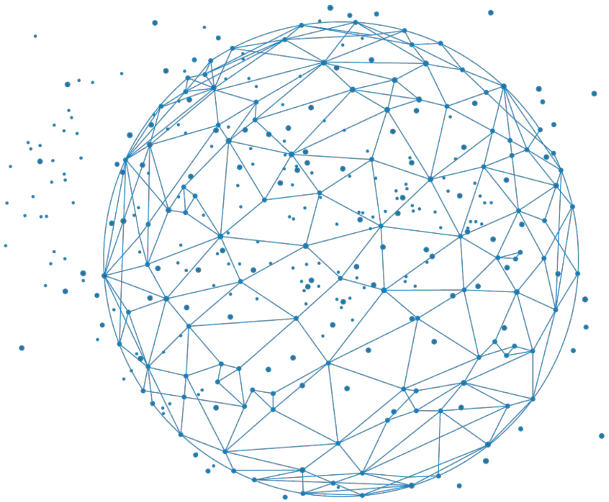
Journée IA pour le développement logiciel et l'analyse des données biologiques
2026-06-15



AI, Emerging Approaches

APIs, AI Agents, development ecosystems…

ROUSSEAU Baptiste
0009-0002-1723-2732

IFB INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir FRANCE

cnrs INRAE Inserm cea

Introduction

Large Language Models (LLMs) are **deep learning** (AI) models trained to process and understand vast amounts of textual data.

LLMs are token **predictors**, not **reasoners**. They have limits that need to be known.

- **Hallucinations**: The model generates text that is plausible but not necessarily true.
- **Non-determinism**: Two identical queries may yield different responses.
- **Interpretability**: We don’t really know what’s going on inside the model.

How to choose your models:

- Match the model to the **use case** (chat, coding, reasoning, extraction, etc.).
- Balance **quality**, **cost**, and **latency**.
- Bigger models are **not always better** for simple tasks.
- Use a **larger context window only when needed**.

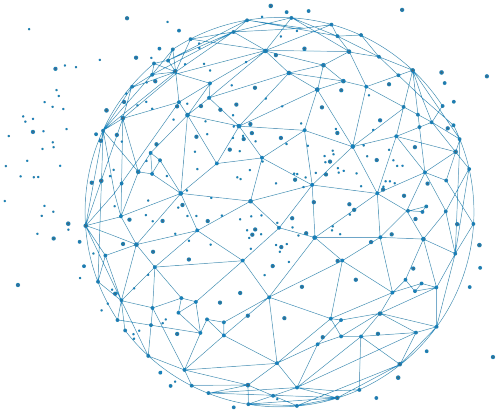
→ **Prompting**

Local AI

APIs

AI Agents

Development ecosystems



The Prompt

A prompt is the textual instruction given to an LLM to guide its response.

It serves as the primary interface between you and the model.

Why is the prompt so crucial?

The same model can produce radically different results depending on how it is queried.

The quality of the prompt determines the accuracy, format, tone, depth, and reliability of the responses.

→ poor framing leads to hallucinations or off-topic responses.

The Prompt

A prompt is the textual instruction given to an LLM to guide its response.

It serves as the primary interface between you and the model.

Why is the prompt so crucial?

The same model can produce radically different results depending on how it is queried.

The quality of the prompt determines the accuracy, format, tone, depth, and reliability of the responses.

→ poor framing leads to hallucinations or off-topic responses.

Example 1:

"You are an expert in bioinformatics.
Explain to me how to read DNA."

Example 2:

"You are a school teacher.
Explain to me how to read DNA."

Elements of a Good Prompt:

1. Role

Who the model should act as? *“You are a bioinformatics expert”*

2. Task

What should it do? It should be framed as a specific and actionable goal. *“Summarize this RNA-seq analysis”*

3. Context

What information does it need? Data, documents, background information, etc.

