Gestion des données et accès à l'information

Chatbots et agents

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

Master Statistique, Apprentissage et Algorithmes
Sorbonne Université

2025-2026

Vocabulaire

Large Language Model

- Modèle de langue
- Complétion de phrase, remplissage de trous

Chatbot / Instruct

- Version adaptée pour le dialogue ou des instructions
- Alignement, RLHF, SFT en fonction de la tâche
- Interactivté

Retrieval Augmented Generation

- Accès à une source de donnée externe
- Pensé à l'origine pour améliorer la génération

Limitations

Pas d'interaction avec l'extérieur

- RAG: tout n'est pas dans le LLM, mais lecture seulement
- Trop rigide, comment assouplir ?

Pas de persistance

- Historique dans la conversation, mais assez limité
- Écriture dans une base de donnée existante
- Persistance des informations données dans la conversation ("Je m'appelle XXX, je travaille à YYY")

Mémoire

Court terme

- Un début: prompt système
- La conversation elle-même
- Ne passe pas d'une conversation à l'autre

Long terme

- Ajouts dans une base de données vectorielles dédiée
- Ajouts dans des fichiers plats

Changement de paradigme

Chatbot

- **Passif**
- texte d'entrée → texte de sortie

RAG

- **Semi-actif**
- Lecture uniquement, et tout le temps: système dédié

Agent

- **Actif**
- Décomposition en sous-tâches
  - Entrée → Analyse
  - Analyse → Action
  - Action → Analyse

Actions ?

Système de fichier

- Lecture-écriture de fichier

Bases de données

- Lecture-écritiure, dans des bases existantes ou dédiées (mémoire)

API

- Tout et n'importe quoi
- Recherche web, envoi d'email, etc

Fonctions dédiées

- Mécanisme pour appeler des fonctions
- Calculatrice

Raisonnement

Toujours des LLM à la base

- Tâche: prédiction du token suivant
- Utilisation d'un outil = tokens pour dire quel outil avec quels arguements

Conséquences

- Tokens de sortie, affichés à l'utilisateurice
- Tokens internes, pour décider quels outils utiliser

Étapes intermédiaires

- *Chain-of-thought → Reasoning*
- Vers la planification

Mécanismes de raisonnement

ReAct (Reasoning + Acting)

- Boucle : *Thought* → *Action* → *Observation*
- Adapation de la génération en fonction des retours des outils.

Chain of Thought (CoT)

- Incitation à “penser étape par étape” pour améliorer la logique déductive.
- Technique de prompting

Self-Reflection / Critique

- Critiquer la sortie, itérer et améliorer
- Agent ou sous-agent

Concrètement

Moteur

- Ollama, vLLM, `transformers`
- Entrée → flux de tokens

Flux de tokens

blablabla {f(toto, tata)} blablabla

- Le moteur analyse
- Puis appele l'outil *f* et injecte le résultat

Outil

- Ce qu'on veut
- Une fonction au sens informatique
- Qui peut faire tout ce qu'on veut
- Liste à configurer dans le moteur

Exemple API OpenAI: définition

```
tools = [ {"type": "function",
        "name": "get_horoscope",
        "description": "Get today's horoscope for an astrologic
        "parameters": {
            "type": "object",
            "properties": {
                "sign": {
                    "type": "string",
                    "description": "An astrological sign like T
                },
            },
            "required": ["sign"],
        },
    }]
```

Exemple API OpenAI: appel

```
def get_horoscope(sign):
    return f"{sign}: Next Tuesday you will befriend a baby otte

input_list = [
    {"role": "user", "content": "What is my horoscope? I am an
]

response = client.responses.create(
    model="gpt-5",
    tools=tools,
    input=input_list,
)
```

Template: exemple GLM

https://huggingface.co/zai-org/GLM-4.5-Air/blob/main/chat_template.jinja

```
You may call one or more functions to assist with the user quer
You are provided with function signatures within <tools></tools
<tools>
{% for tool in tools %}
{{ tool | tojson(ensure_ascii=False) }}
{% endfor %}
</tools>
For each function call, output the function name and arguments
<tool_call>{function-name}
<arg_key>{arg-key-1}</arg_key>
<arg_value>{arg-value-1}</arg_value>
<arg_key>{arg-key-2}</arg_key>
<arg_value>{arg-value-2}</arg_value>
```

