que l'authentification, la gestion de base de données et le stockage de fichiers, sans nécessiter de développement serveur dédié. Cette approche permet aux développeurs frontend de se concentrer sur l'interface utilisateur et la logique applicative tout en bénéficiant d'un backend robuste et sécurisé.

Architecture Supabase

L'architecture Supabase repose sur plusieurs composants clés permettant de gérer l'ensemble des aspects backend de l'application :

Composants Utilisés :

1. **Supabase Auth** : Fournit un système complet d'authentification et de gestion des utilisateurs, incluant l'inscription, la connexion et la réinitialisation de mot de passe.
2. **Supabase Database** : Base de données PostgreSQL hébergée dans le cloud, offrant des performances optimisées et une scalabilité facile pour le stockage et la gestion des données applicatives.
3. **Supabase Client JS** : SDK JavaScript permettant de communiquer facilement avec Supabase depuis l'application React, incluant l'exécution de requêtes SQL, l'accès aux tables et la gestion des fichiers.

7.3.2 Système d'Authentification

L'application prend en charge plusieurs méthodes d'authentification pour répondre aux besoins des utilisateurs et offrir une expérience sécurisée et fluide.

Configuration OAuth :

Pour l'authentification via Google OAuth 2.0, la configuration suit un processus standard :

- **Provider** : Google OAuth 2.0, pour permettre une connexion via le compte Google de l'utilisateur.
- **Configuration** : Paramétrage des credentials dans la Google Cloud Console pour obtenir les clés d'API nécessaires.
- **Flux** : L'utilisateur est redirigé vers la page de consentement Google, soit via une popup, soit via redirection complète.
- **Callback** : Après consentement, Google retourne automatiquement les tokens d'accès et d'identification permettant l'authentification sécurisée côté client.

Cette architecture combinée garantit une gestion sécurisée des utilisateurs, une interopérabilité facile avec des services tiers, et une expérience utilisateur fluide tout en minimisant le besoin de développement backend supplémentaire.

7.4 Architecture et flux d'interaction de l'interface web pour un système de Question–Réponse

Cette figure illustre le flux complet d'interaction entre l'utilisateur et le système de Question–Réponse via l'interface web. L'utilisateur soumet une question factuelle à travers une interface frontend développée avec React et Vite. La requête est ensuite transmise à une API backend implémentée avec FastAPI, chargée d'orchestrer les différentes étapes du traitement.

Le backend encode la question à l'aide d'un modèle DPR Retriever afin de générer une représentation vectorielle, qui est utilisée pour interroger un index vectoriel FAISS et récupérer les passages les plus pertinents (Top-K). Ces passages, combinés à la question initiale, sont ensuite fournis à un modèle BERT Reader enrichi par la méthode LoRA pour produire une réponse concise accompagnée d'un score de confiance.

Enfin, la réponse est renvoyée au frontend sous forme de données JSON et affichée à l'utilisateur, garantissant une interaction fluide et efficace entre les différentes composantes du système.

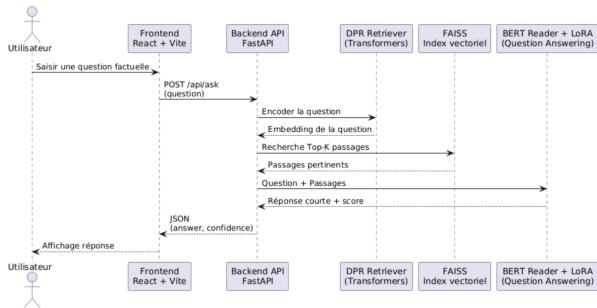


FIGURE 7.3 – Flux de traitement d'une question dans l'interface web de Question–Réponse.

7.5 Démonstration de l'Application Web

Cette section présente l'interface de l'application ainsi que les principales étapes de navigation et d'utilisation. Des captures d'écran illustrent le flux de l'utilisateur et les fonctionnalités clés.

