Power Week 2025

#pw2025

18 - 19 - 20 novembre 2025

IBM Innovation Studio Paris

S19 – Comment rendre un LLM open source plus pertinent pour l'IBM i avec un simple RAG ?

18 novembre 14:45 - 15:45

Gabriel AMPHOUX
CFD-Innovation
gamphoux@cfd-innovation.fr

IBM

common FRANCE

〈INNOVATION〉

- Conseil
- Formation
- Développement

ASSISTANCE

- Web et Open Source — php, git, open source, node.js
- Communication — OpenSSL Cryptography and SSL/TLS Toolkit, SOAP, REST, Web Services
- Valorisation des données — SYD, SQL, IBM DB2

Sommaire

- Un peu de vocabulaire
- Interroger un LLM
- Comment améliorer un LLM ?
  - Fine-tuning : Solution complexe et couteuse mais très efficace
  - RAG : Solution plus simple et très peu couteuse
- Le RAG c'est quoi ?
- Un RAG en local
- Une API et un front sur IBM i
- La démo

Un peu de vocabulaire :

- **Model (Modèle) :** Réseau de neurones entraîné à une tâche spécifique. Par exemple comprendre ou générer du texte. J'utiliserai aussi LLM (large language model) pour parler de gros modèles.
- **Contexte (Prompt) :** Informations fournies au modèle avant la génération.
- **Inférence :** Étape où le modèle génère la réponse. On peut dire que c'est le temps d'attente entre l'envoi d'une requête à un modèle et la génération du message.

Suite du vocab

- **Chunk :** Morceau de texte découpé à partir d'un document (paragraphe, section...). Facilite l'analyse et la recherche dans les données.
- **Vecteur / Embedding :** Représentation numérique d'un texte dans un espace sémantique. Deux textes proches en sens → vecteurs proches.

Interroger un LLM

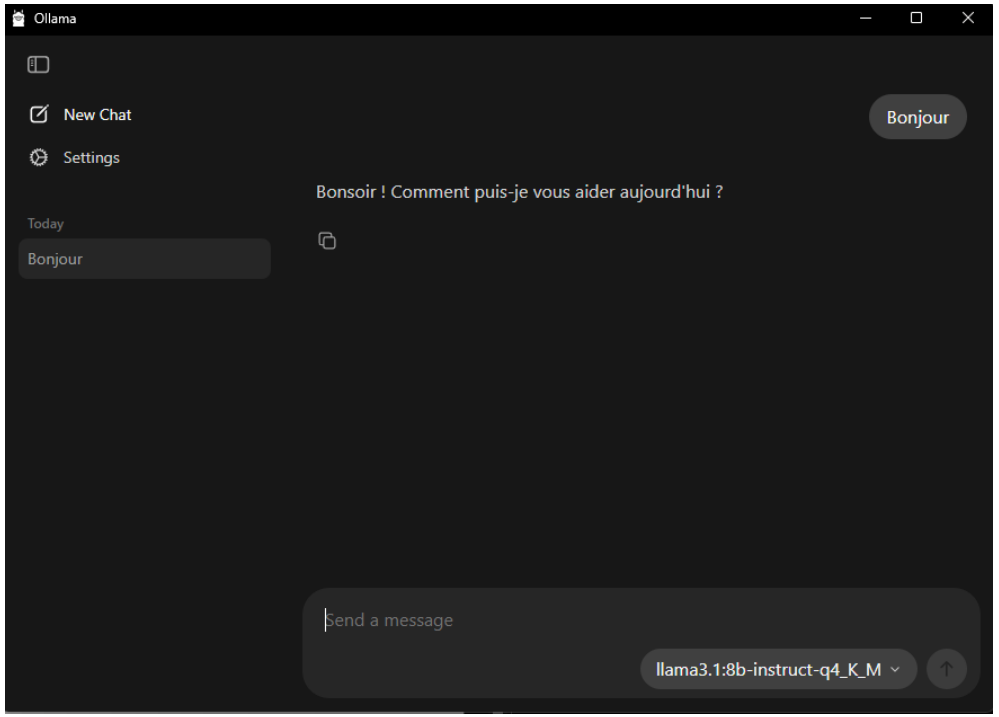
- Utilisation ChatGPT => on interroge un LLM de OpenAI (souvent GPT-5 ou GPT-4)
- Utilisation d'un model avec API => On peut interroger un model à distance => on n'a pas à héberger le model qui peut demander beaucoup de puissance de calcul.
- Utilisation d'un model local => On peut par exemple utiliser llama.cpp pour interroger un model qu'on a en local. C'est possible de le faire directement sur son IBM i !