Tool parser: exemple vLLM

https://github.com/vllm-project/vllm/blob/main/vllm/tool_parsers/glm47_moe_tool_parser.py

```
class Glm47MoeModelToolParser(Glm4MoeModelToolParser):
    def __init__(self, tokenizer: TokenizerLike):
        super().__init__(tokenizer)
        self.func_detail_regex = re.compile(
            r"<tool_call>(.*?)(<arg_key>.*?)?</tool_call>", re.
        )
        self.func_arg_regex = re.compile(
            r"<arg_key>(.*?)</arg_key>(?:\\n|\s)*<arg_value>(.*
            re.DOTALL,
        )
```

Agents

Mémoire

- Prompt système
- Fichiers
- Base de données vectorielle

Planification

- Décomposition en tâches élémentaires
- Décider quelques outils appeler, sur quelles données

Outils

- Système de fichier
- Accès à la base de données
- Recherche web
- Etc etc

Des agents déjà partout

Plus seulement des chatbots

- chatgpt.com, claude.ai, lechat.mistral.ai, etc
- Grosses évolutions depuis 2023
- Recherche web disponible
- Mémorisation entre les conversations

Assistants de programmation

- Claude Code, Crush, etc
- Accès au système de fichier nécessaire
- Souvent: recherche web, API externes pour chercher la documentation, etc

Les outils

Tout et n'importe quoi

- Bas niveau: fonction exécutable par le moteur du LLM
- Haut niveau: tout ce à quoi le langage de programmation peut accéder

Vraiment tout

- Au plus simple: une fonction Python (ou autre)
- Besoin complexe: une seule fonction ne suffit plus

Besoin de standardisation

- Découpler le moteur du LLM
- Et l'implémentation des outils

Simple fonction

Exemple

- cf API OpenAI plus haut

Intégré dans le moteur

- Manque de souplesse
- Risqué: planter le moteur, accéder à trop de choses

Acceptable

- Pour des choses vraiment simples
- Ou comme passerelle pour accéder à d'autres choses

Appels de fonction

Prendre la décision

- Description de la fonction en langue naturelle
- Intégrée au prompt système
- cf exemple template plus haut

Réaliser l'appel

- Description des arguments
- Leur sens et leur nature (typage)

Programmation

- Fonction classique pas suffisante
- Besoin de metadonnées: description et typage

Schema

Exemple Langchain

```python
from langchain.tools import tool

@tool
def search_database(query: str, limit: int = 10) -> str:
    """Search the customer database for records matching the qu

    Args:
        query: Search terms to look for
        limit: Maximum number of results to return
    """
    return f"Found {limit} results for '{query}'"
```

Schema encore plus structuré

Langchain + Pydantic

```python
class WeatherInput(BaseModel):
    """Input for weather queries."""
    location: str = Field(description="City name or coordinates
    units: Literal["celsius", "fahrenheit"] = Field(
        default="celsius",
        description="Temperature unit preference"
    )
    include_forecast: bool = Field(
        default=False,
        description="Include 5-day forecast"
    )
@tool(args_schema=WeatherInput)
def get_weather(location: str, units: str = "celsius", include_
    """Get current weather and optional forecast."""
```

Hallucinations dans les appels

Aucune garantie

- Mauvais appels, fonctions non-existances, format invalide,
- Rôle de l'entraînement pour pousser à respecter les consignes
- Travail sur le template (et donc sur le prompt système)

Validation

- De l'appel lui-même: structure attendue par le moteur
- Des arguments de la fonction
- Du format de la sortie

Solutions

- Pydantic et autres bibliothèques du genre pour la validation
- Auto-critique pour analyser les erreurs et réessauer
- Interactivté

Plus de souplesses

Fonction

- Liste statique décrite dans le runtime
- Dépend du fournisseur du chatbot ou agent
- Pas de choix pour les utilisateurices

Protocoles

- Bibliothèques d'outils
- Accessibles sur le réseau, ou comme des processus locaux
- Externe au moteur du LLM
- Souvent gérable par les utilisateurices

Agent Skills

```
---
name: pdf-processing
description: Extract PDF text, fill forms, merge files. Use whe
---

# PDF Processing

## When to use this skill
Use this skill when the user needs to work with PDF files...

## How to extract text
1. Use pdfplumber for text extraction...

## How to fill forms
...
```