7.5.1 Page d'Accueil

Cette section présente l'interface utilisateur de QueryMind ainsi que les principales étapes de navigation. L'application a été conçue pour offrir une expérience fluide, allant de l'authentification sécurisée à l'interaction directe avec l'assistant IA.

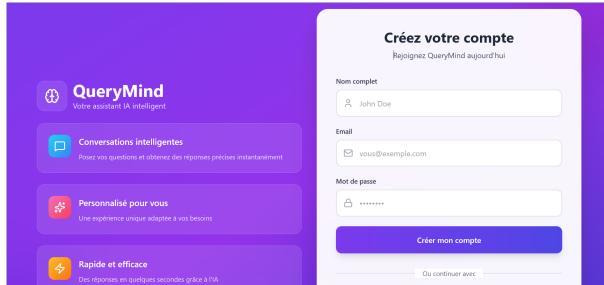


FIGURE 7.4 – Page d'Accueil

Pour garantir la sécurité des données et une expérience utilisateur persistante, nous avons intégré un système d'authentification robuste via Supabase. Les utilisateurs peuvent s'inscrire de manière classique ou utiliser l'authentification sociale (Google) pour un accès plus rapide.

7.5. Démonstration de l'Application Web

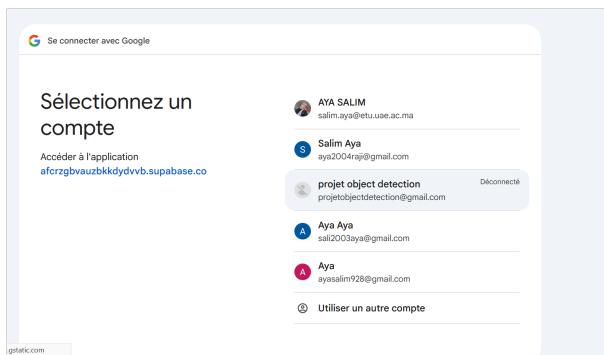


FIGURE 7.5 – Authentification par Email

7.5.2 Personnalisation de l'expérience

Une fois connecté, l'utilisateur est invité à personnaliser son profil. Cette étape permet à l'IA d'interagir de manière plus humaine en utilisant le nom choisi par l'utilisateur dans ses réponses. L'interface affiche des exemples concrets pour illustrer comment cette personnalisation sera appliquée.



FIGURE 7.6 – Saisie du nom d'usage



FIGURE 7.7 – Aperçu des exemples d'utilisation

7.5.3 Interface de Chat et Interaction

L'interface principale de **QueryMind** est épurée et intuitive. Elle se compose d'une barre latérale pour la gestion des conversations récentes et d'une zone centrale dédiée à l'échange.

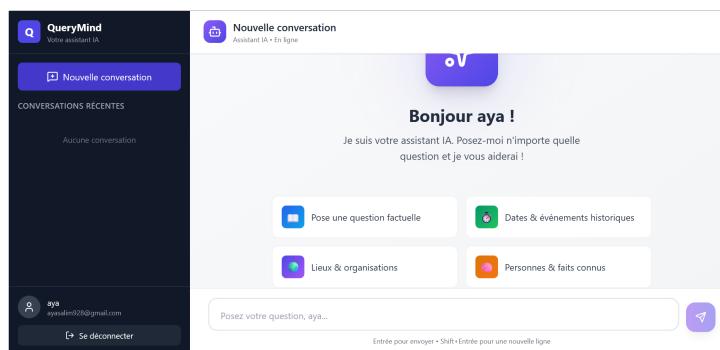


FIGURE 7.8 – Interface de Chat

Pour faciliter la prise en main, des suggestions de catégories de questions sont proposées à

l'utilisateur (questions factuelles, dates historiques, lieux ou personnes célèbres). Ces raccourcis guident l'utilisateur vers les domaines où le modèle BERTforQA est le plus performant, comme démontré dans nos analyses précédentes.