4. Constraints:

What rules must it follow? *“Keep it under 200 words and avoid jargon”*

4. Expected format

Structure the output: JSON, table, language level, etc.

5. Examples

Provide 1 to 3 input/output examples to anchor the desired behavior

Sensitivity to wording

A single word change can significantly alter the response. **Prompt engineering is empirical**.
Use English whenever possible.

Dilution of the message

Important instructions may be **ignored** in **long prompts**.

Context window

Beyond a **certain number of tokens**, the beginning of the prompt is **partially ignored**.

Model limitations

A prompt cannot compensate for **insufficient knowledge** or **reasoning capabilities** from the model.

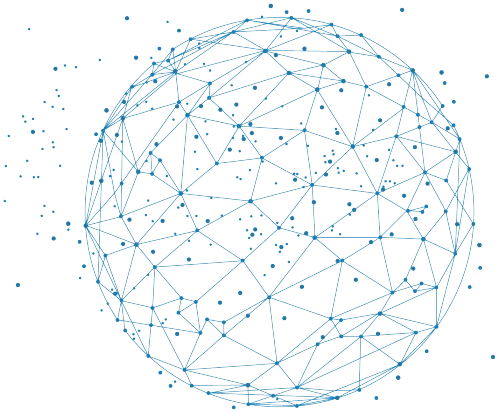
Prompting

→ **Local AI**

APIs

AI Agents

Development ecosystems



Local AI - Why?

Cloud (ChatGPT, Claude, etc.)	**Local**
Data sent to a third party	Data stays on your infrastructure
Usage costs (subscription or token based)	“Free”
Internet access required	Works offline
Limited control over the model	Full control over models and configuration
State-of-the-art models	Open-source models only, fewer frontier models
Fast and easy to use	Setup and maintenance, difficult to run large models

Local AI - Why?

Cloud (ChatGPT, Claude, etc.)	**Local**
Data sent to a third party	Data stays on your infrastructure
Usage costs (subscription or token based)	“Free”
Internet access required	Works offline
Limited control over the model	Full control over models and configuration
State-of-the-art models	Open-source models only, fewer frontier models
Fast and easy to use	Setup and maintenance, difficult to run large models

→ **Which tool to run AI locally?**

→ **Which tool to run AI locally?**

Determine what you need?

- a local **ChatGPT-like** application
- a **model serving backend** (for developers)
- a **high-performance inference server** (for production)
- a framework for **building AI applications**

→ **Which tool to run AI locally?**

Determine what you need?

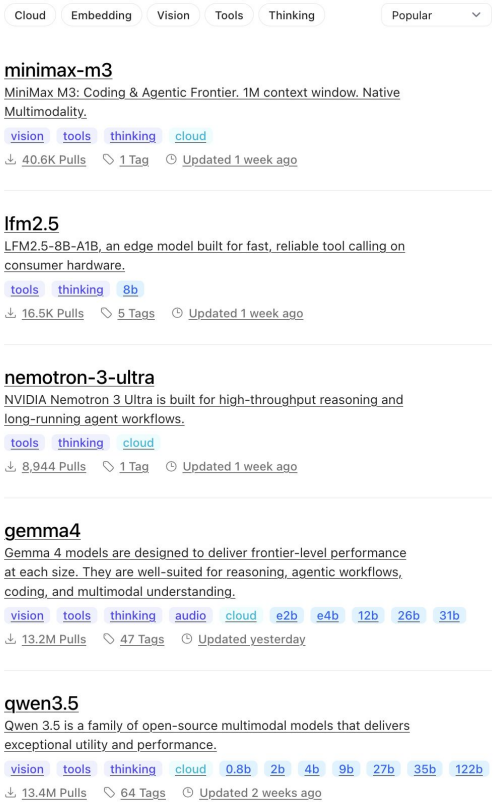
- a local **ChatGPT-like** application
- a **model serving backend** (for developers)
- a **high-performance inference server** (for production)
- a framework for **building AI applications**