Description

Frontmatter

- name et description
- utilisé pour savoir quelle SKILL utiliser en fonction du prompt

Reste du fichier

- Description de ce qu'on peut faire
- Et de comment le faire
- Souvent des commandes shell

Fonctionnement

Initialisation

- Chargement des frontmatter uniquement
- Pour ne pas surcharger le contexte

En génération

- Une collection de skills disponibles
- Choix d'une skill pertinente
- Chargement du fichier SKILL.md correspondant

Dépendances

- Accès en lecture à des fichiers
- Exécution de commandes shell

Exemple

https://raw.githubusercontent.com/anthropics/skills/refs/heads/main/skills/docx/SKILL.md

```
name: docx
description: "Use this skill whenever the user wants to create,

# DOCX creation, editing, and analysis

A .docx file is a ZIP archive containing XML files.

| Task | Approach |
|------|----------|
| Read/analyze content | `pandoc` or unpack for raw XML |
| Create new document | Use `docx-js` - see Creating New Docume
| Edit existing document | Unpack → edit XML → repack - see Edi
```

Exemple (suite)

````
### Converting .doc to .docx

Legacy `.doc` files must be converted before editing:

```bash
python scripts/office/soffice.py --headless --convert-
to docx document.doc
```

### Reading Content

```bash
# Text extraction with tracked changes
pandoc --track-changes=all document.docx -o output.md
````
````

MCP - Model Context Protocol

Standard ouvert

- Pour décrire les outils disponibles
- Pour décrire les entrées (type, commentaire)
- Pour décrire les sorties

Serveurs MCP

- Accessible sur le réseau (publics ou privés)
- En local, un processus externe auquel parle l'agent

Intérêt

- Standardisation des descriptions
- Découplage entre le moteur LLM et les outils
- Plus besoin d'utiliser le même langage de programmation
- Donne plus de choix aux utilisateurices

Architecture client-serveur

Client

- Assistants de code: Claude Code, Crush, etc
- IDE: VSCode et cie
- Interfaces de chat

Serveur

- Processus externe
- Sur le réseau ou en local
- Gère tout ce que le LLM ne peut pas faire directement

Capacités du MCP

Ressources

- Données statiques ou dynamiques exposées par le serveur (contenu d'un fichier, une ligne dans une base de données)
- Lecture seule

