

Large Language Models, Agentic Design Patterns and beyond

Julien Perez

Enseignant-Chercheur, Statistical Machine Learning, EPITA

Directeur, IA et Pédagogie, IONIS Education Group

October 15, 2025



Enseignant-Chercheur, HDR

EPITA : École d'ingénieurs en informatique

IA, apprentissage automatique et programmation différentiable

Directeur de la recherche

Groupe IONIS Education

Centre de recherche en IA pour la pédagogie

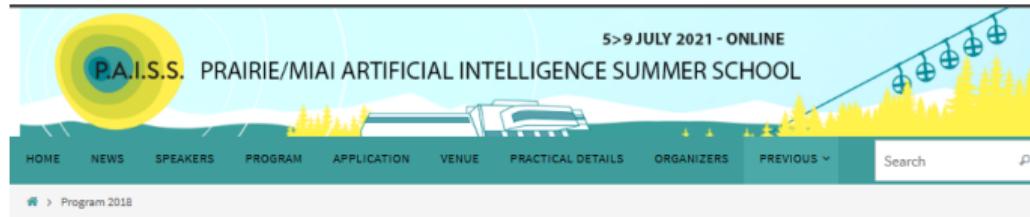
Recherche

Alignment des modèles génératifs

Focalisation sur les CodeLLMs

Applications à l'éducation et à la pédagogie

Hello



Lecture slides

Léon Bottou: Artificial Intelligence – Unsupervised Learning and Causation ([pdf](#))

Kyunghyun Cho: Neural Machine Translation ([pptx](#))

Hugo Larochelle: Generalizing from few examples with meta-learning ([pdf](#))

Diane Larlus, Visual Search ([pdf](#))

Yann LeCun, Deep Learning: Past, Present and Future ([pdf](#))

Nicolas Mansard, Motion generation for agile robots ([pdf](#))

Julien Mairal, Large-Scale Optimization for Machine Learning ([pdf](#))

Remi Munos, Off-policy deep RL ([pdf](#))

Julien Perez, Machine Reading, Models and Applications ([pdf](#))

Jean Ponce, Weakly Supervised and Unsupervised Methods for Image and Video Interpretation ([pptx](#))

Cordelia Schmid, Action recognition ([part 1](#), [part 2](#))

Andrew Zisserman, Self-Supervised Learning ([pdf](#))

Other lecture slides will be made available shortly.



Hello

Multi Documents Answering

Wikipedia ⊕ Predict Clear

Matched documents

Blade Runner (franchise)	657.27
Blade Runner	657.27
Blade Runner 2049	609.13
Replicant	609.13
Blade Runner (1997 video game)	543.61

Blade Runner

Blade Runner is a 1982 American neo-noir science fiction film directed by Ridley Scott, written by Hampton Fancher and David Peoples, and starring Harrison Ford, Rutger Hauer, Sean Young, and Edward James Olmos. It is a loose adaptation of Philip K. Dick's novel "Do Androids Dream of Electric Sheep?" (1968). The film is set in a dystopian future Los Angeles of 2019, in which synthetic humans known as replicants are bioengineered by the powerful Tyrell Corporation to work on off-world colonies. When a fugitive group of replicants led by Roy Batty (Hauer) escapes back to Earth, burnt-out cop Rick Deckard (Ford) reluctantly agrees to hunt them down.

* Blade Runner "initially underperformed in North American theaters and polarized critics; some praised its thematic complexity and visuals, while others were displeased with its unconventional pacing and plot. It later became an acclaimed cult film regarded as one of the all-time best science fiction movies. Hailed for its production design depicting a" retrofitted "future," Blade Runner "is a leading example of neo-noir cinema. The soundtrack, composed by Vangelis, was nominated in 1983 for a BAFTA and a Golden Globe as best original score.

The film has influenced many science fiction films, video games, anime, and television series. It brought the work of Philip K. Dick to the attention of Hollywood, and several later big-budget films were based on his work. In the year after its release, "Blade Runner" won the Hugo Award for Best Dramatic Presentation, and in 1993 it was selected for preservation in

LABS

Machine Reading Project - NAVER LABS Europe

Figure: PAISS 2021 - Machine Reading, models and applications

Plan

- 1 Introduction
- 2 Modèles de langue et IA Générative
- 3 Design Pattern Réflexion
- 4 Pattern Planification
- 5 Pattern Multi-Agent
- 6 Conclusion

Définition formelle

- Un modèle génératif cherche à estimer la distribution $P(X)$ ou $P(X | Y)$ pour produire de nouveaux exemples.
- En langage, on se limite souvent à l'auto-régression :

$$P(x_1, \dots, x_n) = \prod_{t=1}^n P(x_t | x_{<t}).$$

- Apprentissage sur vastes corpus (textes, images, sons).¹
- **Exemple** : Un modèle comme GPT, entraîné sur des milliards de phrases, génère la suite probable d'un texte, e.g., Il fait beau aujourd'hui, donc... → je vais me promener.

¹Goodfellow I., et al., *Generative Adversarial Networks*, NeurIPS, 2014.

Construction d'un LLM

- ➊ Collecte/filtrage de centaines de milliards de tokens.
- ➋ Pré-entraînement auto-régressif : minimiser la perte de cross-entropy.
- ➌ Post-entraîné supervisé sur dialogues humains.
- ➍ Post-entraîné supervisé sur des préférences humaines.
- ➎ Constitutional AI : règles éthiques et auto-critique.

Objectif LM

$$\mathcal{L} = - \sum_t \log P_\theta(x_t | x_{<t})$$

Perspectives de développement

Lois d'échelle

$$\text{Loss} \propto N^{-a} + D^{-b} + C^{-c}$$

- Performance suit une loi de puissance selon le nombre de paramètres (N), données (D), ressources de calcul (C).^a
- Pour l'école : privilégier modèles intermédiaires ou ouverts pour optimiser efficacité et accessibilité.

^aKaplan J., et al., "Scaling Laws for Neural Language Models," arXiv:2001.08361, 2020.

Empreinte carbone

- Pré-entraînement GPT-3 : ~ 3 GWh.^a
- Requête ChatGPT : 0,17–1,7 Wh selon la longueur du prompt.
- Recommandation : usage frugal et solutions libres pour minimiser l'impact

Pipeline typique d'une requête

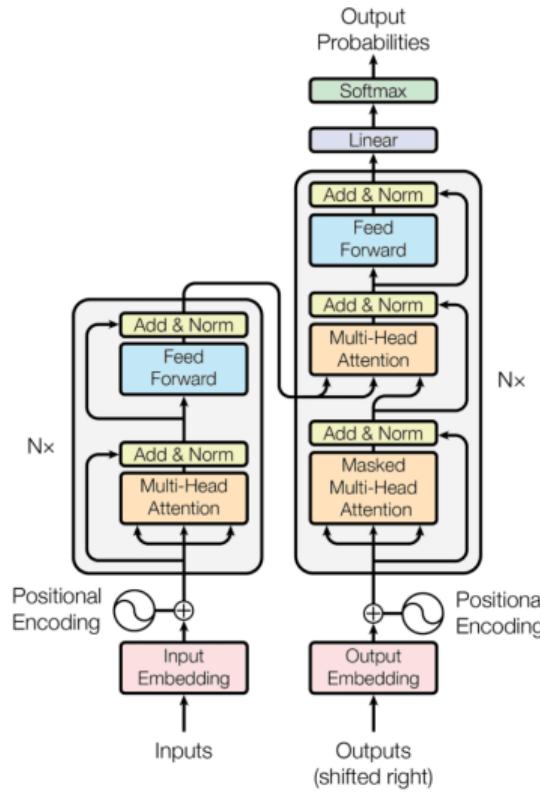
- ① **Tokenisation** : découpage et encodage en jetons.
- ② **Embedding** : projection numérique des jetons.
- ③ **Modélisation** : passage dans le réseau (Transformer).
- ④ **Projection softmax** : distribution sur le vocabulaire.
- ⑤ **Échantillonnage** : argmax / top-k / nucleus sampling.²

²Holtzman A., et al., "The Curious Case of Neural Text Degeneration," ICLR, 2020.

Architecture Transformer simplifiée

- Blocs empilés :
 - Self-attention multi-tête,
 - Feed-forward positionnel.
- Positional encodings pour l'ordre des jetons.
- Fort parallélisme : efficacité sur GPU/TPU.^a

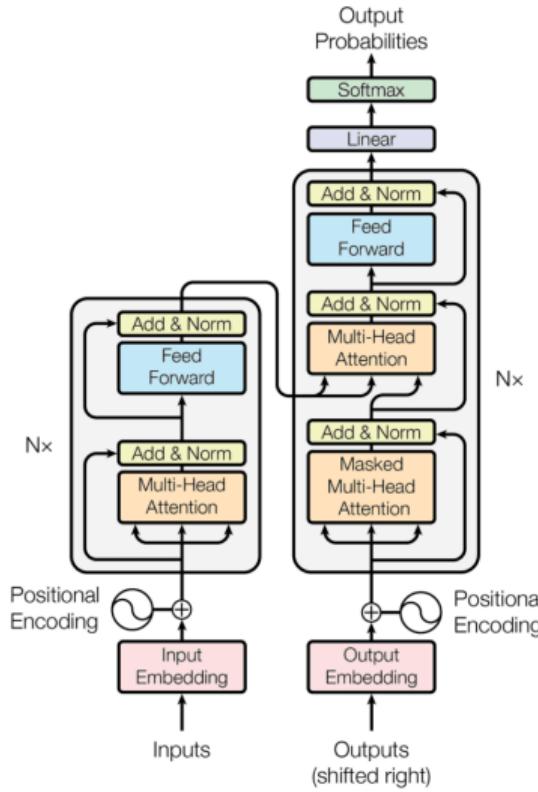
^aVaswani A., et al., "Attention Is All You Need," NeurIPS, 2017.