Model open source

- Quand on parle de model en local => On parle d'un model dont on a récupéré tous les poids, il faut utiliser les bons outils pour l'interroger :
  - Bibliothèque PyTorch ou Tensorflow sur python
  - Produit un peu plus complet : llama.cpp => qu'on va pouvoir utiliser pour avoir un model en local sur son ibm i ! (Pytorch n'étant pas disponible)
  - Produit encore plus complet OLLAMA => Se base sur llama.cpp c'est ce qu'on va utiliser pour aujourd'hui.

Exemple d'utilisation d'un model en local

- Etape 1 : télécharger ollama et le model qu'on veut interroger.
- Etape 2 : écrire un petit script python pour l'interroger ou lancer simplement ollama en choisissant le model :



Et si on veut APIser tout ça :

- On fait notre petit script python qui utilise FAST-API pour exposer facilement des services et on interroge notre model.
- Interrogation du model :

```
def infer_ollama(prompt: str) -> str:
    payload = {
        "model": OLLAMA_MODEL,
        "prompt": prompt,
        "stream": False,
    }
    try:
        r = requests.post(OLLAMA_URL, json=payload, timeout=60)
        r.raise_for_status()
        return r.json().get("response", "")
    except requests.RequestException as e:
        raise HTTPException(status_code=502, detail=f"Erreur Ollama: {e}")
```

Suite

- Route fast API :

```
@app.get("/llama-38b/norag")
def mon_chemin_message(payload: Message = Body(...,
description='{"msg":"ton message"}')):
    if not payload.msg or not payload.msg.strip():
        raise HTTPException(status_code=400, detail="Le champ 'msg' est
requis et non vide.")
    return {
        "request": payload.msg,
        "response": infer_ollama(payload.msg.strip()),
        "model": f"ollama:{OLLAMA_MODEL}",
    }
```

Et donc pour l'interroger

- Il suffit d'appeler notre API, on peut tester avec un CURL :

```
curl -X GET -H "Content-Type: application/json" -d '{"msg":"Quel IBM i service
pour voir le storage utilisateur ?"}' http://localhost:8000/llama38b/norag
```

- La réponse pourrait être : (prochain slide)

{"request":"Quel IBM i service pour voir le storage utilisateur ?","response":"Pour afficher les informations de stockage utilisateurs sur l'IBM i, vous pouvez utiliser la commande `DSPUSRSTOR` (Display User Storage) ou bien les outils d'interface utilisateur tels que le System Administrator Workbench (SAW) ou System i Navigator.\n\n- **Ligne de commande** : La commande `DSPUSRSTOR` est un commanditaire qui permet de récupérer des informations sur la consommation de stockage utilisateurs. Vous pouvez y accéder en ligne de commande à partir du système IBM i, voici comment vous l'utiliser :\n\n ```sql\n DSPUSRSTOR USER(*ALL) OUTPUT(*OUTFILE) FILE(USRTOT.TXT)\n ```\n\n- **Interface utilisateur** : L'interface System Administrator Workbench (SAW) ou System i Navigator permet également d'accéder aux informations de stockage utilisateur sous une forme plus graphique et intuitive. Cela implique de se connecter au système IBM i via l'une des interfaces disponibles.\n\nSi vous cherchez à utiliser une interface utilisateur spécifique pour afficher les données, j'encouragerais à essayer le **System Administrator Workbench (SAW)** ou **System i Navigator**, qui sont tous deux conçus pour offrir une vue d'ensemble du système IBM i et sa gestion.\n\nSi vous préférez utiliser la ligne de commande, alors `DSPUSRSTOR` est l'outil que vous recherchez.","model":"ollama:llama3.1:8b-instruct-q4_K_M"}

