Introduction au RAG pour les LLM

Retrieval-Augmented Generation

Philippe Helluy

Plan du cours

1. Introduction et motivation
2. Concepts fondamentaux
3. Architecture d'un système RAG
4. Embeddings et similarité
5. Stockage vectoriel
6. Récupération de documents
7. Génération augmentée
8. Mise en pratique

Rappels sur les LLM

Large Language Model (LLM)

- Modèle de langage pré-entraîné sur d'énormes corpus de texte
- Capable de générer du texte cohérent
- Exemples : Gemini, Qwen, Mistral, etc.

Limitations :

- Connaissance figée au moment de l'entraînement
- Hallucinations (génération d'informations fausses)
- Pas d'accès à des données spécifiques ou privées

Comment donner accès à des informations à jour ou spécifiques ?

Solutions possibles :

- Ajouter des données dans le prompt ? → Limité par la taille du contexte
- Ré-entraîner le modèle ? → Coûteux et lent
- Fine-tuning ? → Limité, risque de *catastrophic forgetting*
- **RAG** → Léger, flexible, efficace

Qu'est-ce que le RAG ?

Retrieval-Augmented Generation

Technique qui combine :

1. **Retrieval (Récupération)** : Sélection d'informations pertinentes dans une base de données
2. **Augmentation** : Enrichissement du contexte dans le prompt du LLM, sans dépassement de la taille maximale
3. **Generation** : Production de réponse par le LLM avec le contexte enrichi

Principe de base du RAG


```
Question utilisateur
      ↓
Récupération de documents pertinents (base de données)
      ↓
Construction du prompt avec les données sélectionnées
      ↓
Génération par le LLM
      ↓
Réponse basée sur les documents
```

Architecture générale

Phase d'indexation

1. Documents sources
2. Découpage en chunks
3. Génération d'embeddings
4. Stockage vectoriel

Phase de requête

1. Question utilisateur
2. Embedding de la question
3. Recherche de similarité
4. Génération de réponse

Phase 1 : Indexation

Préparation de la base de connaissances

1. **Collecte** : Rassembler les documents (PDF, web, bases de données)
2. **Découpage** : Diviser en *chunks* (morceaux) de taille appropriée
3. **Embedding** : Convertir chaque chunk en vecteur numérique
4. **Stockage** : Sauvegarder dans une base vectorielle

Qu'est-ce qu'un embedding ?

Représentation vectorielle d'un texte

- Transformer le texte en vecteur de nombres réels avec un réseau de neurones (appelé encodeur ou modèle d'embedding)
- Le vecteur est appelé *embedding* (plongement en français)
- Le modèle est entraîné pour que des textes similaires aient des embeddings proches dans l'espace vectoriel

Exemple :

- "chat" et "félin" → vecteurs proches
- "chat" et "voiture" → vecteurs éloignés

Modèles d'embedding

Exemples de modèles populaires :

- Sentence-BERT (SBERT)
- all-MiniLM-L6-v2
- multilingual-e5-base
- text-embedding-ada-002 (OpenAI)

Caractéristiques importantes :

- Dimension du vecteur (384, 768, 1536...)
- Support multilingue
- Compromis précision/vitesse

Découpage des documents en morceaux

Stratégies :

- Par taille fixe (ex: 500 tokens)
- Par paragraphe ou section
- Avec chevauchement (*overlap*)

Compromis :

- Chunks trop petits → perte de contexte
- Chunks trop grands → dilution de l'information

Stockage et recherche de vecteurs

Quelques logiciels:

- FAISS (Facebook AI)
- Chroma
- Pinecone
- Weaviate
- Milvus

Permettent une recherche efficace par similarité

Mesures de similarité

Comment comparer deux vecteurs ?

1. **Similarité cosinus** : Angle entre vecteurs - Valeur entre −1 et 1 - Indépendant de la magnitude
2. **Distance euclidienne** : Distance géométrique

Recherche de documents pertinents

1. Recevoir la question de l'utilisateur
2. Générer l'embedding de la question
3. Calculer la similarité avec tous les chunks
4. Sélectionner les k chunks les plus similaires

Généralement : k = 3 à 5 documents. Si la base de données est grande, on peut augmenter k. Algorithme efficace de recherche (*Approximate Nearest Neighbors*, ANN).