Phase d'inférence

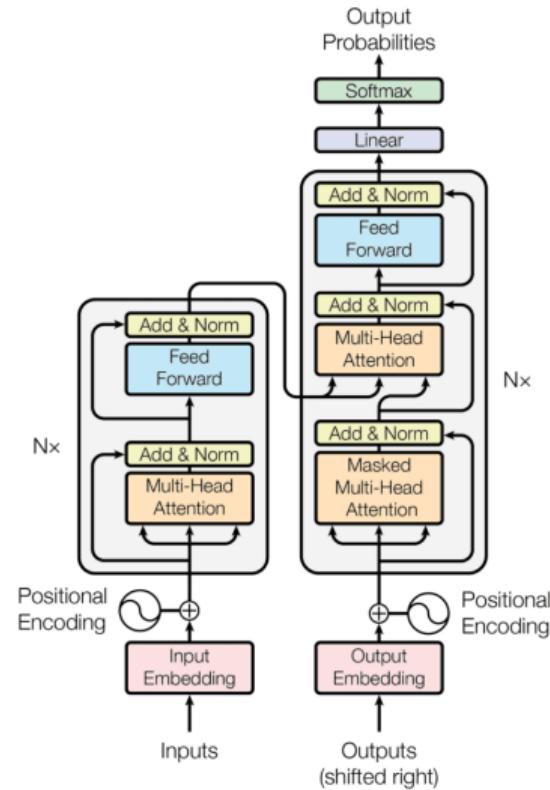
- ① Saisie du *prompt*, une requête en langage naturel.
- ② Modélisation auto-régressive : on génère x_{t+1} à partir de $x_{\leq t}$.
- ③ Boucle jusqu'à la longueur désirée ou un jeton de fin.^a

^aRadford A., et al., "Language Models are Unsupervised Multitask Learners," OpenAI, 2019.



Avancées récentes dans les modèles de langage

- **Modèles à grande échelle** : milliards de paramètres (GPT-4, Gemini, Claude, etc.).
- **Techniques d'entraînement améliorées** :
 - Pré-entraînement multi-modal (texte, image, audio),
 - Alignement par renforcement (RLHF, DPO),
 - Fine-tuning sur tâches spécialisées.
- **Efficacité et déploiement** :
 - Quantization, distillation, modèles experts,
 - Agents et mémoire contextuelle longue (1M+ tokens).
- **Tendances actuelles** : modèles open-source performants (Llama 3, Mistral, Qwen2).



La prédiction du mot suivant comme objectif fondamental

- L'objectif d'apprentissage d'un modèle de langage est défini comme :

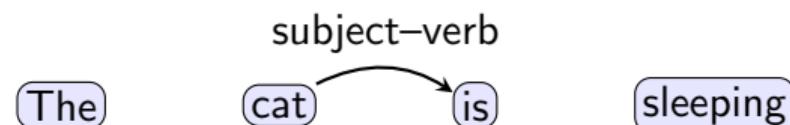
$$\min_{\theta} \mathcal{L}(\theta) = - \sum_{t=1}^n \log P_{\theta}(x_t | x_{<t})$$

- Cet objectif force le modèle à approximer la vraie distribution linguistique $P^*(x_t | x_{<t})$.
- En pratique, la minimisation de \mathcal{L} implique la capture :
 - ① des régularités grammaticales,
 - ② des relations sémantiques entre concepts,
 - ③ et des dépendances causales implicites du monde représenté dans le langage.
- Ainsi, l'optimisation de la vraisemblance conditionnelle devient un **apprentissage du monde à travers le langage**.

Régularités grammaticales apprises par le modèle

Exemples :

- The cat is sleeping.
- The dogs are barking.



Le modèle apprend :

- la dépendance entre le **nombre du sujet** et la forme du verbe,
- des structures syntaxiques fréquentes,
- des contraintes de position (ordre des mots).

Dépendance syntaxique capturée automatiquement.

Relations sémantiques entre concepts

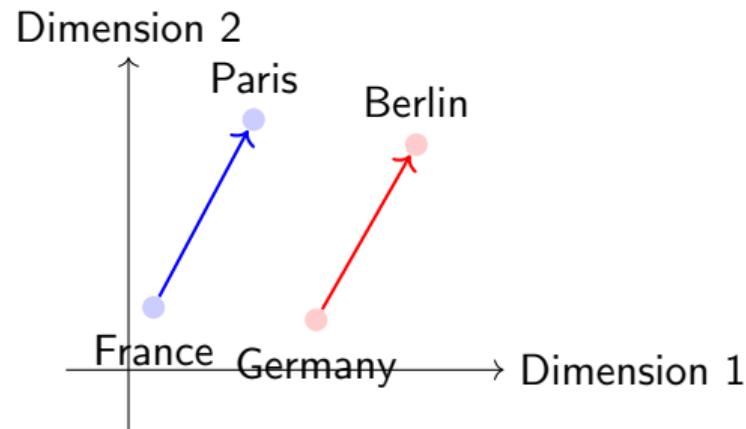
Exemples d'entraînement :

- Paris is the capital of France.
- Berlin is the capital of Germany.

Le modèle apprend :

- des relations de type (**ville, capitale, pays**),
- des analogies vectorielles :

$$\text{Paris} - \text{France} \approx \text{Berlin} - \text{Germany}$$



Espaces d'embeddings : structure sémantique géométrique.

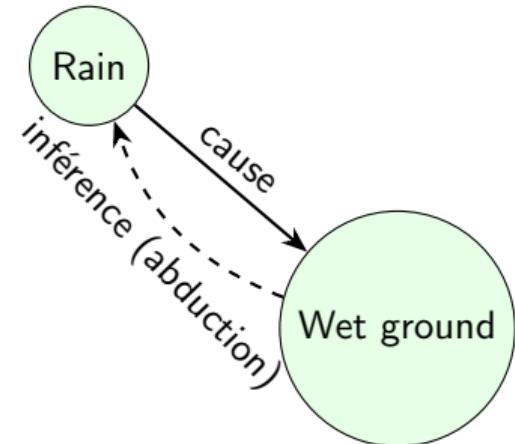
Dépendances causales implicites

Exemples d'entraînement :

- If it rains, the ground gets wet.
- The ground is wet, so it must have rained.

Le modèle apprend :

- la cooccurrence entre événements,
- une direction causale implicite,
- des régularités du monde exprimées par le langage.



Représentation causale implicite extraite du langage.

Principe

Le langage encode une projection compacte du monde physique, social et conceptuel :

$$X = f(W) + \varepsilon$$

où W désigne l'état latent du monde, et f une fonction linguistique de description.

- Apprendre $P(X)$ revient à approximer $P(W)$, c'est-à-dire à modéliser les régularités du monde réel.
- Le modèle développe donc implicitement une représentation latente Z_θ telle que :

$$Z_\theta \approx \mathbb{E}[W | X_{<t}]$$

- Cette représentation encode le **sens commun**, les relations causales et les structures conceptuelles nécessaires à la prédiction linguistique.

Raisonnement comme inférence probabiliste I

Cadre probabiliste

À chaque étape t , le modèle estime :

$$P_\theta(x_t \mid x_{<t}) \propto \exp(h_\theta(x_{<t}) \cdot e(x_t))$$

où $h_\theta(x_{<t})$ est une représentation contextuelle, et $e(x_t)$ un embedding lexical.

- La cohérence temporelle et logique émerge de la contrainte d'auto-cohérence des séquences.

Raisonnement comme inférence probabiliste II

- Le modèle apprend à approximer une distribution jointe :

$$P(x_1, \dots, x_n) = \prod_{t=1}^n P(x_t | x_{<t})$$

ce qui impose une **consistance logique** entre les prédictions successives.

- Le raisonnement est alors une inférence probabiliste dans l'espace latent Z_θ .

Le sens commun comme régularité de haute entropie

- Le modèle maximise la vraisemblance des séquences naturelles, donc minimise l'entropie conditionnelle :

$$H(X_t | X_{<t}) = - \sum_x P(x_t, x_{<t}) \log P(x_t | x_{<t})$$

- Les séquences conformes au sens commun — c'est-à-dire cohérentes causalement et pragmatiquement — réduisent cette entropie.
- Le sens commun émerge ainsi comme un **invariant de compression** :

“raisonner” \Leftrightarrow réduire l'incertitude sur les séquences plausibles.

- Cette perspective relie la modélisation linguistique à la théorie de l'information et à la cognition bayésienne.

Émergence des capacités de raisonnement

- La prédiction du mot suivant requiert :
 - **Raisonnement inductif** — généralisation à partir de contextes observés,
 - **Raisonnement abductif** — inférence de causes plausibles,
 - **Raisonnement déductif** — maintien de cohérence logique.
- Ces formes d'inférence émergent comme contraintes d'optimisation sur $P(x_t | x_{<t})$.
- Le modèle se comporte comme un système d'inférence amortie sur des structures symboliques latentes.

Thèse centrale

L'apprentissage auto-régressif sur le langage est une forme de **compression informationnelle du monde** :

$$\min_{\theta} \mathcal{L}(\theta) \Rightarrow Z_{\theta} \text{ encode les structures causales et sémantiques du monde.}$$

- La prédiction du mot suivant n'est pas une tâche superficielle :
 - elle induit la construction de représentations latentes cohérentes ;
 - elle permet l'émergence du raisonnement et du sens commun ;
 - elle unifie langage, inférence et cognition sous un même objectif probabiliste.

Limites : hallucinations et biais

Hallucinations factuelles

- Le modèle peut inventer des faits plausibles mais inexacts.
- Les citations et sources sont souvent non vérifiables.
- Les réponses doivent toujours être recoupées.^a

^aJi Z., et al., "Survey of Hallucination in Natural Language Generation," arXiv:2202.03629, 2022.