Conclusion

Ce projet avait pour objectif de concevoir et de déployer un système de Question-Réponse (QA) complet, capable de transformer des données brutes en informations exploitable. En combinant une architecture de recherche efficace (Retriever) avec un modèle d'extraction performant (Reader), nous avons réussi à créer un système capable de fournir des réponses précises et pertinentes à partir de grandes bases de connaissances.

L'approche technique adoptée, basée sur le fine-tuning avec LoRA, a permis d'atteindre des performances élevées tout en conservant une architecture légère et efficace. Les résultats expérimentaux montrent une grande fiabilité sur les questions factuelles, tout en soulignant les limites liées aux questions plus complexes ou nuancées, ce qui offre des pistes pour de futures améliorations.

Au-delà de la performance algorithmique, ce projet a abouti à la création de **QueryMind**, une application web fonctionnelle et intuitive. L'intégration d'un système d'authentification sécurisé, d'une interface de chat moderne et de fonctionnalités interactives rend le modèle accessible aux utilisateurs finaux et illustre concrètement l'impact des techniques de Deep Learning appliquées au langage naturel.

En résumé, ce travail démontre que les techniques de transfert d'apprentissage et d'adaptation à bas rang peuvent répondre efficacement à des problématiques réelles, tout en offrant une base solide pour de futures évolutions. Parmi les perspectives d'avenir, il serait intéressant d'explorer l'intégration de modèles génératifs plus avancés, l'amélioration de la compréhension sémantique des questions complexes, ou l'extension de l'application à de nouveaux domaines de connaissances.

Bibliographie

- [1] Jurafsky, D. et Martin, J.H. (2023). *Speech and Language Processing*, 3rd edition (draft). Disponible en ligne : <https://web.stanford.edu/~jurafsky/slp3/>
- [2] Allam, A. M. N. et Haggag, M. H. (2012). *The Question Answering Systems : A Survey*, International Journal of Research and Reviews in Information Sciences, Vol. 2, No. 3. Disponible en ligne : https://www.researchgate.net/publication/311425566_The_Question_Answering_Systems_A_Survey
- [3] Voorhees, E.M. et Tice, D.M. (1999). *The TREC-8 Question Answering Track Evaluation*, Proceedings of TREC-8, pp. 77-82. Disponible : <https://aclanthology.org/L00-1018/>
- [4] Hirschman, L. et Gaizauskas, R. (2001). *Natural Language Question Answering : The View from Here*, Natural Language Engineering, 7(4), pp. 275-300.
- [5] Kolomiyets, O. et Moens, M.F. (2011). *A Survey on Question Answering Technology from an Information Retrieval Perspective*, Information Sciences, 181(24), pp. 5412-5434.
- [6] Green, B.F., Wolf, A.K., Chomsky, C. et Laughery, K. (1961). *Baseball : An Automatic Question-Answerer*, Proceedings of the Western Joint Computer Conference, pp. 219-224.
- [7] Rajpurkar, P., Zhang, J., Lopyrev, K. et Liang, P. (2016). *SQuAD : 100,000+ Questions for Machine Comprehension of Text*, Proceedings of EMNLP, pp. 2383-2392.
- [8] Quarteroni, S. et Manandhar, S. (2009). *Designing an Interactive Open-Domain Question Answering System*, Natural Language Engineering, 15(1), pp. 73-95.
- [9] Mishra, A. et Jain, S.K. (2016). *A Survey on Question Answering Systems with Classification*, Journal of King Saud University - Computer and Information Sciences, 28(3), pp. 345-361.
- [10] Chen, D., Fisch, A., Weston, J. et Bordes, A. (2017). *Reading Wikipedia to Answer Open-Domain Questions*, Proceedings of ACL, pp. 1870-1879. Disponible : <https://aclanthology.org/P17-1171/>
- [11] Kwiatkowski, T., Palomaki, J., Redfield, O., et al. (2019). *Natural Questions : A Benchmark for Question Answering Research*, Transactions of the ACL, 7, pp. 453-466.
- [12] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W.W., Salakhutdinov, R. et Manning, C.D. (2019). *HotpotQA : A Dataset for Diverse, Explainable Multi-hop Question Answering*, Proceedings of EMNLP, pp. 2369-2380.

