Projet RAG des contes de fées de Charles Perrault

Nicolas NGAUV

Anissa THEZENAS

Transformers, BERT, RAG

dispensé par Madame Fedchenko

Année universitaire 2024-2025

Master 2 / Semestre 2

1. Introduction

Ce travail s'inscrit dans le cadre de notre master en Traitement Automatique des Langues, et vise à concevoir, implémenter et évaluer un système de question-réponse basé sur l'approche RAG (Retrieval-Augmented Generation). Concrètement, nous avons constitué un index vectoriel FAISS à partir d'un corpus multilingue centré sur l'œuvre de Charles Perrault, puis déployé un modèle de langage pour extraire les passages pertinents et générer des réponses contextualisées en français, anglais et espagnol. Notre étude s'articule autour de quatre axes principaux : la capacité du système à récupérer avec précision des fragments de texte permettant de répondre à des questions factuelles, la qualité et la fiabilité des réponses générées – en particulier l'absence d'hallucinations –, l'adaptabilité multilingue du pipeline, et enfin l'impact des choix de configuration (taille des fragments, modèles d'embeddings, nombre de fragments restitués) sur les performances globales. Le rapport qui suit détaille l'architecture mise en place, le protocole d'évaluation adopté, les résultats obtenus, les difficultés rencontrées et les perspectives d'amélioration envisagées.

2. Constitution de la base de données

Le corpus que nous avons exploité comprend 18 fichiers totalisant 84 256 tokens. Il regroupe d'une part une biographie détaillée de Charles Perrault retraçant sa vie, son œuvre et son influence sur la littérature française, et d'autre part quatre de ses contes emblématiques – "La Belle au bois dormant", "Peau d'âne", "Le Petit Chaperon rouge" et "La Barbe bleue" – chacun disponible en version française (langue originale), anglaise et espagnole (traductions de référence). Après acquisition depuis diverses sources numériques, les fichiers ont été nettoyés et normalisés (suppression des éléments superflus, uniformisation de la casse) puis chargés récursivement dans le système. Afin d'étudier l'effet de la granularité sur la recherche, chaque document a ensuite été segmenté en fragments de 500 ou 2 000 caractères, avec un chevauchement de 200 caractères entre morceaux, selon deux configurations distinctes. Chaque fragment a enfin été converti en vecteur d'embedding à l'aide de deux modèles pré-entraînés : l'un léger (all-MiniLM-L6-v2), optimisé pour la rapidité, et l'autre plus complexe (paraphrase-multilingual-mpnet-base-v2), supposé mieux capter les nuances sémantiques, notamment en contexte multilingue.

3. Architecture et paramétrage

Le cœur du système repose sur trois index FAISS distincts, chacun associant un modèle d'embedding à une stratégie de découpage spécifique. Un premier index utilise MiniLM avec des fragments de 500 caractères, un second emploie MiniLM sur des morceaux de 2 000 caractères, et le troisième s'appuie sur MPNet avec un découpage à 500 caractères. La recherche de similarité approximative permet de récupérer, pour chaque question, les k

fragments les plus proches dans l'espace vectoriel. Le pipeline complet est orchestré par un script Python (rag.py) offrant des options de configuration dynamiques : chemins d'accès au corpus et à l'index, reconstruction de l'index, choix du modèle d'embedding, sélection du modèle de génération (Qwen/Qwen2.5-1.5B-Instruct ou tinyllama), paramètre top-k, taille et chevauchement des fragments, et langue de la question/réponse (codes ISO 639-1). Conçu pour tourner en CPU sur un MacBook Pro à 8 Go de RAM, le système met l'accent sur la reproductibilité et la modularité.

4. Choix du modèle de génération

Les premiers essais ont été réalisés avec tinyllama, dont la légèreté était attrayante pour des itérations rapides. Toutefois, ce modèle a rapidement révélé un fort taux d'hallucinations et des incohérences fréquentes, à tel point qu'il ne parvenait pas à répondre à des questions factuelles simples peut être dû au fait que nous explorions et testions à tâtons le script et les limites, comme la couleur du chaperon rouge. En conséquence, nous avons adopté Qwen2.5-1.5B-Instruct. Malgré un temps de réponse plus élevé (de l'ordre de 20 à 80 secondes), ce dernier a fourni des réponses plus stables et plus cohérentes, a mieux respecté la consigne "Répondez uniquement d'après le contexte" et a démontré de solides capacités multilingues.

5. Protocole d'évaluation

Pour évaluer rigoureusement l'ensemble, nous avons défini quatre scénarios : variation de top-k, comparaison MiniLM vs MPNet, comparaison Qwen vs TinyLlama, et désactivation du filtre de langue. Dans chaque cas, trois questions identiques – "Qui est Charles Perrault ?" (FR), "Who is Charles Perrault?" (EN), et "¿Quién es Charles Perrault?" (ES) – ont été soumises au système. Nous avons mesuré la retrieval_precision (proportion de fragments réellement pertinents), exact_match, F1 token-level, BLEU, ROUGE-L, similarité sémantique (emb_sim), cohérence linguistique (lang_consistent), taux de réponses "Je ne sais pas" (unknown), latence (latency_s) et variation mémoire (mem_delta_mb).