Biais et stéréotypes

- Les données d'entraînement reflètent parfois des stéréotypes sociaux.
- Risque d'amplification des biais sexistes, raciaux, culturels.
- Importance des filtres éthiques et du post-traitement.^a

^aBender E. M., et al., "On the Dangers of Stochastic Parrots," FAccT, 2021.

De la prédiction au raisonnement contrôlé

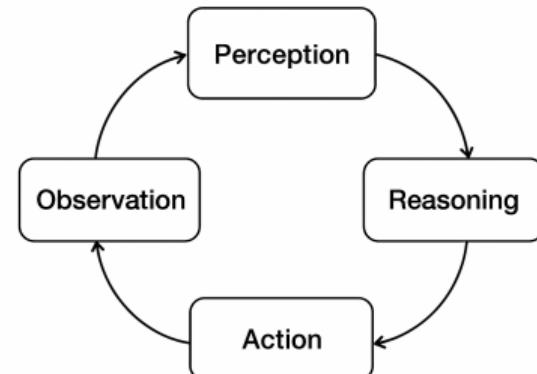
- Les LLMs excellent dans la **génération fluide et cohérente**, mais leur objectif d'apprentissage — la prédiction du mot suivant — ne garantit pas la **vérité factuelle**.
- L'enjeu n'est plus seulement de produire du texte plausible, mais de construire un **comportement raisonné et vérifiable**.
- Cette transition appelle un changement de paradigme : passer de modèles *statiques* à des **systèmes agentiques** capables d'agir, de planifier et de se corriger.

Problématique

Comment transformer un modèle de langage génératif en un agent capable de contrôler, d'expliquer et de justifier ses propres productions ?

Vers une approche agentique

- L'agentification vise à doter les LLMs de **boucles de raisonnement et d'action**.
- Elle combine trois dimensions :
 - ① **Réflexion** : introspection et auto-correction du raisonnement.
 - ② **Interaction** : usage d'outils externes pour valider les faits.
 - ③ **Coordination** : planification ou coopération entre agents spécialisés.
- Cette approche transforme les modèles en **systèmes actifs**, capables de gérer la véracité, la mémoire et la complexité des tâches.



Boucle perception–raisonnement–action

Des limites des LLMs aux Agentic Design Patterns

- Les **hallucinations** et biais révèlent les failles du raisonnement implicite des modèles.
- Pour surmonter ces limites, on introduit des **modèles d'organisation de comportements** — les **Agentic Design Patterns**.
- Ces patterns fournissent une grammaire de conception pour :
 - ① Structurer le raisonnement (Réflexion),
 - ② Vérifier les faits et exécuter des actions (Tools),
 - ③ Planifier et décomposer les tâches (Planification),
 - ④ Coopérer entre agents (Multi-Agent).

Transition

De la génération linguistique à la **raison actionnable** : c'est tout l'enjeu des Agentic Design Patterns.

Introduction aux Agentic Design Patterns

- Les **Agentic Design Patterns** sont des méthodes pour structurer et optimiser les comportements intelligents que l'on souhaite construire à partir de modèles de langage.
- Ils contribuent à résoudre des problèmes complexes en contribuant à la véracité des productions, en divisant les tâches, en utilisant des outils externes.
- 4 patterns principaux :
 - 1 Réflexion
 - 2 Tools
 - 3 Planification
 - 4 Multi-Agent

- **Hallucinations** : Les modèles génèrent parfois des informations fausses ou non factuelles.
- **Mémoire limitée** : Les modèles ne peuvent pas retenir toutes les informations nécessaires.
- **Complexité des tâches** : Les tâches nécessitent souvent plusieurs étapes ou compétences spécifiques.
- **Besoin de flexibilité** : Les agents doivent s'adapter à des contextes variés et dynamiques.

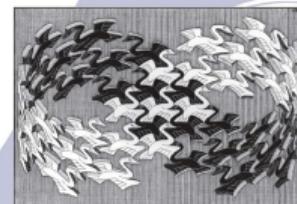
Comment les Patterns répondent à ces limitations

- **Réflexion** : Permet de valider des tâches complexes par un mécanisme de vérification.
- **Tools** : Compense les limitations des modèles en utilisant des outils externes.
- **Planification** : Structure les tâches pour une gestion efficace.
- **Multi-Agent** : Exploite la collaboration entre agents spécialisés.

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

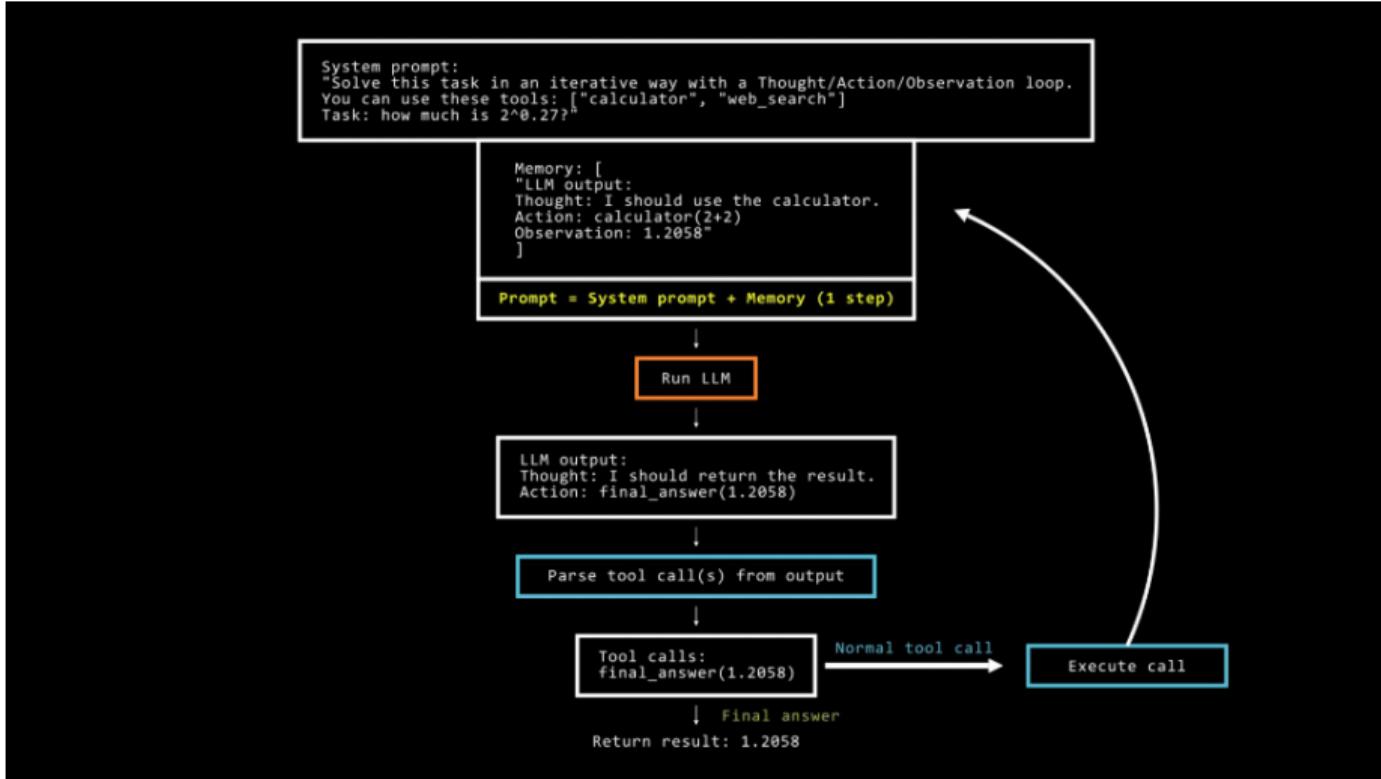


Cover art © 1994 M.C. Escher / Corbis Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

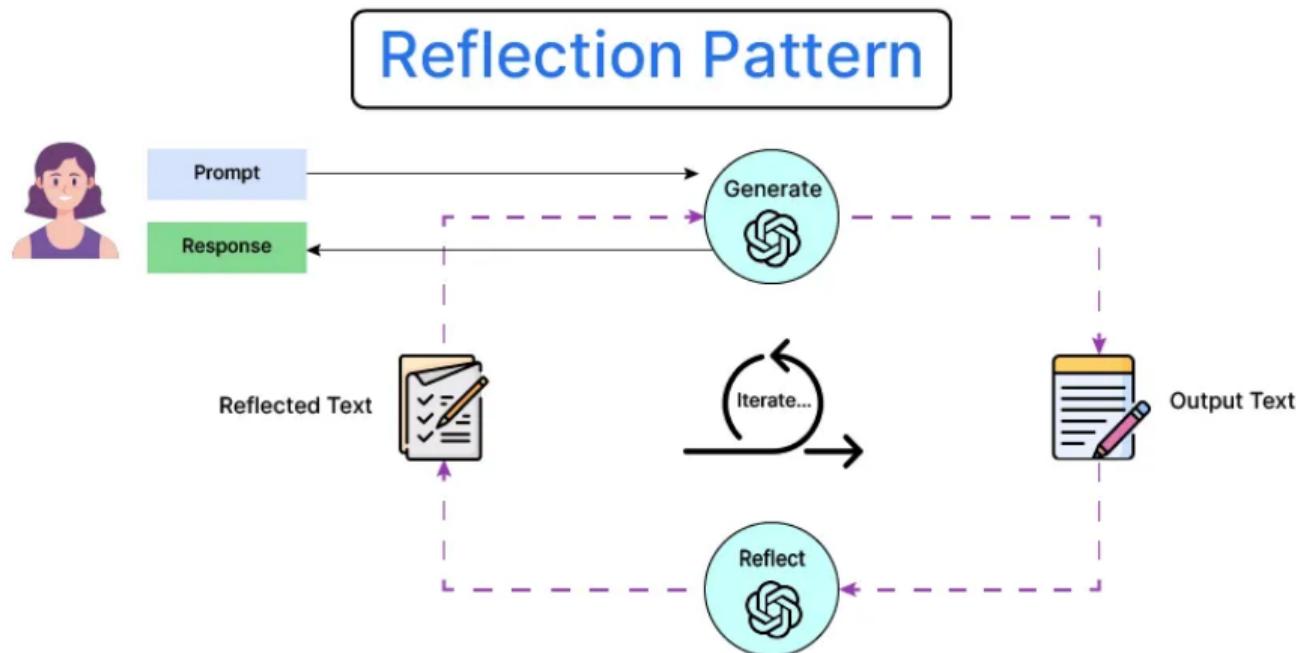


Agentification



- **Définition** : Permet à un agent de diviser une tâche complexe en sous-réflexions successives.
- **Utilité** : Améliore la précision et la profondeur des réponses en segmentant le raisonnement. Meilleure compréhension des tâches complexes.
- **Exemple** : Résumer un texte en extrayant les idées clés avant de formuler un résumé final. Résolution de problèmes nécessitant plusieurs niveaux d'analyse.