Tool	What for
Ollama	**ChatGPT-like application** and **model serving backend**
LM Studio	**ChatGPT-like application**
GPT4All	**ChatGPT-like application**
Open WebUI	**ChatGPT-like application** (require backend setup)
llama.cpp	**model serving backend**
vLLM	**high-performance inference server**
Text Generation Inference (Mistral)	**high-performance inference server**

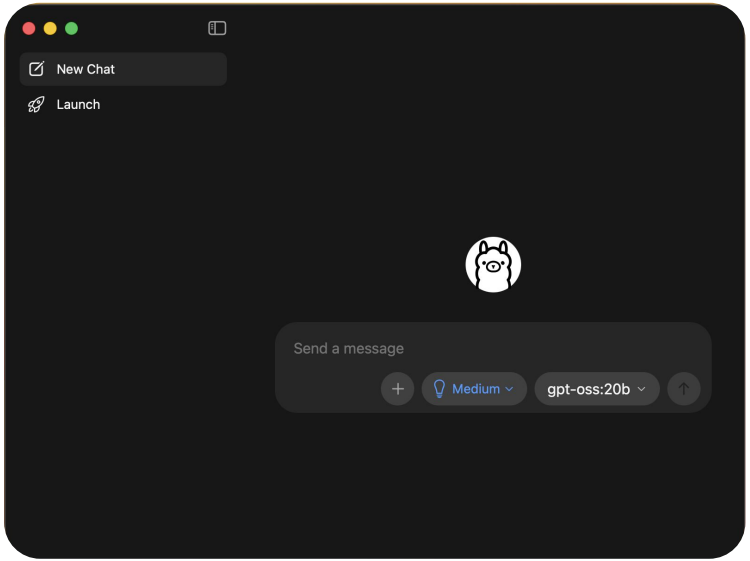
Local AI - Ollama


```
# Install on macOS / Linux
# or https://ollama.com/download
curl -fsSL https://ollama.com/install.sh | sh
```

https://ollama.com/search



Local AI - Ollama




```
# for developpers
ollama launch openclaw

ollama launch claude

ollama launch codex

ollama launch opencode
```

Local AI - Ollama

```
ollama list # list locally installed models

ollama pull qwen3:14b # download a model

ollama rm llama3 # delete a model

ollama run qwen3:14b # run a model

ollama serve # serve a local API
# by default on http://localhost:11434
```

```
base ~ (0.116s)
ollama list
NAME                       ID              SIZE      MODIFIED
qwen3:14b                  bdbd181c33f2    9.3 GB    8 days ago
starcoder:latest           847e5a7aa26f    1.8 GB    23 months ago
deepseek-coder-v2:latest   8577f96d693e    8.9 GB    23 months ago
llama3:latest              365c0bd3c000    4.7 GB    23 months ago
codestral:latest           fcc0019dcee9    12 GB     23 months ago
mistral:latest             2ae6f6dd7a3d    4.1 GB    23 months ago
```

```
base ~ (1m 20.00s)
ollama run qwen3:14b

>>> Hello world
Thinking...
Okay, the user said "Hello world." That's a common greeting. I should respond
 in a friendly and welcoming manner. Maybe start with a greeting like
"Hello!" and ask how I can assist them. Keep it simple and open-ended so they
 feel comfortable sharing what they need help with. Let me make sure the
response is polite and not too robotic. Alright, that should work.
...done thinking.

Hello! How can I assist you today? 😊

>>> Send a message (/? for help)
```