Outils

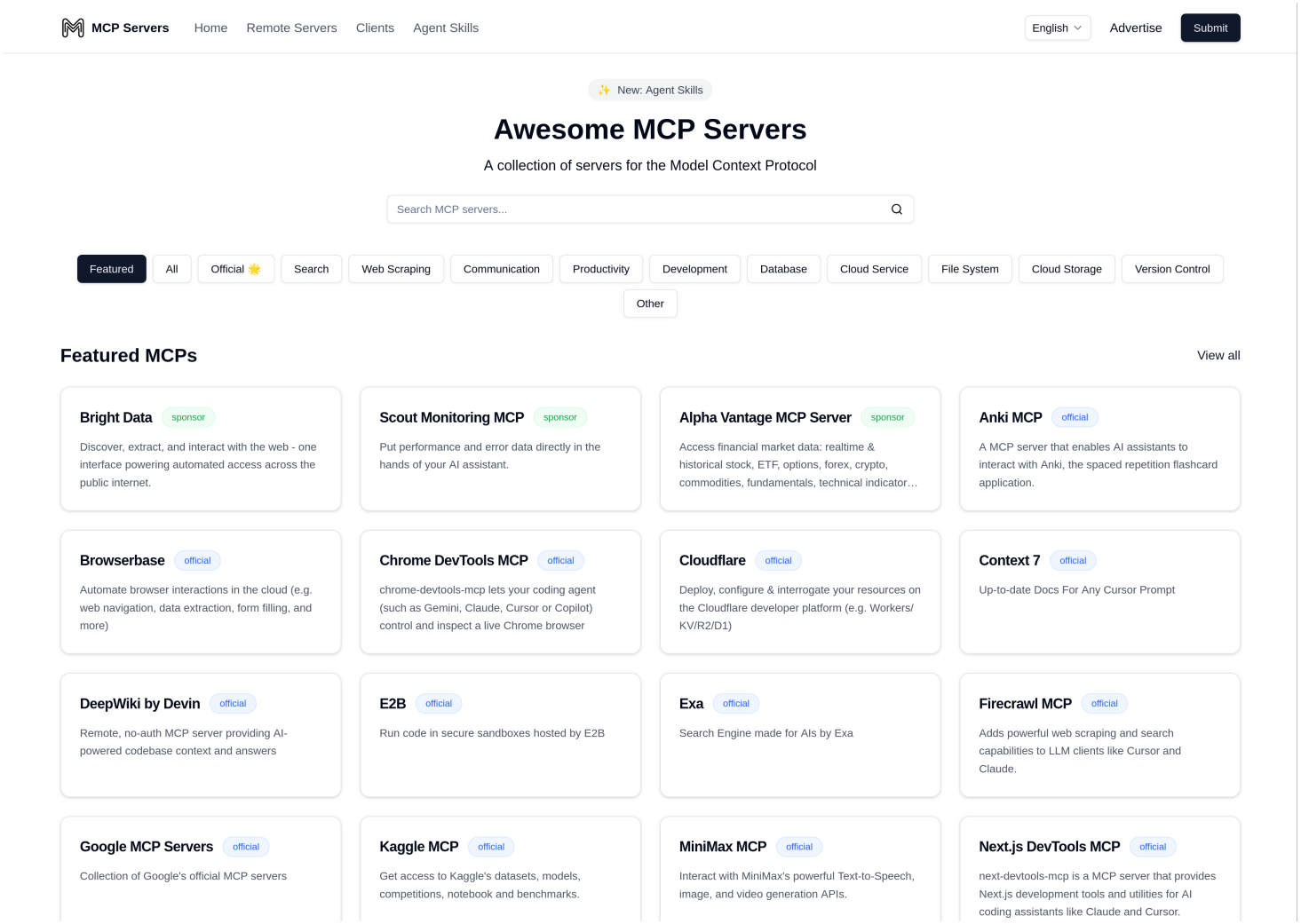
- Fonctions exécutables
- Peuvent modifier l'état du système