Design Pattern Reflexion - Principe



Réflexion, exemple de Prompt

Objectif

Inciter l'agent à évaluer son propre raisonnement avant de proposer une solution définitive.

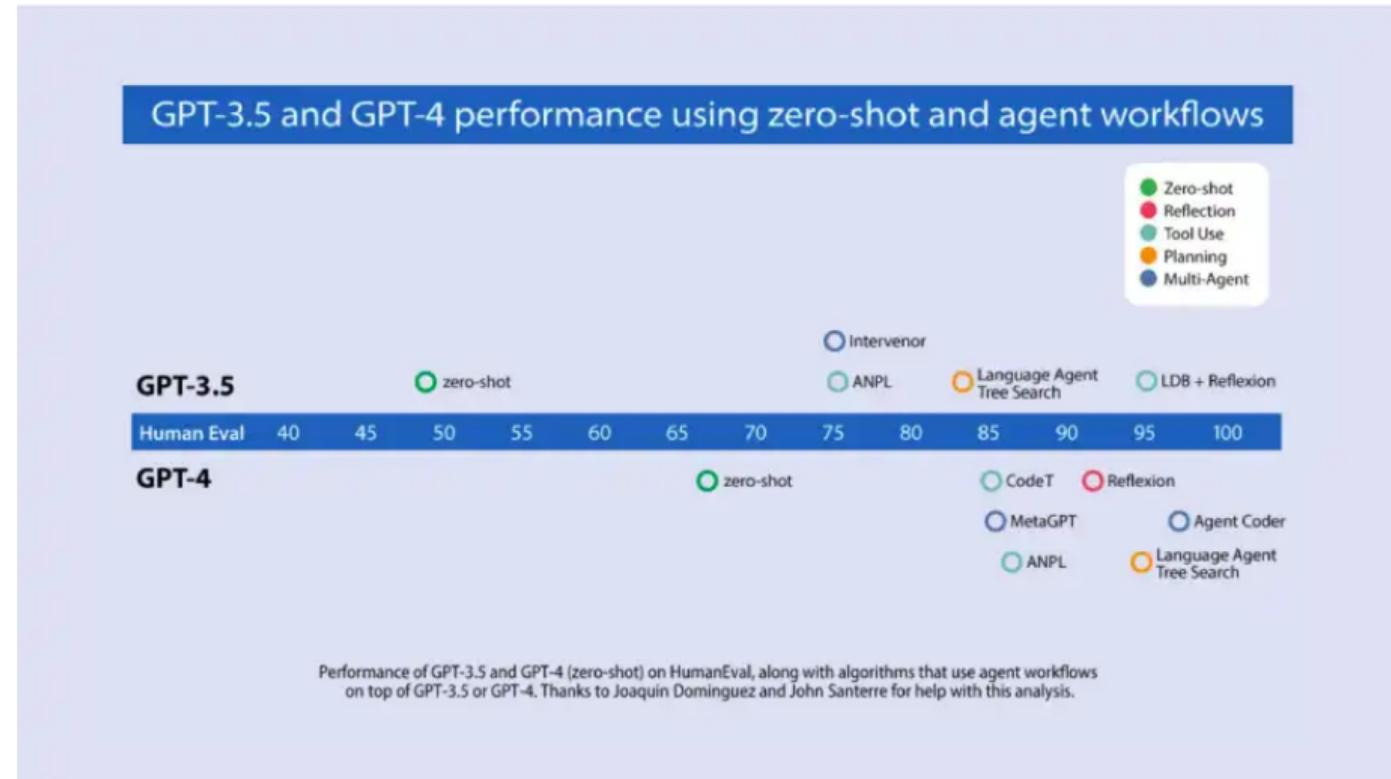
Prompt :

Finalise cette réponse, prends un moment pour réfléchir aux étapes précédentes. Y a-t-il des points où ton raisonnement pourrait être optimisé ou corrigé ? Prends en compte les résultats intermédiaires et toute incohérence potentielle.

Utilisation

Ce type de prompt aide l'agent à **revisiter** ses actions et à ajuster sa stratégie avant de rendre un verdict final, augmentant ainsi la précision et la qualité des réponses.

Design Pattern Reflexion - Bénéfices

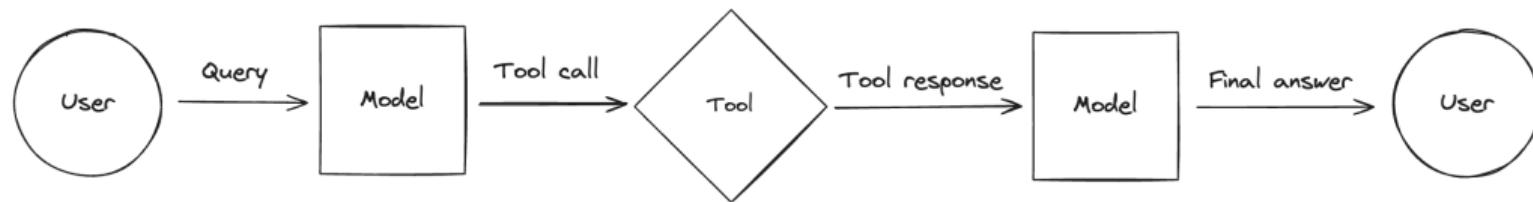


Exemples d'avantages concrets

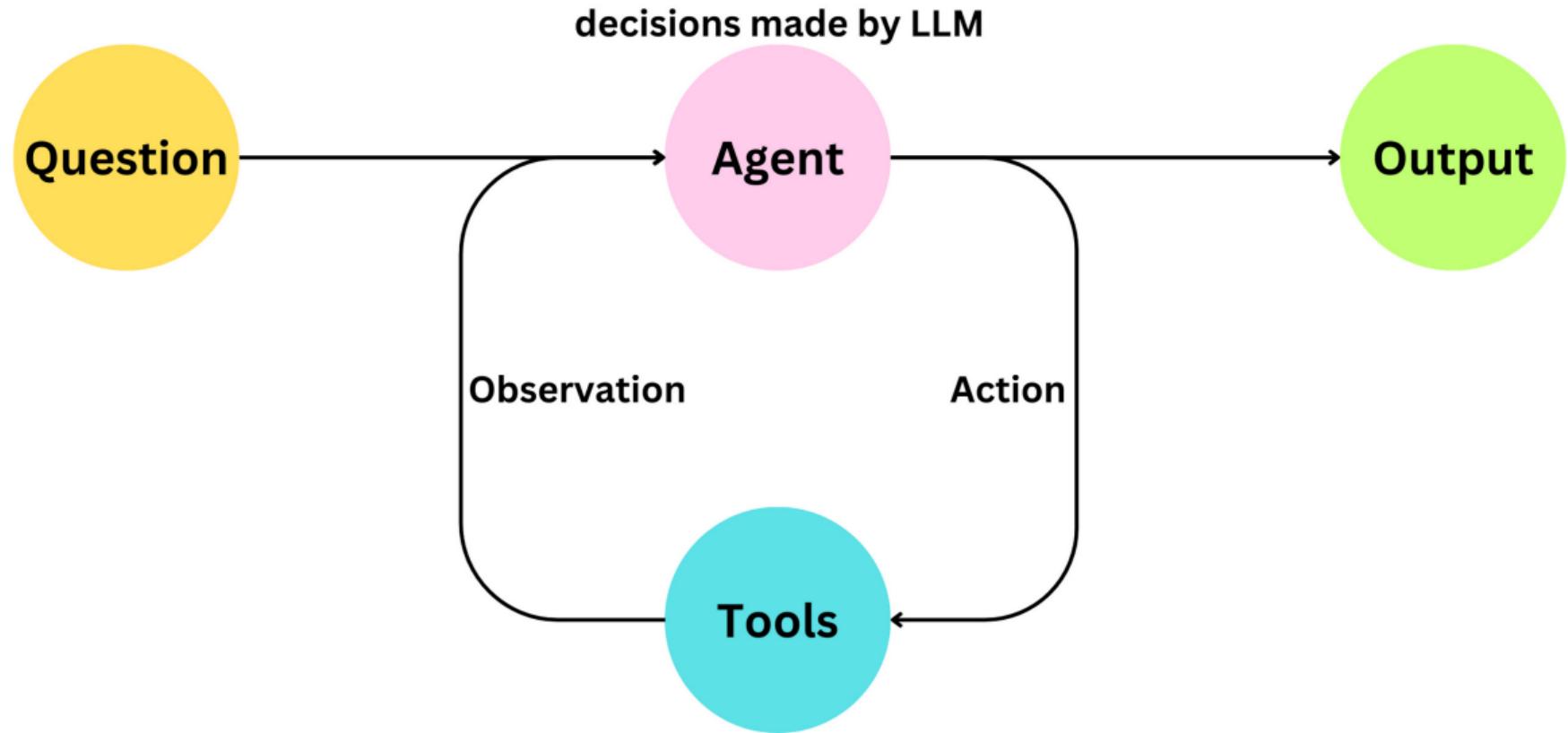
- **Applications en IA générative** : Lors de la génération de textes complexes, comme la synthèse d'articles scientifiques, ce pattern permet à l'IA de structurer ses idées de manière plus logique.
- **Analyse de documents juridiques** : Le pattern Réflexion est utilisé pour analyser des documents longs et techniques, en fournissant des réflexions successives pour valider les interprétations juridiques.
- **Support à la prise de décision** : En entreprise, un agent utilisant ce pattern peut générer des réflexions sur différentes options stratégiques avant de recommander la meilleure solution.