- [13] Robertson, S. et Zaragoza, H. (2009). *The Probabilistic Relevance Framework : BM25 and Beyond*, Foundations and Trends in Information Retrieval, 3(4), pp. 333-389. DOI : 10.1561/1500000019
- [14] Hochreiter, S. et Schmidhuber, J. (1997). *Long Short-Term Memory*, Neural Computation, 9(8), pp. 1735-1780.
- [15] Pennington, J., Socher, R. et Manning, C.D. (2014). *GloVe : Global Vectors for Word Representation*, Proceedings of EMNLP, pp. 1532-1543.
- [16] Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. et Yih, W. (2020). *Dense Passage Retrieval for Open-Domain Question Answering*, Proceedings of EMNLP, pp. 6769-6781. Disponible : <https://aclanthology.org/2020.emnlp-main.550/>
- [17] Devlin, J., Chang, M.W., Lee, K. et Toutanova, K. (2019). *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding*, Proceedings of NAACL-HLT, pp. 4171-4186.
- [18] Johnson, J., Douze, M. et Jégou, H. (2019). *Billion-scale Similarity Search with GPUs*, IEEE Transactions on Big Data, 7(3), pp. 535-547.
- [19] Xiong, L., Xiong, C., Li, Y., Tang, K.F., Liu, J., Bennett, P., Ahmed, J. et Overwijk, A. (2021). *Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval*, Proceedings of ICLR.
- [20] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S. et Kiela, D. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, Proceedings of NeurIPS, pp. 9459-9474. Disponible : <https://arxiv.org/abs/2005.11401>
- [21] Brown, T.B., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*, Proceedings of NeurIPS, pp. 1877-1901.
- [22] Gao, Y., Xiong, Y., Gao, X., et al. (2023). *Retrieval-Augmented Generation for Large Language Models : A Survey*, arXiv preprint arXiv :2312.10997.
- [23] Ram, O., Levine, Y., Dalmedigos, I., et al. (2023). *In-Context Retrieval-Augmented Language Models*, Transactions of the ACL, 11, pp. 1316-1331.
- [24] codefinty
Modèles Génératifs Basés sur les Transformers
- [25] Nexa,Digital School, June 25, 2025 *Qu'est-ce que BERT, le modèle NLP conçu par Google ?*
- [26] geeksforgeeks,11 septembre 2025 *Modèle BERT - Traitement automatique du langage naturel*
- [27] Varun Mathur,11 mai 2023 *BERT et ses variantes de modèle*
- [28] geeksforgeeks,23 juillet 2025 *T5 (Transformateur de transfert texte-à-texte)*
-

- [29] Zhu, F., Lei, W., Wang, C., Zheng, J., Poria, S. et Chua, T.-S. (2021). *Retrieving and Reading : A Comprehensive Survey on Open-Domain Question Answering*. arXiv preprint arXiv :2101.00774. Disponible en ligne : <https://arxiv.org/pdf/2101.00774.pdf>
- [30] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. et Yih, W.-t. (2020). *Dense Passage Retrieval for Open-Domain Question Answering*. arXiv preprint arXiv :2004.04906. Disponible en ligne : <https://arxiv.org/pdf/2004.04906.pdf>
- [31] Devlin, J., Chang, M.-W., Lee, K. et Toutanova, K. (2018). *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv preprint arXiv :1810.04805. Disponible en ligne : <https://arxiv.org/pdf/1810.04805.pdf>
- [32] Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. et Chen, W. (2021). *LoRA : Low-Rank Adaptation of Large Language Models*. arXiv preprint arXiv :2106.09685. Disponible en ligne : <https://arxiv.org/pdf/2106.09685.pdf>