Dépasser les limites des LLM

Cours ≪ Recherche d'Information et Graphe de Données ≫

Nicolas Delestre

INSA ROUEN NORMANDIE

Aides de mes amis imaginaires

- Ce cours a été rédigé en LaTeX sous Emacs avec le mode Copilot
- J'ai très souvent échangé avec ChatGPT (version 5.2) et Gemini (version 3) pour présenter les concepts de ce cours plus clairement

INSA ROUEN NORMANDIE

Plan

INSA ROUEN NORMANDIE

Limites des LLM

Constats

- La connaissance d'un LLM est bornée par la période de collecte de ses données d'entraînement
- Un LLM généraliste n'a pas accès aux connaissances privées ou métier
- Sans mécanisme externe, un LLM ne peut ni interroger, ni agir un système

Conséquences

- Erreurs et hallucinations fréquentes
- Format de réponses non adaptés aux métiers

INSA ROUEN NORMANDIE

Quatre familles de solutions

Idée directrice

Relier un LLM à un besoin spécifique peut se faire en modifiant le modèle, en enrichissant son contexte, en lui donnant des outils (avec connexions *ad hoc* ou standardisées)

Les trois/quatre approches

- *Fine-tuning*
- *Retrevial-Augmented Generation* (RAG)
- Agentification
- *Model Context Protocol* (MCP)

INSA ROUEN NORMANDIE

Fine-tuning : principe

Définition

Le fine-tuning ajuste un modèle pré-entraîné sur un jeu de données ciblé afin de modifier ses comportements et ses connaissances

Quand l'utiliser

- Style de réponse souhaité et stable
- Tâche répétitive et bien définie
- Corpus d'apprentissage suffisamment large et propre

INSA ROUEN NORMANDIE

Fine-tuning : données et prérequis

Données d'apprentissage

- Type de données :
  - Paires (instruction, réponse)
  - Conversations annotées
  - Exemples positifs et négatifs pour préférences
- Quantité de données :
  - Quelques milliers d'exemples pour un style ou une tâche simple
  - Plusieurs dizaines de milliers pour des comportements plus complexes ou variés

Prérequis

- GPU avec plus de VRAM que pour l'inférence (coût financier)
- Plusieurs heures à plusieurs jours d'entraînement selon la taille du modèle et des données d'apprentissage
- Des compétences en *machine learning* pour préparer les données, ajuster les hyperparamètres et évaluer le modèle appris

Fine-tuning : avantages et limites

Avantages

- Réponses plus cohérentes avec un domaine ou un style
- Latence faible à l'inférence
- Moins de dépendance à un système de recherche externe

Limites

- Coût d'entraînement et maintenance
- Mise à jour lente si la connaissance change souvent
- Risque d'apprendre du bruit ou des biais du aux données d'apprentissage

INSA ROUEN NORMANDIE

Idée générale

Retrieval-Augmented Generation

- Enrichir le prompt avec des passages de documents retrouvés au moment de la requête
  - FAQ d'entreprise
  - Assistant de recherche documentaire
  - Système de question-réponse sur des données métier
- Fournir au LLM une connaissance à jour et contextualisée sans réentraîner le modèle tout en pouvant citer les sources

INSA ROUEN NORMANDIE

Processus

Étapes du fonctionnement d'un RAG

- Phase d'indexation
  - Découper les documents en segments (*chunks*)
  - Calculer des embeddings et indexer dans une base vectorielle
- Phase de requête
  - Rechercher les segments pertinents pour une question
  - Construire un prompt avec ces segments et la question de l'utilisateur
  - Générer une réponse et citer les sources

Avantages

- Pour mettre à jour la connaissance, il suffit de réindexer les documents
- Indépendant du modèle, peut être utilisé avec n'importe quel LLM
- Coût de mise en place et de maintenance plus faible que le fine-tuning

Lien avec les embeddings

Importance des embeddings

- La recherche sémantique dépend de la qualité des représentations
- Les choix de chunking influencent la pertinence et la précision
- Les métriques de similarité et le reranking impactent le top-k