- **Définition** : Permet à un agent d'utiliser des outils externes pour accomplir des tâches.
- **Utilité** : Compense les limitations des modèles en accédant à des informations ou compétences externes.
- **Exemple** : Utiliser une API de recherche pour valider des faits avant de répondre à une question.

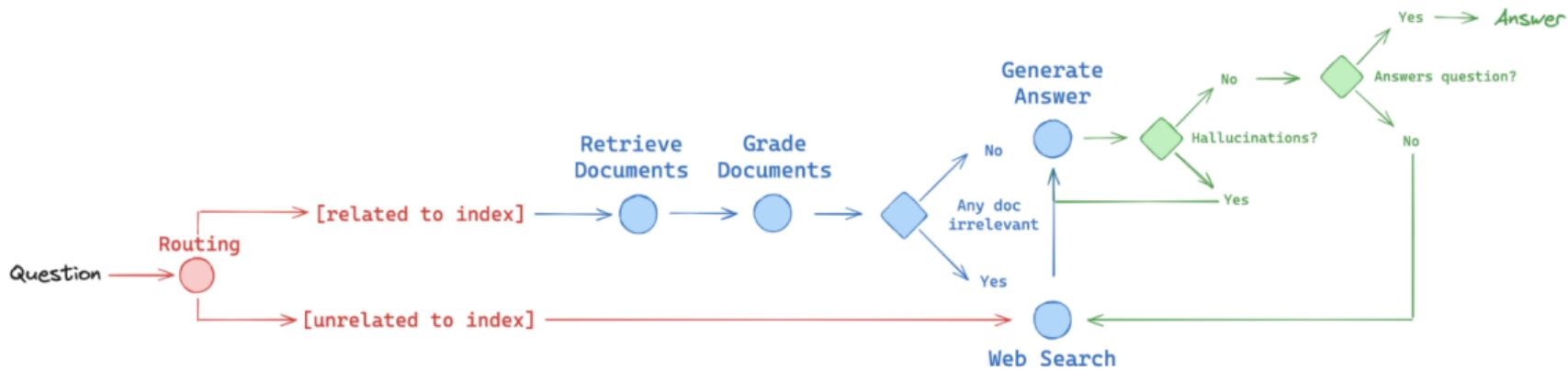
Design Pattern Tool - Principe



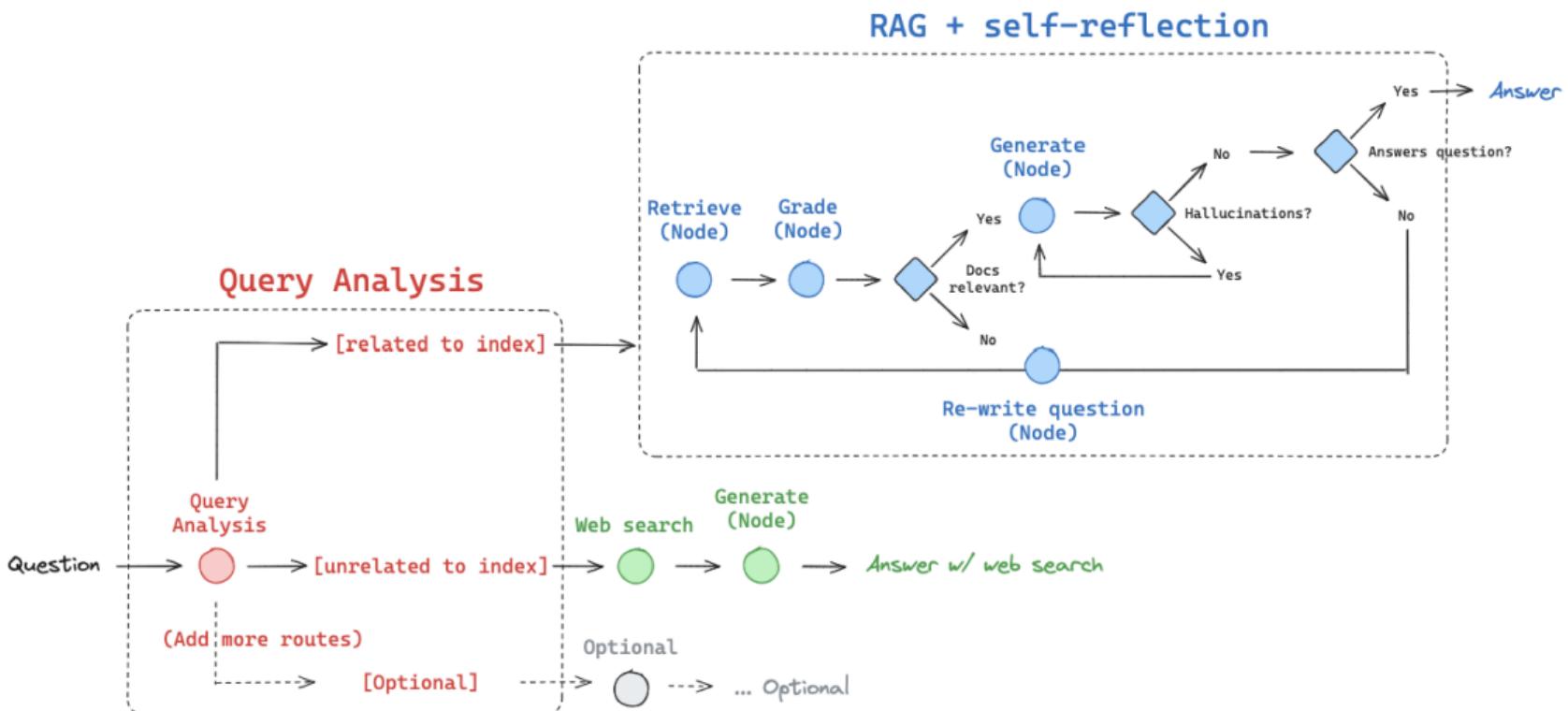
Design Pattern Tool - Principe



Design Pattern Tool - Principe



Design Pattern Tool - Principe



Design Pattern Tools - Principe

Published as a conference paper at ICLR 2017

LEARNING END-TO-END GOAL-ORIENTED DIALOG

Antoine Bordes, Y-Lan Boureau & Jason Weston

Facebook AI Research
New York, USA

{abordes, ylan, jassej}@fb.com

ABSTRACT

Traditional dialog systems used in goal-oriented applications require a lot of domain-specific handcrafting, which hinders scaling up to new domains. End-to-end dialog systems, in which all components are trained from the dialogs themselves, escape this limitation. But the encouraging success recently obtained in chit-chat dialog may not carry over to goal-oriented settings. This paper proposes a testbed to break down the strengths and shortcomings of end-to-end dialog systems in goal-oriented applications. Set in the context of restaurant reservation, our tasks require manipulating sentences and symbols in order to properly conduct conversations, issue API calls and use the outputs of such calls. We show that an end-to-end dialog system based on Memory Networks can reach promising, yet imperfect, performance and learn to perform non-trivial operations. We confirm these results by comparing our system to a hand-crafted slot-filling baseline on data from the second Dialog State Tracking Challenge (Henderson *et al.*, 2014a). We show similar result patterns on data extracted from an online concierge service.

1 INTRODUCTION

The most useful applications of dialog systems such as digital personal assistants or bots are currently goal-oriented and transactional: the system needs to understand a user request and complete a related task with a clear goal within a limited number of dialog turns. The workhorse of traditional dialog systems is *slot-filling* (Lemon *et al.*, 2006; Wang and Lemon, 2013; Young *et al.*, 2013) which predefines the structure of a dialog state as a set of slots to be filled during the dialog. For a restaurant reservation system, such as the location, price and type of a restaurant. Slot-filling is proven reliable but is inherently hard to scale to new domains: it is impossible to manually encode all features and slots that users might refer to in a conversation.

End-to-end dialog systems, usually based on neural networks (Song *et al.*, 2014; Vinyals and Le, 2015; Li *et al.*, 2015; Seffner *et al.*, 2015a, b; Xie *et al.*, 2015), eschew such limitations: all their components are directly trained on past dialogs, with no assumption on the domain or dialog state structure, thus making it easy to automatically scale up to new domains. They have shown promising performance in non goal-oriented chit-chat settings, where they were trained to predict the next utterance in social media and forum threads (Ritter *et al.*, 2011; Wang *et al.*, 2013; Lowe *et al.*, 2015) or movie conversations (Banchs, 2012). But the performance achieved on chit-chat may not necessarily carry over to goal-oriented conversations. As illustrated in Figure 1 in a restaurant reservation scenario, conducting goal-oriented dialog requires skills that go beyond language modeling, e.g., asking questions to clearly define a user request, querying Knowledge Bases (KBs), interpreting requests from queries and dialog operators, users of complex dialog transaction. This makes end-to-end dialog systems challenging to design and to evaluate, especially when comparing chit-chat performance in itself is not straightforward (Liu *et al.*, 2016). In particular, it is unclear if end-to-end models are in a position to replace traditional dialog methods in a goal-directed setting: can end-to-end dialog models be competitive with traditional methods even in the well-defined narrow-domain tasks where they exist? If not, where do they fall short?