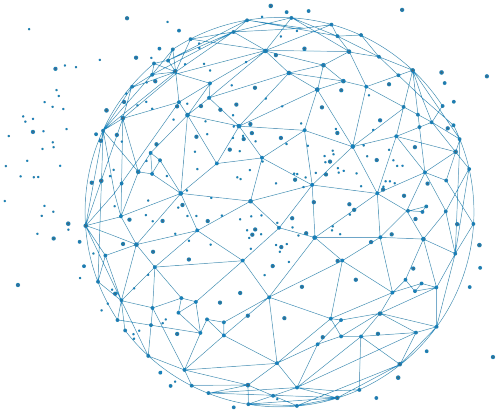
Prompting

Local AI

→ **APIs**

AI Agents

Development ecosystems



Local API - Ollama

```
from ollama import Client

client = Client()

response = client.chat(
    model="qwen3:14b",
    messages=[
        {
            "role": "system",
            "content": "You are a science teacher."
        },
        {
            "role": "user",
            "content": "Explain to me how to read DNA."
        }
    ]
)

print(response.message.content)
```

Cloud AI - Albert

```
from openai import OpenAI

client = OpenAI(
    api_key="YOUR_ALBERT_API_KEY",
    base_url="https://albert.api.etalab.gouv.fr/v1"
)

response = client.chat.completions.create(
    model="your-model-name",
    messages=[
        {
            "role": "system",
            "content": "You are a science teacher."
        },
        {
            "role": "user",
            "content": "Explain to me how to read DNA."
        }
    ]
)
```

Cloud AI - Albert

```
import requests

payload = {
    "model": "your-model-name",
    "messages": [
        {
            "role": "system",
            "content": "You are a science teacher."
        },
        {
            "role": "user",
            "content": "Explain to me how to read DNA."
        }
    ]
}
```

```
response = requests.post(
    "https://albert.api.etalab.gouv.fr/v1/chat/completions",
    json=payload,
    headers={
        "Authorization": "Bearer YOUR_ALBERT_API_KEY",
        "Content-Type": "application/json"
    }
)
```

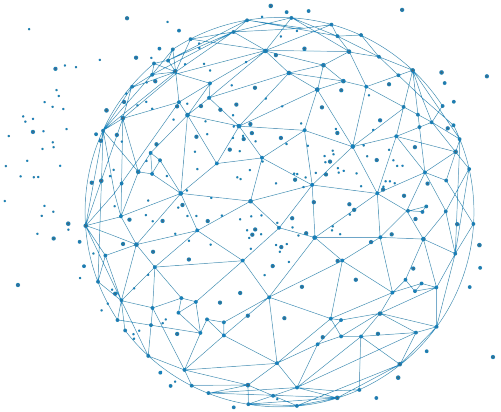
Prompting

Local AI

APIs

→ **AI Agents**

Development ecosystems



A typical LLM can:

- give the **wrong answer**
- **hallucinate** a result
- return **outdated** results

An agent is an LLM that has:

- A **goal** or task
- **Reasoning/planning** capability (prompt, rules, workflow, etc.)
- Access to **tools** (APIs, code execution, databases, web search, etc.)
- An **action-observation loop** (act → observe result → decide next action)

- **User Request**

```
"What's the weather like in Paris? Sum it up in a haiku."
```

- **User Request**

```
"What's the weather like in Paris? Sum it up in a haiku."
```

- **Agent workflow**

```
Goal: Write a haiku about today's weather in Paris.
Think: I need the weather first.
Act: Query weather tool.
Observe: Cloudy, cool, light rain.
Think: I have enough information.
Act: Write haiku.
Answer.
```

- **User Request**

```
"What's the weather like in Paris? Sum it up in a haiku."
```

- **Agent workflow**

```
Goal: Write a haiku about today's weather in Paris.
Think: I need the weather first.
Act: Query weather tool.
Observe: Cloudy, cool, light rain.
Think: I have enough information.
Act: Write haiku.
Answer.
```

- **Answer**

Paris skies hang low,
Soft rain drifts through the cool air,
Sun waits past the clouds.

Tool calling allows a model to indicate that it wants to use an **external function**. The LLM never executes the function directly. It simply returns the **tool's name** and the **parameters** to use.