6. Résultats et interprétation

Dans l'ensemble des expérimentations menées, la précision du retrieval est restée nulle : quel que soit le paramètre *top-k*(de 1 à 10) ou le modèle d'embedding employé (MiniLM ou MPNet), l'index FAISS n'a jamais restitué le fragment contenant littéralement la réponse de référence. Cette défaillance de la phase de recherche se répercute directement sur les métriques textuelles : les scores BLEU demeurent compris entre 0,0 et 0,01, les scores ROUGE-L varient de 0,03 à 0,08, et les F1 token-level oscillent entre 0,02 et 0,08. En dépit de ces valeurs faibles, la similarité sémantique (emb_sim) se maintient autour de 0,60–0,63, ce qui témoigne d'une certaine cohérence thématique : les réponses générées restent globalement « sur le sujet », mais n'en restituent que des portions partielles et

approximatives. Par ailleurs, le taux de réponses « Je ne sais pas » est systématiquement nul, ce qui révèle la propension des modèles de langage à inventer des informations plutôt qu'à admettre leurs limites, même lorsqu'une consigne explicite de retournement de réponse leur a été donnée. En revanche, la cohérence linguistique est parfaite dans tous les scénarios.

Sur le plan des performances, TinyLlama s'est distingué par une latence très réduite (5 à 10 secondes par requête) et une empreinte mémoire moindre, contre 20 à 80 secondes et une consommation plus importante pour Qwen2.5-1.5B-Instruct. Cette caractéristique fait de TinyLlama un outil de choix pour des itérations rapides. En revanche, Qwen2.5-1.5B-Instruct, malgré une latence accrue, a offert une qualité de génération plus stable et un meilleur respect des consignes de forme.

Le cas de figure *top-k* = 1 illustre particulièrement bien nos difficultés de retrieval. Pour la question « Qui est Charles Perrault ? » en français, le système a extrait un long extrait biographique puis généré la réponse « Charles Perrault était un écrivain français célèbre pour ses contes pour enfants… », là où la référence attendue était « écrivain français du XVII^{e} siècle, auteur de célèbres contes de fées ». Sur ce simple scénario, la précision de retrieval est de 0, la F1 token-level s'élève à 0,0766, le BLEU à 0,0, le ROUGE-L à 0,0794, et l'emb_sim à 0,5968. Cette seule requête a nécessité 12,71 secondes de calcul en CPU et a fait croître l'usage mémoire de près de 6 160 Mo. Lorsque l'on passe à l'anglais pour « Who is Charles Perrault? », la génération reste tout aussi décalée (« French author known for fairy tales » au lieu de « French writer, known for famous fairy tales »), avec un BLEU de 0,0086, un ROUGE-L de 0,0506, un emb_sim à 0,6231, une latence de 18,74 secondes et un surplus de mémoire de 95 Mo. En espagnol, la similarité sémantique se situe autour de 0,6042, mais la latence grimpe à 59,06 secondes en raison de contextes plus volumineux, et l'on observe un relâchement mémoire de 465 Mo après garbage collection.

Augmenter *top-k* à 3 n'a pas permis d'améliorer ces indicateurs : en français, la latence atteint 69,06 secondes et l'usage mémoire 6 698,95 Mo, tandis que la F1 chute à 0,0242 et le ROUGE-L à 0,0339. Ces résultats confirment que multiplier les fragments ne suffit pas à cibler l'information pertinente, tant la segmentation actuelle dilue ou morcelle les passages clés.

Comparativement, TinyLlama, soumis au même protocole (*top-k* = 1, embedding MiniLM), répond en 5,20 secondes, soit environ quatre fois plus vite que Qwen. Les scores qu'il atteint (F1 = 0,0445, BLEU = 0,0114, ROUGE-L = 0,0433, emb_sim = 0,5968) sont très proches de ceux de Qwen, mais l'empreinte mémoire se limite à 4 082 Mo. TinyLlama apparaît ainsi comme un compromis temps/qualité pertinent pour des phases exploratoires, tandis que Qwen reste préférable lorsque la précision formelle prime, malgré sa lenteur.

Enfin, l'index dédié à l'embedding MPNet n'a guère modifié la retrieval_precision (toujours 0), mais il a permis de faire passer emb_sim à environ 0,62, au prix d'un coût plus élevé en

temps d'indexation et en latence de recherche. Cette amélioration marginale de la similarité sémantique confirme qu'un modèle d'embedding plus riche peut renforcer la cohérence globale, sans toutefois résoudre le problème fondamental de segmentation et de ranking.