This paper aims to make it easier to address these questions by proposing an open resource to test end-to-end dialog systems in a way that 1) favors reproducibility and comparisons, and 2) is lightweight and easy to use. We aim to break down a goal-directed objective into several subtasks to test some crucial capabilities that dialog systems should have (and hence provide error analysis by design).

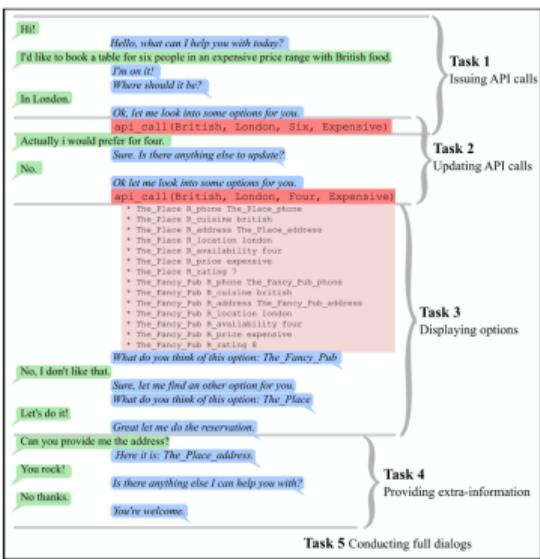


Figure 1: **Goal-oriented dialog tasks.** A user (in green) chats with a bot (in blue) to book a table at a restaurant. Models must predict bot utterances and API calls (in dark red). Task 1 tests the capacity of interpreting a request and asking the right questions to issue an API call. Task 2 checks the ability to modify



Design Pattern Tools - exemple Ollama

```
[  
  {  
    "type": "function",  
    "function": {  
      "name": "get_weather",  
      "description": "Get weather for a city",  
      "parameters": {  
        "type": "object",  
        "properties": {  
          "city": {"type": "string"},  
          "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}  
        },  
        "required": ["city"]  
      }  
    }  
  }  
]
```

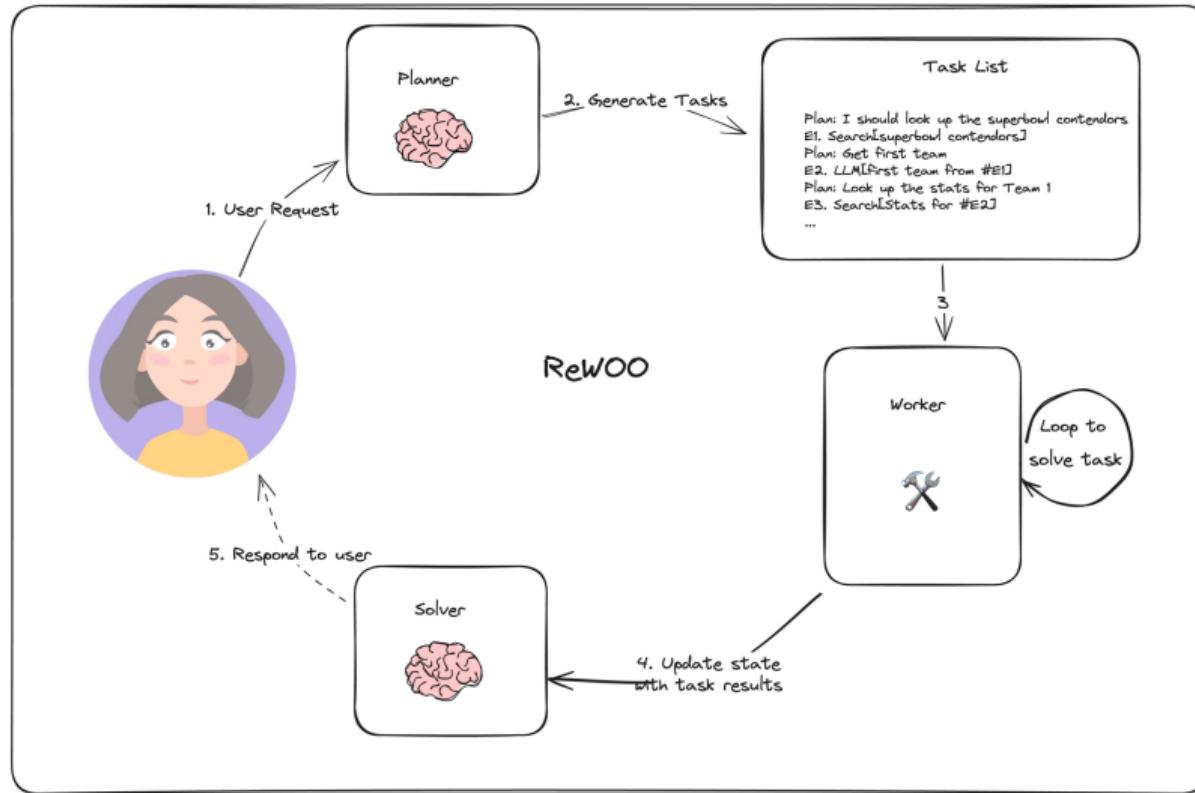
Avantages du Pattern Tools

- Accès à des informations en temps réel.
- Amélioration des capacités de calcul et de vérification.
- Adaptabilité aux tâches spécifiques grâce à l'intégration d'outils spécialisés.

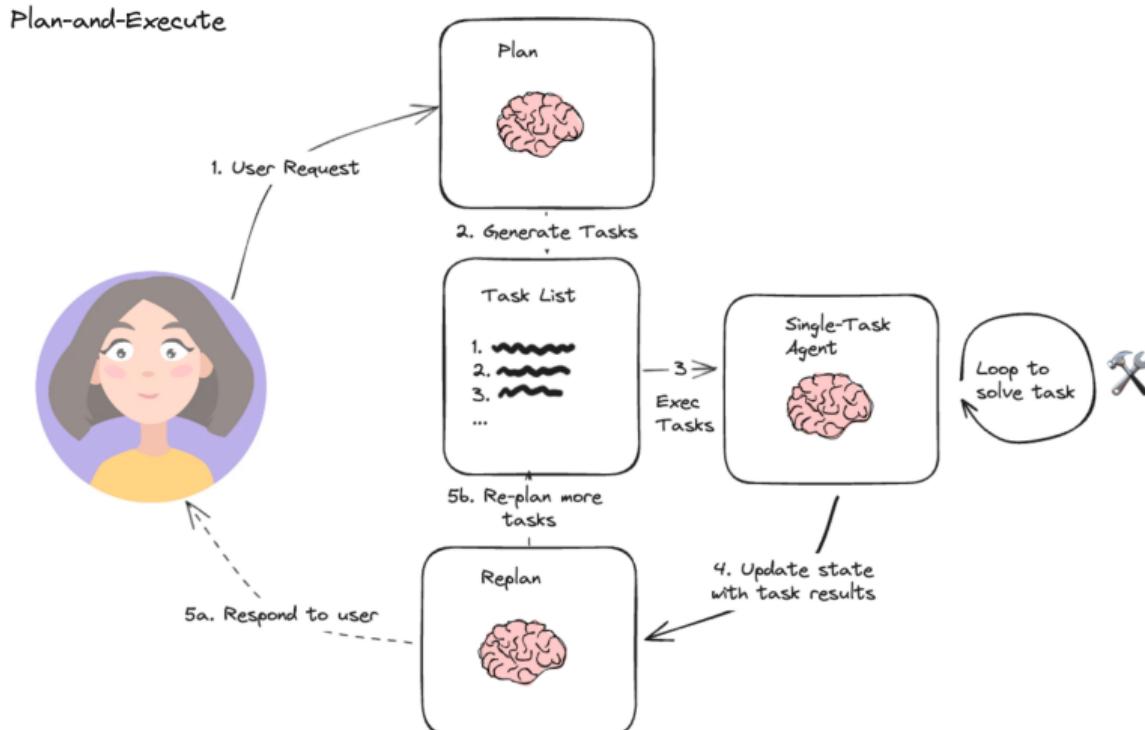
Design Pattern Planification

- **Définition** : Organise les étapes d'une tâche complexe pour améliorer la gestion des processus.
- **Utilité** : Permet de structurer les décisions et d'adapter les actions en fonction des besoins.
- **Exemple** : Planifier les étapes d'un projet, de la collecte des données à l'analyse finale.

Design Pattern Planification



Design Pattern Planification



Avantages du Pattern Planification

- Efficacité accrue grâce à une organisation structurée.
- Gestion de la complexité en divisant les tâches en étapes gérables.
- Flexibilité pour ajuster les plans en fonction des changements.

Exemple d'utilisation

- **Recherche d'informations** : Interroger une base de données pour récupérer des données précises.
- **Extraction de données** : Analyser des documents avec des outils externes.
- **Prise de décision assistée** : Utiliser des calculs externes pour recommander des solutions.

- **Définition** : Exploite la collaboration entre plusieurs agents spécialisés.
- **Utilité** : Permet de résoudre des problèmes complexes en parallèle, en combinant les compétences de plusieurs agents.
- **Exemple** : Gestion de la logistique avec des agents pour la gestion des stocks, la planification des transports et le suivi des livraisons.