Phase 3 : Génération

Construction du prompt augmenté

```
Contexte:
[Chunk 1 pertinent]
[Chunk 2 pertinent]
[Chunk 3 pertinent]

Question: [Question de l'utilisateur]

Réponse:
```

Le LLM complète et génère une réponse basée sur le contexte fourni (en plus de ses propres connaissances).

Avantages du RAG

- Accès à des informations à jour
- Pas besoin de ré-entraînement
- Réduction des hallucinations
- Sources traçables
- Données privées/spécifiques
- Moins coûteux que le fine-tuning
- Facilement actualisable

Limites et défis

- Qualité dépend de la base de documents
- Chunking optimal difficile à déterminer
- Coût computationnel de l'embedding
- Taille du contexte limitée
- Besoin d'infrastructure (base vectorielle)
- Latence de récupération

Variantes et améliorations

Techniques avancées :

- **HyDE** : Hypothetical Document Embeddings
- **Re-ranking** : Réordonnancement des résultats
- **Query expansion** : Enrichissement de la requête
- **Multi-query** : Plusieurs formulations
- **Fusion** : Combinaison de plusieurs stratégies

RAG vs Fine-tuning

Critère	RAG	Fine-tuning
Coût	Faible	Élevé
Mise à jour	Facile	Difficile
Données	Documents	Pairs Q/R
Traçabilité	Oui	Non
Latence	Moyenne	Faible

Cas d’usage

Où utiliser le RAG ?

- Documentation technique (chatbot d’aide)
- Support client (base de connaissances)
- Recherche scientifique (analyse d’articles)
- Juridique (recherche dans les lois)
- Médical (informations à jour)
- Entreprise (données internes)

Frameworks et outils

Bibliothèques Python populaires :

- LangChain : Framework complet
- LlamaIndex : Indexation et récupération
- Haystack : Pipelines NLP
- Transformers : Modèles HuggingFace
- Sentence-Transformers : Embeddings

Pipeline RAG simple

```
# 1. Charger documents
documents = load_documents("data/")

# 2. Créer embeddings
embeddings = create_embeddings(documents)

# 3. Stocker dans DB vectorielle
vector_db = VectorStore(embeddings)

# 4. Requête
query = "Ma question"
docs = vector_db.search(query, k=3)

# 5. Générer réponse
response = llm.generate(docs, query)
```

Évaluation d'un système RAG

Métriques importantes :

- **Précision** : Documents pertinents récupérés
- **Rappel** : Tous les documents pertinents trouvés
- **MRR** : Mean Reciprocal Rank
- **NDCG** : Normalized Discounted Cumulative Gain

Qualité de génération :

- Fidélité aux sources
- Pertinence de la réponse
- Cohérence

Considérations pratiques

1. Choisir la bonne taille de chunks
2. Utiliser des embeddings adaptés à la langue
3. **Nettoyer et prétraiter les documents**
4. Prévoir un système de cache
5. Suivi des requêtes pour amélioration
6. Tester avec différents paramètres k
7. Valider les sources retournées

En production

- **Scalabilité** : Gérer des millions de documents
- **Performance** : Optimiser les temps de réponse
- **Sécurité** : Contrôler l'accès aux documents
- **Supervision** : Suivre la qualité des réponses
- **Coûts** : appels aux API calls, stockage, calcul

Tendances

- RAG multi-modal (texte + images)
- RAG conversationnel (mémorisation)
- Agents autonomes avec RAG
- **RAG avec graphe de connaissances**: prochain cours
- Optimisation automatique des hyperparamètres

Travaux pratiques

Nous allons voir :

1. Chargement de documents markdown
2. Génération d'embeddings
3. Recherche par similarité
4. Génération de réponse avec un LLM

Code disponible dans `https://github.com/phelluy/ragllm`