La nécessité d'utiliser une base de données vectorielle

- Permettre de stocker et de rechercher efficacement des millions de segments
- Faciliter la mise à l'échelle
- Quelques base de données vectorielle open source : Pinecone, Weaviate, Milvus, Chroma (prototypage), pgvector (extension de PostgreSQL)

Points de vigilance

Sources d'échec fréquentes

- Mauvais *chunking*
- Documents mal interprétés (PDF, image, etc.)
- Bruit et silence dans le contexte du prompt (performance de l'*embedding*) et donc dans les résultats
- Prompt trop long qui peut poser problème dans le contexte/historique du chatbot
- Réponse non contrainte qui extrapole au-delà des sources

Bonnes pratiques

- Ajouter citations et extraits utilisés
- Utiliser un *reranker* : reclasser, et possiblement réduire, le nombre de passages fournis au LLM (embedding plus performant, mais plus lent, ou même utiliser un LLM spécialisé dans le classement)
- Évaluer le RAG avec un jeu de questions représentatif

Agentification : du texte à l'action

Définition

Un agent combine un LLM avec des outils et une boucle de décision pour accomplir une tâche

Ce que cela apporte

- Interroger des systèmes
- Enchaîner des étapes de résolution
- Automatiser des actions contrôlées

INSA ROUEN NORMANDIE

Agentification : boucle de fonctionnement

Boucle simplifiée

- Interpréter la demande
- Planifier ou choisir une action
- Appeler un outil
- Observer le résultat
- Répéter jusqu'à obtention d'une réponse qui ne fait plus appel à un outil

Exemple ChatBot utilisant des outils avec langchain 1 / 7

main

```
43 def main(modele: str, url: str, cle_api: str = "XXX"):
44     llm = ChatOpenAI(model=modele,
45                      base_url=url,
46                      api_key=cle_api)
47     agent = create_agent(llm, [add, mul],
48                          system_prompt="Tu es un agent qui doit utiliser les outils pour faire les calculs. Ne fais jamais de calcul
                                mental. Décompose les problèmes complexes en plusieurs appels d'outils.")
49     handler = DebugHandler()
50     print("Agent prêt. Tape 'exit' pour quitter.\n")
51     while True:
52         user_input = input(">>> ")
53         if user_input.lower() in ["exit", "quit"]:
54             print("Fin.")
55             break
56         try:
57             result = agent.invoke(
58                 {"messages": [HumanMessage(content=user_input)]},
59                 config={"callbacks": [handler]},
60             )
61             print("Réponse finale")
62             print(result["messages"][-1].content)
63             print()
64         except Exception as e:
65             print("\nErreur :", e)
```

Exemple ChatBot utilisant des outils avec langchain 2 / 7

Les outils

```
31 @tool
32 def add(a: int, b: int) -> int:
33     """Additionne deux entiers."""
34     print("Appel de la fonction add")
35     return a + b
36
37 @tool
38 def mul(a: int, b: int) -> int:
39     """Multiplie deux entiers."""
40     print("Appel de la fonction mul")
41     return a * b
```

INSA ROUEN NORMANDIE

Exemple ChatBot utilisant des outils avec langchain 3 / 7

Le débugger

```python
class DebugHandler(BaseCallbackHandler):
    def on_chat_model_start(self, donnees_json, messages, **kwargs):
        print("Appel du LLM")
        if messages and isinstance(messages[0], list):
            groupes_messages = messages
        else:
            groupes_messages = [messages]
        for i, groupe_messages in enumerate(groupes_messages):
            print(f" Groupe de messages {i}")
            for message in groupe_messages:
                type_message = getattr(message, "type", message.__class__.__name__)
                contenu_message = getattr(message, "content", str(message))
                print(f"  [{type_message.upper()}] {contenu_message}")
                print(f"  Toutes les informations: {message}")
```