Architecture d'agent

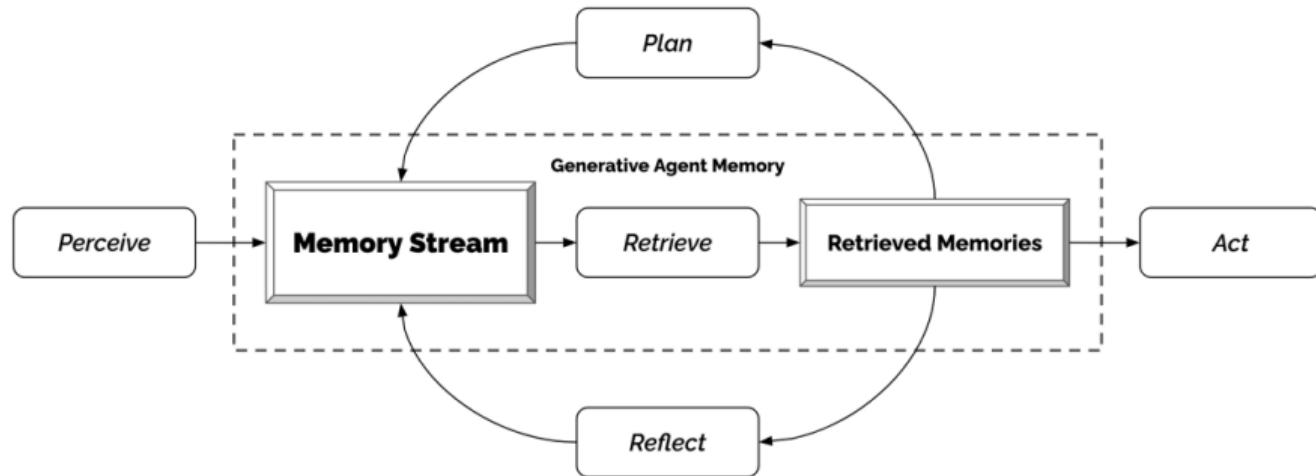


Figure 5: Our generative agent architecture. Agents perceive their environment, and all perceptions are saved in a comprehensive record of the agent's experiences called the memory stream. Based on their perceptions, the architecture retrieves relevant memories and uses those retrieved actions to determine an action. These retrieved memories are also used to form longer-term plans and create higher-level reflections, both of which are entered into the memory stream for future use.

Architecture d'agent

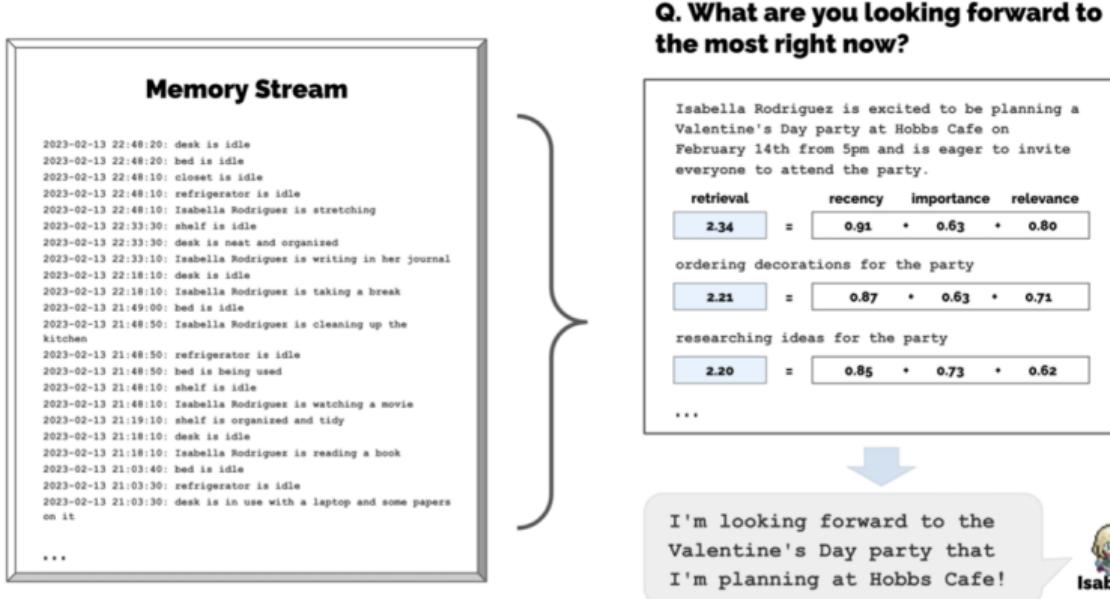
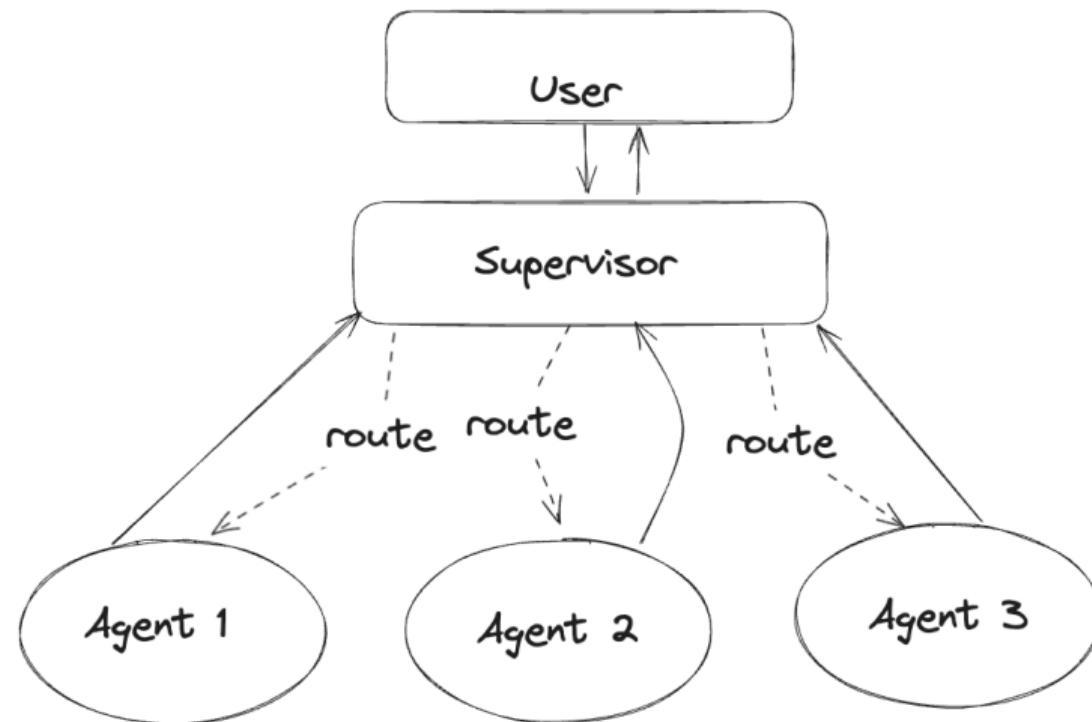
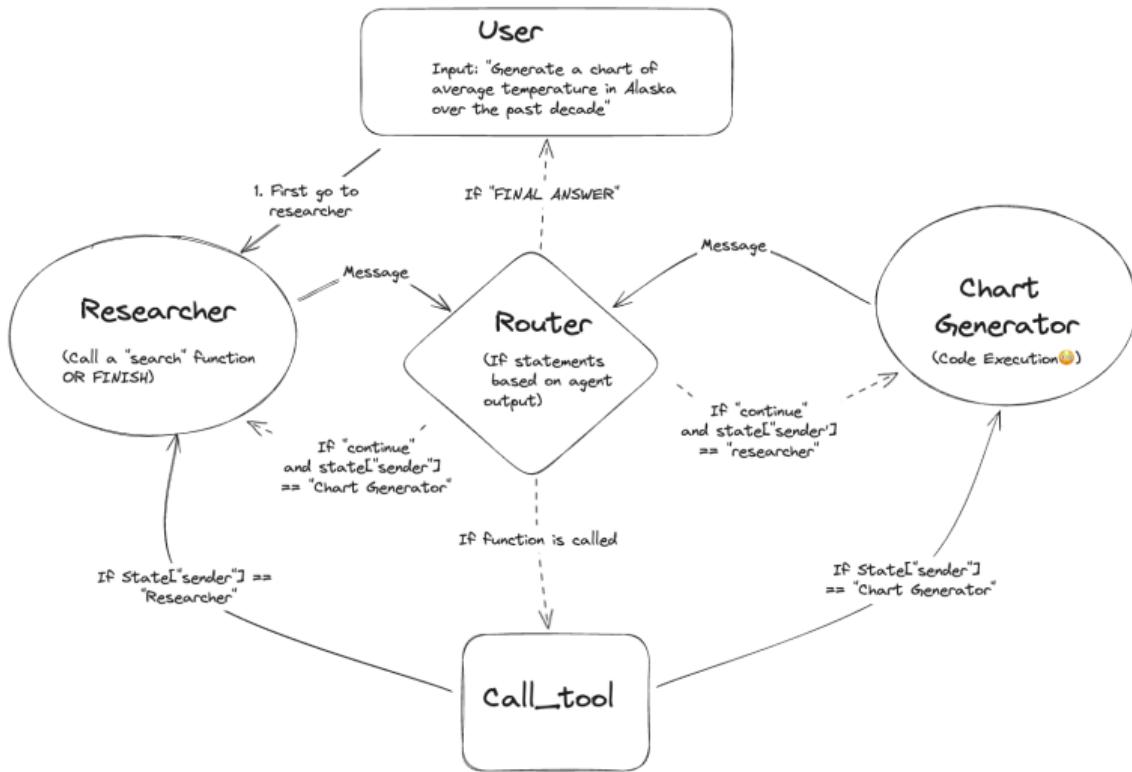


Figure 6: The memory stream comprises a large number of observations that are relevant and irrelevant to the agent's current situation. Retrieval identifies a subset of these observations that should be passed to the language model to condition its response to the situation.