Prompts

- Templates de prompts pré-rédigés

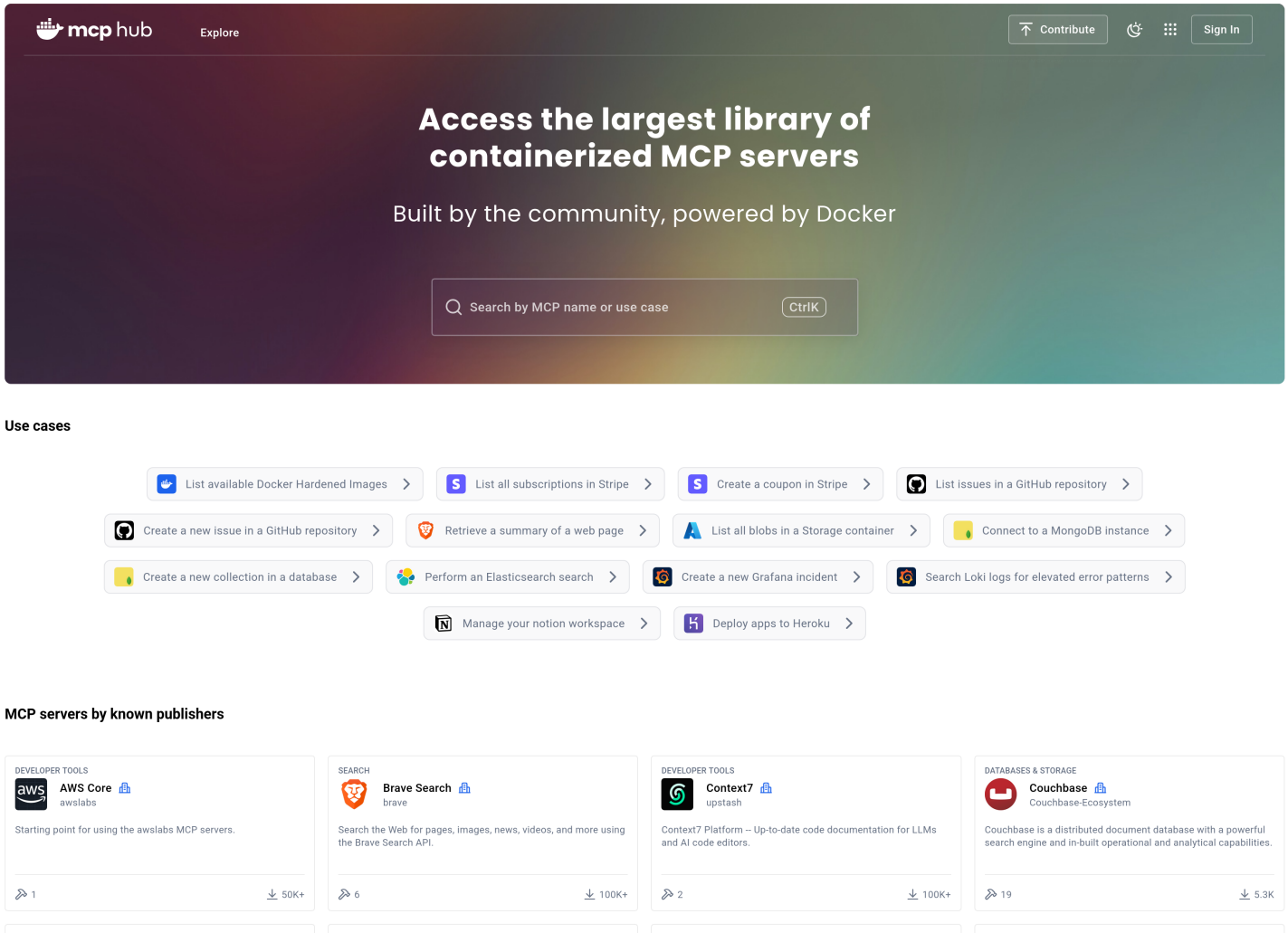
Écosystème MCP

https://mcpservers.org



Écosystème MCP

https://hub.docker.com/mcp



Avantages

Développeur

- Écriture facile d'interface de serveurs
- Utilisable avec quasiment tous les agents

Agents

- Large possiblités d'accès externes
- Ne dépend pas des dévelopeurs de l'agent

Utilisateurices

- Large catalogue en fonction des besoins

Retour vers le plus simple

OpenAPI

- (attention au P !)
- Standard pour décrire des services web (donc API http)
- Documentation (pour les humains, mais utilisables par des LLM)

Intérêt pour les agents

- Déjà documenté, avec spécification complète des entrées sorties
- Déjà accessible sur le réseau
- Facile à implémenter

Au final

- Pas très différent de MCP
- Mais plus primitif pour l'usage IA Agentique

Plusieurs agents

Besoin d'agents collaboratifs

- Construire une application analytique complexe
  - Agent A : Récupère les données (Data Engineer)
  - Agent B : Analyse les données (Data Scientist)
  - Agent C : Génère le rapport (Writer)
- Pas forcément le même LLM (pour des raisons de coût)
- Pas forcément les mêmes accès (pour des raisons de sécurité)

Protocole standardisé

- Échange d'informations entre les agents
- Besoin de métadonnées (description de ce qu'on fait, schema d'entrée et de sortie)

ACP - Agent Communication Protocol (ACP)

Standard pour les messages inter-agents

- Sémantique des messages (Request, Inform, Refuse).
- Métadonnées de routage (Qui envoie, à qui, quel ID de conversation).
- Encapsulation des payloads (JSON, Texte, Données binaires).

Orchestration multi-agents

Superviseur

- Agent en chef qui interagit avec l'utilisateur
- Délégation dynamique à des sous-agents

Hiérarchie

- Structure en arbre
- Plusse de niveaux de délégation

Séquentiel

- Chaîne de production simple
- Sortie d'un agent = entrée du suivant

Graphes complexes

- Boucles de traitement complexe
- Conditions dynamiques

Conclusion

Déjà des agents partout

- Chatbots modernes
- Assistants de code

Nouveau schéma de pensée

- Avant: exécution d'actions puis construction d'un prompt
- Agents: intégration des actions dans la génération

Inconvénients et risques

- Contextes énormes: coûteux en temps et en mémoire
- Ne pas oublier l'évaluation
- Failles de sécurités

DANGER DANGER DANGER DANGER

- Peu de recul coté cyber-sécurité, peu de solutions