Exemple ChatBot utilisant des outils avec langchain 4 / 7

Le script

```python
import argparse
from langchain.agents import create_agent
from langchain_core.tools import tool
from langchain_core.messages import HumanMessage, BaseMessage
from langchain_core.callbacks import BaseCallbackHandler
from langchain_openai import ChatOpenAI


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Chatbot avec outils")
    parser.add_argument("--modele",
                        required=True,
                        help="Nom du modèle LLM (par exemple 'Qwen/Qwen2.5-1.5B-Instruct' ou 'mistralai/Mistral-Nemo-Instruct-2407')"
    )
    parser.add_argument("--url",
        required=True,
        help="URL du serveur LLM (par exemple 'http://localhost:8100/v1')"
    )
    parser.add_argument(
        "--cle",
        default="XXX",
        help="Clé API (optionnelle)"
    )
    args = parser.parse_args()
    main(args.modele, args.url, args.cle)
```

Exemple ChatBot utilisant des outils avec langchain 5 / 7

Lancement du serveur

```
vllm serve Qwen/Qwen2.5-1.5B-Instruct
    --dtype float32
    --host 0.0.0.0
    --port 8100
    --enable-auto-tool-choice
    --tool-call-parser hermes
```

Exemple ChatBot utilisant des outils avec langchain 6 / 7

Exemple d'utilisation des outils

```
Agent prêt. Tape 'exit' pour quitter.

>>> Calcule 2 plus 5 et multiplie le résultat par 10.
Appel du LLM
 Groupe de messages 0
  [SYSTEM] Tu es un agent qui doit utiliser les outils pour faire les calculs. Ne fais jamais de calcul mental. Décompose les problè
        mes complexes en plusieurs appels d'outils.
  Toutes les informations: content="Tu es un agent qui doit utiliser les outils pour faire les calculs. Ne fais jamais de calcul
        mental. Décompose les problèmes complexes en plusieurs appels d'outils." additional_kwargs={} response_metadata={}
  [HUMAN] Calcule 2 plus 5 et multiplie le résultat par 10.
  Toutes les informations: content='Calcule 2 plus 5 et multiplie le résultat par 10.' additional_kwargs={} response_metadata={} id='
        ae6f5b8d-3906-426a-bc97-29996824a1c1'
Appel d'un outil
  données JSON : {'name': 'add', 'description': 'Additionne deux entiers.'}
  paramètres : {'a': 2, 'b': 5}
Appel de la fonction add
Résultat de l'outil: content='7' name='add' tool_call_id='chatcmpl-tool-9cc36c5e8605c705'
Appel d'un outil
  données JSON : {'name': 'mul', 'description': 'Multiplie deux entiers.'}
  paramètres : {'a': 7, 'b': 10}
Appel de la fonction mul
Résultat de l'outil: content='70' name='mul' tool_call_id='chatcmpl-tool-87c2a09461ba6c72'
```

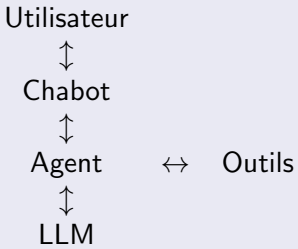
Exemple ChatBot utilisant des outils avec langchain 7 / 7