- Fausse et beaucoup trop longue !
- On va essayer de voir comment améliorer ça

Fine-tuning

- Quand on pense améliorer un model on pense fine-tuning => réentrainement.
- C'est en effet efficace mais il faut :
  - Enormément de données
  - Une équipe qualifiée :
    - Pour mettre en place les aspects techniques.
    - Pour pouvoir évaluée la pertinence de l'entrainement.
  - Des ressources machines importantes, c'est bien plus demandant d'entrainer que de consommer un model.

Donc le fine-tune

- C'est cher et les entreprises qui ont essayer de fine-tune un model pour du RPG ont une ou plusieurs équipes entièrement dédiées à ça.
- Nous on aimerait voir comment rendre ça plus performant mais sans que ça nous coute trop. => On en veut toujours plus pour moins !
- Et c'est là qu'intervient le RAG.

RAG qu'est ce que c'est ?

- Retrieval Augmented Generation => génération augmentée de récupération.
  Pour faire simple le RAG c'est deux étapes :
  - Récupération : lors d'une requête il va commencer par chercher dans un ensemble de documents (ou de data) les informations les plus pertinentes pour répondre à la requête.
  - Génération : il utilise les informations récupérées comme contexte de la requête et génère la réponse en fonction.
- Donc ça nous permet d'ajouter des informations propres à notre système lors d'une demande.

Comment ça fonctionne techniquement

- On a donc plusieurs étapes :
  - EN AMONT :
    - Récupération de notre documentation
    - Indexation de tous ces documents
  - Avantage : on maitrise totalement les informations qu'on lui donne => plus récentes que celle sur lesquels il a été entrainé, données inexistantes ou peu présentes dans son entrainement, on peut facilement les mettre à jour.
  - AU MOMENT DE LA REQUETE :
    - Indexation de la requête pour pouvoir rechercher dans l'index de nos documents les points pertinents
    - Récupération des X points les plus pertinent, ajout de ces derniers dans le contexte de la requête

Etape par étape

- Récupération de la documentation :
  - Sans surprise : on récupère notre documentation. Si on a des doc en interne pour un ERP ou des pdfs de documentation.
  - On peut mettre beaucoup de données mais ce n'est pas toujours pertinent de mettre trop d'informations similaires ou au contraire trop différentes.
  - Il est possible de faire des récupérations thématiques :
    - Uniquement des docs sur les commandes système pour un exploitant.
    - Uniquement de la documentation métiers d'un ERP pour un utilisateur non technique.

Indexation des documents

- On transforme les documents en vecteur avec un model prévu à cet effet. On appelle ça l'embeding (ou vectorisation) et on utilise des modèles d'embeding.
- Une fois qu'on les a vectorisé, on indexe nos vecteurs pour pouvoir faire des recherches plus rapides.
- Pour ça plusieurs façons de faire :
  - Utiliser FAISS sur python, une bibliothèque prévue à cet effet => très utile pour un très gros jeu de données.
  - On peut cependant le faire sans FAISS avec OLLAMA et NUMPY, c'est largement suffisant pour un jeu de données restreint et notre démo.

Demande utilisateur

- L'utilisateur fait une requête :
  - On embed la requête et on recherche dans la base documentaire indexée les vecteurs les plus proches. Enfin "on", un model le fait.
  - On récupère les "n" chunk (morceau de texte) correspondant aux indexs les plus proches et on les ajoute au prompt pour le LLM.
  - On peut choisir le nombre de chunk, attention, plus on en met, plus le contexte sera long donc plus il sera long à traiter pour le LLM.
  - On envoie la requête d'utilisateur avec le contexte qu'on y a ajouté.

Pour notre démo, qu'est ce qu'on va faire ?