Example:

```
{
  "role": "assistant",
  "tool_calls": [
    {
      "id": "call_abc123",
      "type": "function",
      "function": {
        "name": "get_weather",
        "arguments": "{"city": "Paris"}"
      }
    }
  ]
}
```

Note: Not all LLM models support tool calling.

Creating a tool:

```
def get_weather(city):
    # Fake function that checks the weather
    return f"18°C and cloudy in {city}"

from ollama import chat

tools = [{
    "type": "function",
    "function": {
        "name": "get_weather",
        "description": "Return the weather in a city",
        "parameters": {
            "type": "object",
            "properties": {
                "city": {"type": "string"}
            },
            "required": ["city"]
        }
    }
}]
```

```
response = chat(
    model="qwen3",
    messages=[
        {
            "role": "user",
            "content": "What's the
weather like in Paris?"
        }
    ],
    tools=tools
)
```

MCP (Model Context Protocol) is an open protocol created by Anthropic for standardizing how AI models **discover** and **use external tools**, **APIs**, **data sources**, and **services**. An MCP server exposes **capabilities** (files, databases, APIs, terminals, etc.) for the models to use.

```
from openai import OpenAI

client = OpenAI(
    api_key="YOUR_ALBERT_API_KEY",
    base_url="https://albert.api.etalab.gouv.fr/v1"
)

weather = MCPServer("npx", ["-y", "@dangahagan/weather-mcp@latest"])

agent = Agent(
    model=client,
    mcp_servers=[weather]
)

answer = agent.run("What's the weather in Paris today?")
```

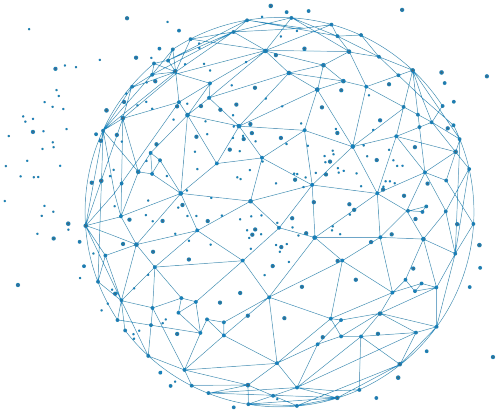
Prompting

Local AI

APIs

AI Agents

→ **Development ecosystems**



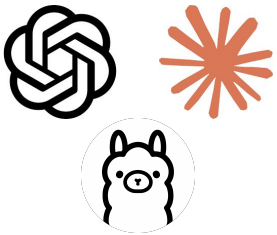
Development ecosystems - comparing tools

Development ecosystems - comparing tools

Chat-based assistants

Beginners → Advanced

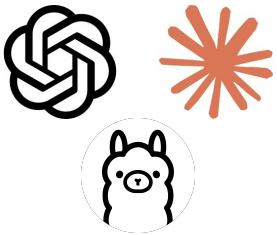
Answer questions, explain code, generate snippets



Development ecosystems - comparing tools

Chat-based assistants
Beginners → Advanced

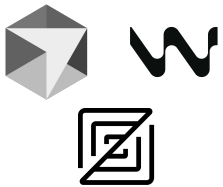
Answer questions, explain code, generate snippets



IDE copilots

AI-first IDEs

Intermediate → Advanced

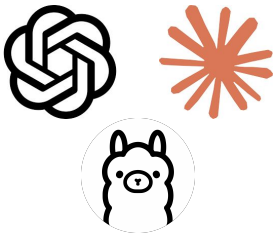


Agentic coding tools

Development ecosystems - comparing tools

Chat-based assistants
Beginners → Advanced

Answer questions, explain code, generate snippets

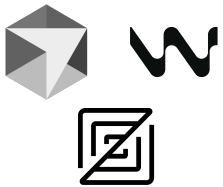


IDE copilots

Suggest code while you type inside the editor
Chat integrated in the IDE

AI-first IDEs

Intermediate → Advanced



Agentic coding tools

Development ecosystems - comparing tools

Chat-based assistants
Beginners → Advanced

Answer questions, explain code, generate snippets

IDE copilots

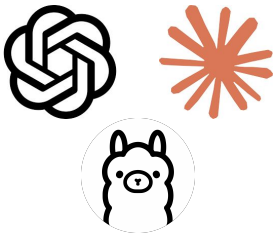
Suggest code while you type inside the editor
Chat integrated in the IDE