Exemple d'utilisation des outils (suite et fin)

```
Appel du LLM
 Groupe de messages 0
  [SYSTEM] Tu es un agent qui doit utiliser les outils pour faire les calculs. Ne fais jamais de calcul mental. Décompose les problè
       mes complexes en plusieurs appels d'outils.
  Toutes les informations: content="Tu es un agent qui doit utiliser les outils pour faire les calculs. Ne fais jamais de calcul
       mental. Décompose les problèmes complexes en plusieurs appels d'outils." additional_kwargs={} response_metadata={}
  [HUMAN] Calcule 2 plus 5 et multiplie le résultat par 10.
  Toutes les informations: content='Calcule 2 plus 5 et multiplie le résultat par 10.' additional_kwargs={} response_metadata={} id='
       ae6f5b8d-3906-426a-bc97-29996824a1c1'
  [AI]
  Toutes les informations: content='' additional_kwargs={'refusal': None} response_metadata={'token_usage': {'completion_tokens': 51,
       'prompt_tokens': 280, 'total_tokens': 331, 'completion_tokens_details': None, 'prompt_tokens_details': None}, 'model_provider
       ': 'openai', 'model_name': 'Qwen/Qwen2.5-1.5B-Instruct', 'system_fingerprint': None, 'id': 'chatcmpl-90612df421f9fc8c', '
       finish_reason': 'tool_calls', 'logprobs': None} id='lc_run--019cf6af-87ef-77f0-ac45-8c13f6bccf61-0' tool_calls=[{'name': 'add
       ', 'args': {'a': 2, 'b': 5}, 'id': 'chatcmpl-tool-9cc36c5e8605c705', 'type': 'tool_call'}, {'name': 'mul', 'args': {'a': 7, '
       b': 10}, 'id': 'chatcmpl-tool-87c2a09461ba6c72', 'type': 'tool_call'}] invalid_tool_calls=[] usage_metadata={'input_tokens':
       280, 'output_tokens': 51, 'total_tokens': 331, 'input_token_details': {}, 'output_token_details': {}}
  [TOOL] 7
  Toutes les informations: content='7' name='add' id='63872e9f-085d-49d6-91c6-c355ef7113de' tool_call_id='chatcmpl-tool-9
       cc36c5e8605c705'
  [TOOL] 70
  Toutes les informations: content='70' name='mul' id='2e23f5e5-df54-4c7e-ac3e-3d220433be11' tool_call_id='chatcmpl-tool-87
       c2a09461ba6c72'
Réponse finale
La somme de 2 et 5 est 7. Multipliant ce nombre par 10 donne 70.
```

En résumé

Communications



Risques

- Appels d'outils incorrects ou coûteux
- Chaînes d'actions non déterministes
- Attaques par injection via contenus externes

Vers un besoin de standardisation

Constats

- Les outils sont codés dans l'application
  - ils doivent être écrits dans le programme, puis passés à l'agent
  - pour ajouter un outil, il faut modifier le code de l'agent, redéployer l'aplication et redémarrer le chatbot
- Les outils sont locaux
  - ils ne peuvent pas être partagés entre plusieurs agents
  - une même fonctionnalité risque d'être implémentée plusieurs fois
  - la compétence pour développer un outil peut être externe à l'équipe qui développe l'agent

Solution

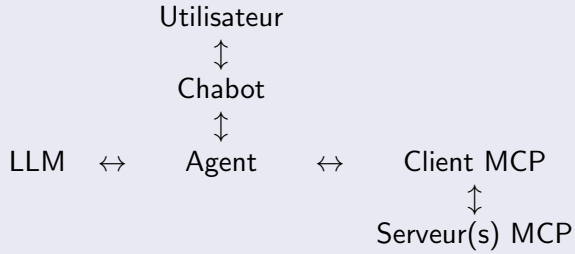
- Découpler les outils de l'agent et les exposer via une interface standardisée

MCP : architecture conceptuelle

Composants

- Un agent qui est en connexion avec un LLM et un client MCP
- Un client MCP qui gère la connexion avec les serveurs MCP
- Des serveurs MCP qui exposent données et outils

Communications



Exemple d'un chatbot utilisant des outils MCP 1 / 5

serveur_mcp.py

```python
import argparse
from fastmcp import FastMCP

serveur = FastMCP("serveur-calcul")

@serveur.tool()
def add(a: int, b: int) -> int:
    """Additionne deux entiers."""
    print("Appel de la fonction add")
    return a + b

@serveur.tool()
def mul(a: int, b: int) -> int:
    """Multiplie deux entiers."""
    print("Appel de la fonction mul")
    return a * b
```

Exemple d'un chatbot utilisant des outils MCP 2 / 5

serveur_mcp.py

```
18 def main(host: str, port: int, transport: str):
19     print("Démarrage du serveur MCP")
20     print(f"Transport : {transport}")
21     print(f"Adresse : {host}:{port}")
22     if transport == "stdio":
23         serveur.run()
24     else:
25         serveur.run(transport=transport,
26                     host=host,
27                     port=port
28                     )
```