- 1 er étape on a embed des pdf sur les IBM i services => pourquoi ? Parce que les LLM sont mauvais sur les ibm i services.
- 2 ème étape => on a récupéré un model que je fais tourner en local sur ma machine => pourquoi ? Parce que je pourrais vouloir que mes données ne sortent pas de mon réseau.
- 3 ème étape => je fais une API toute simple en local sur mon pc pour interroger le model.
- 4 ème etape => je fais une API qui me permet d'embed la demande, de chercher les n chunks les plus pertinent, de les ajouter à la requête et d'interroger un model local.

La suite ?

- Pour la partie back on sera bon ! On a fait tout ce qu'il fallait faire :
  - On peut faire tourner cette API sur un PC connecté à notre réseau.
  - Il faudra probablement penser à vérifier que le pare-feu ne bloque pas les demandes.

- Maintenant prenons un IBM i connecté à ce même réseau et intéressons-nous au front ! Le but c'est de pouvoir interroger le model "RAGgé" depuis notre IBM i.

Comment faire ?

- Simplement en BASH avec une commande CURL. Très bien pour du test mais difficilement lisible.
- En SQL avec les requêtes HTTP GET CLOB, ça pourrait nous permettre de faire un mini chat en RPG !
- Dans un autre langage permettant d'interroger une API. Et nous on va le faire sur une page web grâce à du PHP (du JS, HTML et CSS aussi évidemment).

Plus en détails :

- Le PHP pour interroger l'API qu'on a fait.
- Le HTML/CSS pour le style et l'affichage de la page.
- Le JS pour ne pas avoir à recharger la page quand on attend un message et afficher "l'assistant réfléchis" ou ce genre de chose.
- Le code est disponible sur le gitlab suivant : https://gitlab.com/cfd-innovationoss/chat-rag-for-i

Et qu'est-ce que je constate ?

- Que la réponse avec RAG est bien meilleure sur les sujets propres aux ibm i services.
- Que c'est un peu plus long à l'inférence => l'inférence c'est quand on interroge un model, le temps de réponse.
- Normal, on agit au moment de l'inférence (embed de la requête, recherche et construction du prompt).

Maintenant la démo

- On se base sur les deux projets :
  - https://gitlab.com/cfd-innovationoss/rag-for-i
  - https://gitlab.com/cfd-innovationoss/chat-rag-for-i

- Commençons par le commencement :
  - Code de l'embed et index => set_up_rag.py => présentation rapide du code de main => on voit surtout dans la méthode embed que j'appelle un model pour faire de l'embeding.

Exposition de l'API

- app.py => on voit deux routes NORAG et RAG.
- Pour NORAG rien de nouveau.
- Pour RAG, on voit :
  - L'étape retrieve : Pour chercher les données correspondantes à la demande. Dans cette étape on a :
    - l'embed de la requête (avec la méthode embed_text qui appelle le model d'embeding utilisé pour embed la base de pdf)
    - la recherche des vecteurs les plus proches avec le cosinus et les tris de tableau.
  - L'étape build_prompt où on assemble les données et la demande.
  - Enfin on a l'étape infer_ollama où on interroge le modèle de génération de texte

Dernière chose à faire :

- Embed nos pdf : python set_up_rag.py
  - On ne le fait pas ici, c'est une étape qui peut être longue en fonction de la quantité de pdf.
- Lancer notre API : uvicorn app:app --host 0.0.0.0 --port 8000
- Et on est tout bon pour le back

Construction du chat

- On part du principe qu'on a bien rempli le fichier de config.
- api.php : on interroge notre API grâce à CURL (ligne 37 à 51). On voit aussi qu'on choisit si on interroge l'API avec ou sans RAG grâce aux ligne 29 et 32.
- index.php : corps de notre page. On voit la structure très simple de cette page et l'intégration du script.js ligne 47.
- script.js : ligne 70 fonction sendMessage on voit qu'on se base sur notre fichier api.php pour interroger notre API (ligne 76) on voit aussi qu'on dit ligne 83 si on veut utiliser le RAG ou non.
- style.css : contient le style de notre page

Et maintenant on interroge :

Notre page est accessible ici : http://monIBMi:XXXX/chatRAG/

Testons ça ensemble.

Des questions ?


MERCI