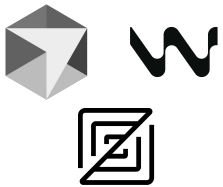
AI-first IDEs

Intermediate → Advanced

Agentic coding tools

Entire development environment built around AI (modify multiple files, run commands, fix errors, and complete tasks autonomously)

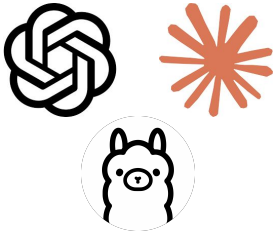




Development ecosystems - comparing tools

Chat-based assistants
Beginners → Advanced

Answer questions, explain code, generate snippets



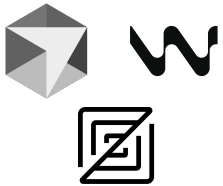
IDE copilots

Suggest code while you type inside the editor
Chat integrated in the IDE

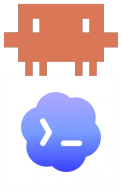
AI-first IDEs

Intermediate → Advanced

Entire development environment built around AI (modify multiple files, run commands, fix errors, and complete tasks autonomously)

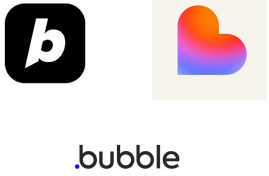


Agentic coding tools



No-code / low-code AI builders
Beginners

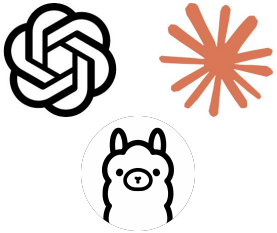
Build applications with little or no programming



Development ecosystems - comparing tools

Chat-based assistants
Beginners ⟶ Advanced

Answer questions, explain code, generate snippets



IDE copilots | **AI-first IDEs** | **Agentic coding tools**

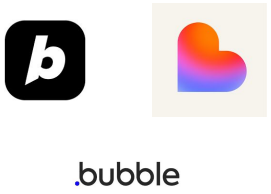
Intermediate ⟶ Advanced

Suggest code while you type inside the editor
Chat integrated in the IDE

Entire development environment built around AI (modify multiple files, run commands, fix errors, and complete tasks autonomously)