INSA ROUEN NORMANDIE

Exemple d'un chatbot utilisant des outils MCP 3 / 5

serveur_mcp.py

```
30 if __name__ == "__main__":
31     parser = argparse.ArgumentParser(description="Serveur MCP de calcul")
32     parser.add_argument("--host",
33                         default="localhost",
34                         help="Adresse d'écoute du serveur MCP (défaut : localhost)"
35     )
36     parser.add_argument("--port",
37                         type=int,
38                         default=8200,
39                         help="Port du serveur MCP (défaut : 8200)"
40     )
41     parser.add_argument("--transport",
42                         default="http",
43                         help="Transport MCP (stdio, http, streamable_http)"
44                         )
45     args = parser.parse_args()
46     main(args.host, args.port, args.transport)
```

Exemple d'un chatbot utilisant des outils MCP 4 / 5

client_mcp.py

```
33 async def recuperer_outils_mcp(url_serveur):
34     client = MultiServerMCPClient(
35         {
36             "calculatrice": {
37                 "url": url_serveur,
38                 "transport": "streamable_http",
39             }
40         }
41     )
42     return await client.get_tools()
```

INSA ROUEN NORMANDIE

Exemple d'un chatbot utilisant des outils MCP 5 / 5

client_mcp.py

```
45 def main(modele: str, url: str, url_mcp: str, cle_api: str = "XXX"):
46     llm = ChatOpenAI(model=modele,
47                      base_url=url,
48                      api_key=cle_api
49                     )
50     outils_mcp = asyncio.run(recuperer_outils_mcp(url_mcp))
51     agent = create_agent(llm,
52                          outils_mcp,
53                          system_prompt="Tu es un agent qui doit utiliser les outils pour faire les calculs. Ne fais jamais de calcul
                                mental. Décompose les problèmes complexes en plusieurs appels d'outils."
54                          )
55     handler = DebugHandler()
56     print("Agent prêt. Tape 'exit' pour quitter.\n")
57     while True:
58         user_input = input(">>> ")
59         if user_input.lower() in ["exit", "quit"]:
60             print("Fin.")
61             break
62         result = asyncio.run(agent.ainvoke({"messages": [HumanMessage(content=user_input)]},
63                                            config={"callbacks": [handler]},
64                                            )
65                              )
66         print("Réponse finale")
67         print(result["messages"][-1].content)
68         print()
```

MCP : ce que cela change

Bénéfices

- Interopérabilité entre modèles et outils
- Réutilisation et mutualisation des connecteurs
- Séparation claire des responsabilités

Limites actuelles

- Écosystème en construction
- Besoin de standards de sécurité et de gouvernance

INSA ROUEN NORMANDIE

Comparer les approches

Grille de lecture

- Données stables ou changeantes
- Besoin de citations et traçabilité
- Besoin d'actions et d'intégration SI
- Coût de mise en place et maintenance

Tableau de synthèse

Comparaison

- Fine-tuning : intégrer des comportements et du style, adapté si le besoin est stable
- RAG : répondre avec des sources à jour, adapté aux connaissances documentaires
- Agents : agir via outils, adapté aux workflows et automatisations
- MCP : standardiser les connexions, adapté à un écosystème d'outils évolutif

Cas d'usage : quel choix

Cas 1 : base documentaire vivante

- Choix : RAG
- Option : Agent pour naviguer, filtrer, résumer

Cas 2 : assistant de rédaction très formaté

- Choix : Fine-tuning
- Option : RAG pour ajouter les faits récents

Cas 3 : automatiser un processus métier

- Choix : Agentification
- Option : MCP pour standardiser les connecteurs

Cas 4 : plateforme multi-outils et multi-modèles

- Choix : MCP
- Option : RAG et agents exposés via des serveurs MCP

Architecture moderne typique

Empilement courant

- RAG pour apporter des faits et des sources
- Agents pour orchestrer des étapes et outils
- MCP pour normaliser l'accès aux ressources

Conclusion

À retenir

- Un LLM généraliste est limité par une connaissance figée et généraliste
- Quatre voies dominent : fine-tuning, RAG, agents, MCP
- Les solutions se combinent souvent plutôt qu'elles ne s'excluent

INSA ROUEN NORMANDIE