En somme, ces observations chiffrées illustrent clairement que, sans un retrieval fiable et un indexage fin, la génération restera irréductiblement marquée par des approximations et des hallucinations, quelle que soit la taille du contexte fourni ou la puissance du modèle de langage employé.

7. Problèmes identifiés

L'analyse fine des résultats révèle que le principal goulot d'étranglement de notre pipeline réside dans la phase de retrieval : quelle que soit la configuration de découpage (taille des chunks ou chevauchement) et le choix du modèle d'embedding, l'index FAISS n'est jamais parvenu à isoler le fragment littéralement porteur de la réponse de référence. Cette incapacité à localiser précisément l'information découle d'une stratégie de fragmentation qui morcelle trop le texte ou, a contrario, dilue l'essentiel dans des passages trop volumineux. Les passages essentiels sont ainsi fréquemment répartis sur plusieurs chunks ou noyés au milieu de contenus moins pertinents, ce qui empêche la similarité cosinus appliquée aux embeddings de les distinguer efficacement.

À cet égard, les modèles d'embedding testés montrent leurs limites. Le modèle all-MiniLM-L6-v2, par sa compacité, ne saisit pas toujours les nuances sémantiques subtiles propres aux textes classiques de Perrault : des distinctions lexicales (par exemple, entre « écrivain » et « conte ») ou des allusions culturelles (références au XVII^{e} siècle) échappent à son espace vectoriel. Le recours à un embedding plus lourd, tel que paraphrase-multilingual-mpnet-base-v2, améliore légèrement la cohérence sémantique (emb_sim), mais n'infléchit pas la précision du retrieval, ce qui indique que la finesse de la représentation sémantique, bien que nécessaire, ne suffit pas à elle seule pour résoudre le problème d'identification du segment véritablement informatif.

Un troisième facteur contributif à cette défaillance est la métrique de similarité utilisée pour le ranking des chunks. La similarité cosinus, bien qu'efficace dans de nombreux contextes, s'avère insuffisante pour discriminer des passages qui partagent globalement la même thématique mais n'en contiennent pas la réponse factuelle. Cette métrique uniforme favorise parfois des fragments volumineux contenant des vocabulaires généraux plutôt que des passages courts et précis, ce qui se traduit par un taux de retrieval_precision systématiquement nul.

Par ailleurs, l'évaluation a mis en lumière le comportement des modèles de génération face à un contexte déficient : confrontés à l'absence de passages parfaitement alignés avec la question, les LLM ont systématiquement « halluciné » plutôt que d'admettre l'inconnu.

Aucune occurrence de la réponse « Je ne sais pas » n'a été enregistrée, malgré une consigne explicite invitant le modèle à reconnaître ses limites. Cette propension à inventer des informations non présentes dans le corpus compromet gravement la fiabilité factuelle du système.

Enfin, les contraintes matérielles introduisent une tension supplémentaire entre la qualité et la faisabilité opérationnelle : sur notre configuration CPU limitée à 8 Go de RAM, l'augmentation de *top-k* fait exploser la latence et l'usage mémoire sans gain tangible en pertinence, ce qui restreint l'échelle et la complexité des expériences envisageables.

Au total, ces observations soulignent que sans amélioration de la stratégie de segmentation, de la richesse des embeddings et du mécanisme de ranking, la génération, même par un LLM de haute qualité, restera prisonnière d'un contexte mal localisé et produira des réponses approximatives, parfois inventées.

8. Pistes d'amélioration

Pour remédier à ces limites, nous proposons de repenser la stratégie de découpage en fragments : passer à un découpage sémantique (phrases, paragraphes, repères structurés), ou adapter dynamiquement la granularité afin de préserver l'intégrité des informations. L'ajout d'un module de re-ranking basé sur un second modèle, plus puissant, permettrait de trier les fragments initialement récupérés afin de sélectionner ceux qui contiennent effectivement la réponse. Le fine-tuning du LLM sur notre corpus de Charles Perrault, via un entraînement ciblé sur des paires questions - réponses extraites du texte, renforcerait la précision et la cohérence de la génération. Enfin, l'enrichissement de l'index FAISS avec des métadonnées (titres, chapitres, balises de scène) offrirait des indices supplémentaires pour guider le retrieval.

9. Conclusion

L'évaluation détaillée de notre système RAG appliqué à l'œuvre de Charles Perrault confirme que la pertinence des réponses dépend avant tout de la qualité du retrieval. Sans un index affiné et des embeddings adéquats, même le meilleur LLM ne peut corriger un contexte mal ciblé. Si TinyLlama témoigne d'une efficacité remarquable en latence, Qwen se révèle plus fiable sur la forme et le fond. Les améliorations envisagées — découpage sémantique, re-ranking, fine-tuning, enrichissement des métadonnées — constituent autant de voies prometteuses pour doter un système RAG multilingue de la capacité à fournir des réponses à la fois factuelles, contextualisées et robustes.