Design Pattern Multi-Agent

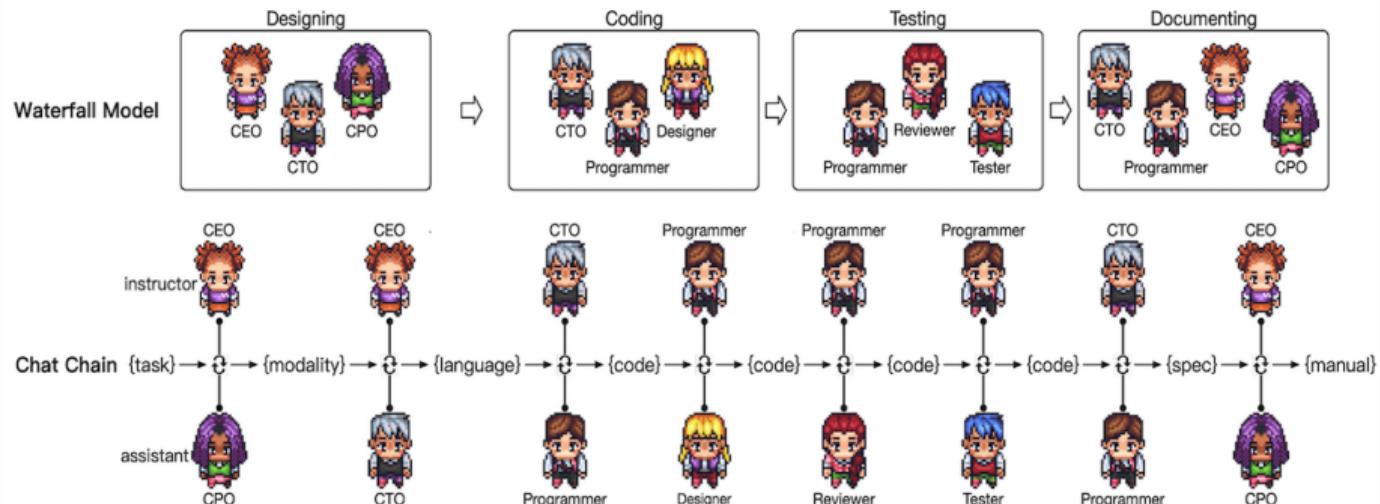


Design Pattern Multi-Agent



Design Pattern Multi-Agent

Agentic Design Patterns: Multi-Agent Collaboration



Proposed ChatDev architecture. Image adapted from “Communicative Agents for Software Development,” Qian et al. (2023).

Design Pattern Multi-Agent

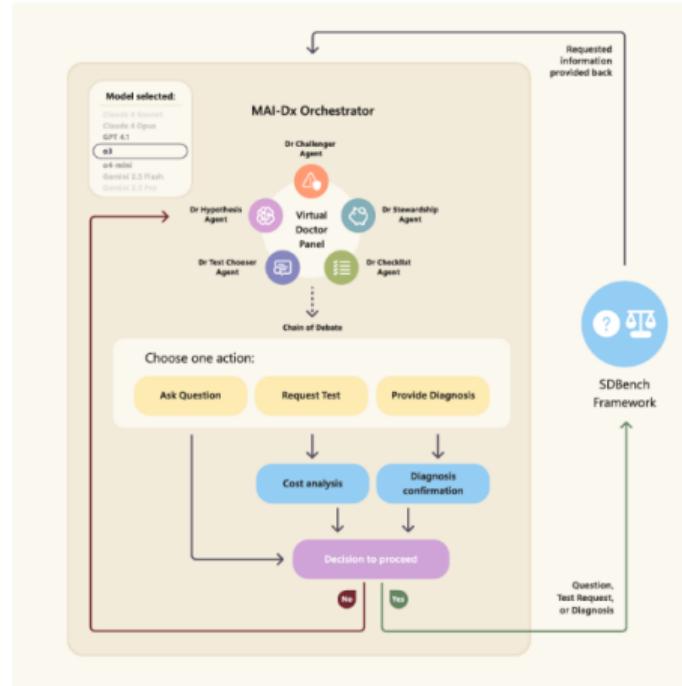


Figure 5: Overview of the MAI-Dx Orchestrator

Architecture Model Context Protocol

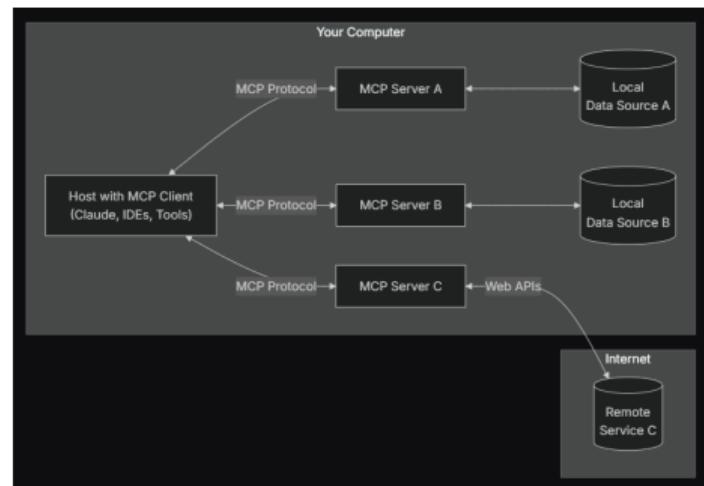
Hôtes MCP : Programmes comme Claude Desktop, des environnements de développement (IDE) ou des outils d'IA souhaitant accéder à des données via MCP.

Clients MCP : Clients du protocole qui maintiennent une connexion 1:1 avec les serveurs.

Serveurs MCP : Programmes légers exposant des fonctionnalités spécifiques à travers le protocole standardisé MCP (Model Context Protocol).

Sources de données locales : Fichiers, bases de données et services de votre ordinateur auxquels les serveurs MCP peuvent accéder de manière sécurisée.

Services distants : Systèmes externes accessibles via Internet (par exemple via des APIs) avec lesquels les serveurs MCP peuvent interagir.



modelcontextprotocol.io/introduction

Les LLMs comme composant central de la programmation

```
1 from pydantic import BaseModel
2
3 # A class based schema for a book
4 class BookSchema(BaseModel):
5     title: str
6     author: str
7     year: int
8
9 result = model.respond("Tell me about The Hobbit",
10                         response_format=BookSchema)
11
12 book = result.parsed
13 print(book)
14 # ^
15 # Note that `book` is correctly typed as
16 # { title: string, author: string, year: number }
```



Typepage

```
1 import lmstudio as lms
2
3 def multiply(a: float, b: float) -> float:
4     """Given two numbers a and b. Returns the product of them."""
5     return a * b
6
7 model = lms.llm("qwen2.5-7b-instruct")
8
9 model.act(
10     "What is the result of 12345 multiplied by 54321?",
11     [multiply],
12     on_message=print,
13 )
```

Tooling

Les LLMs comme composant central de programmation I

- **Au-delà des outils externes** : Plutôt que de considérer les LLMs comme des objets autonomes, comment les intégrer directement dans le paradigme même de programmation.
 - Analogue aux ramasse-miettes : une intégration fluide, au niveau du langage, pour automatiser les tâches répétitives ou complexes.
 - Exemple : les LLMs pourraient générer dynamiquement des chemins d'exécution ou optimiser la logique à la compilation ou à l'exécution.

Les LLMs comme composant central de programmation II

- **Backend pour les langages** : Utiliser les LLMs comme interpréteur à l'exécution ou comme moteur de compilation pour une nouvelle génération de langages.
 - Au lieu d'une syntaxe rigide, les développeurs écrivent un pseudocode orienté intention ; les LLMs le traduisent en instructions exécutables.
 - Cela réduit la charge syntaxique et rapproche la programmation du raisonnement humain.

Repenser la programmation logique I

- **Un potentiel de renouveau** : La programmation logique (ex. : Prolog) pourrait connaître un regain d'intérêt, car elle correspond plus naturellement aux forces des LLMs que les langages impératifs comme Python.
 - Le paradigme déclaratif—basé sur des règles et de l'inférence—reflète la capacité des LLMs à raisonner à partir de contraintes exprimées en langage naturel.
 - Exemple : les développeurs spécifient le "quoi" (objectifs/contraintes) en texte ; les LLMs en déduisent le "comment", à la manière d'une résolution logique.

- **Adaptabilité au contrôle par LLM** : Contrairement au flot de contrôle pas-à-pas de la programmation impérative, l'abstraction de la programmation logique pourrait mieux exploiter le raisonnement probabiliste des LLMs.
 - Les instructions explicites de Python s'accordent mal avec la tendance des LLMs à interpréter l'intention ; des systèmes à la Prolog pourraient combler cet écart.
 - Approche hybride : les LLMs génèrent dynamiquement des règles logiques, associées à des solveurs formels pour des résultats déterministes (Li et al., 2024).

Conclusion

- Les **Agentic Design Patterns** sont essentiels pour structurer et optimiser les agents IA et complémenter les capacités et limitations des modèles de langages comme base de connaissance et moteur d'inférence.
- Ils permettent de résoudre des problèmes complexes, d'améliorer la précision et d'augmenter l'efficacité.
- Les 4 premiers patterns (Réflexion, Tools, Planification, Multi-Agent) offrent des solutions adaptées à différents besoins.
- L'agentification reste une question largement ouverte.