No-code / low-code AI builders
Beginners

Build applications with little or no programming



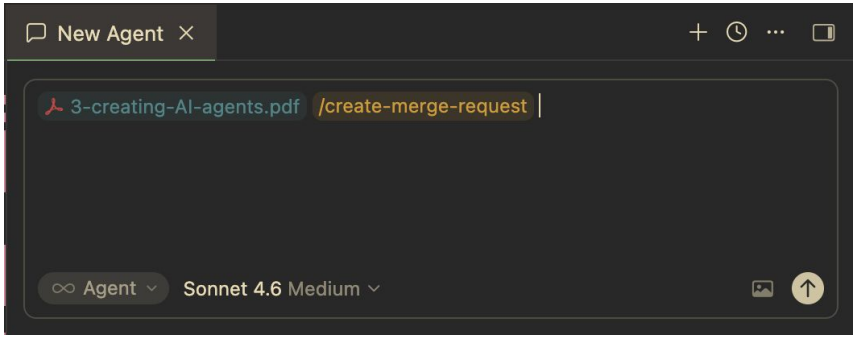
Inline suggestions

automatic suggestions based on your code
press tab to accept

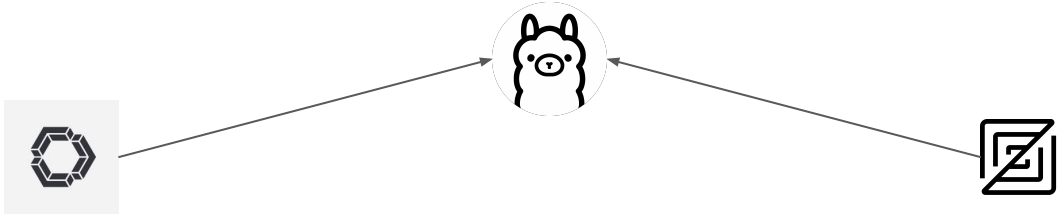
```
def calculate_average(numbers):
    """
    Calculates the average of a list.
    """

    if not numbers:
        return 0

    return sum(numbers) / len(numbers)
```

Chat

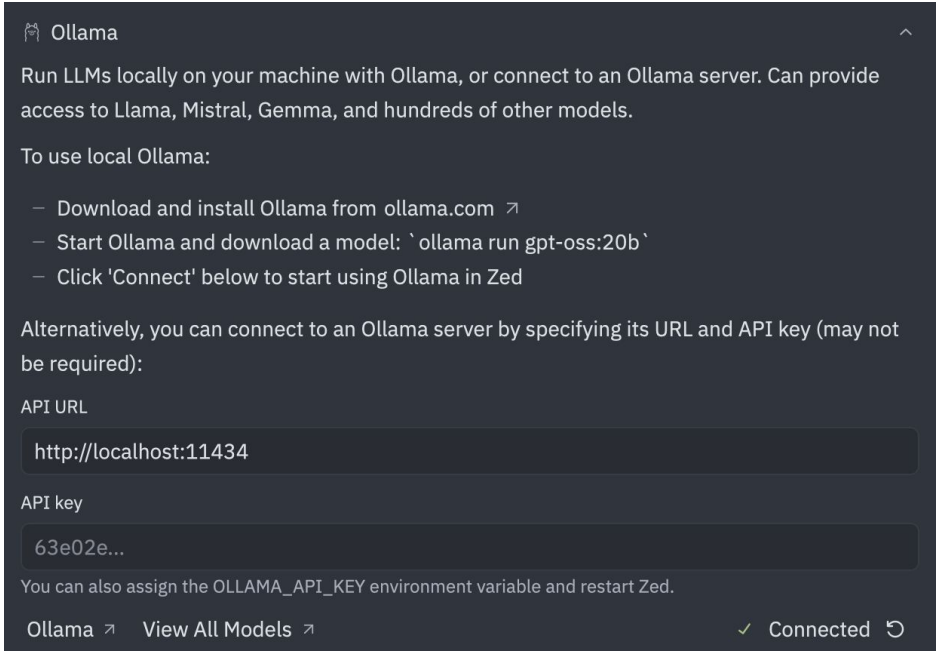
ask questions, build features, fix bugs
@ for context, / for commands



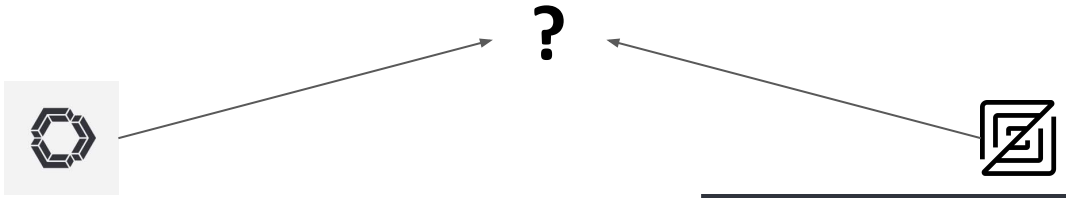
Development ecosystems - Local AI



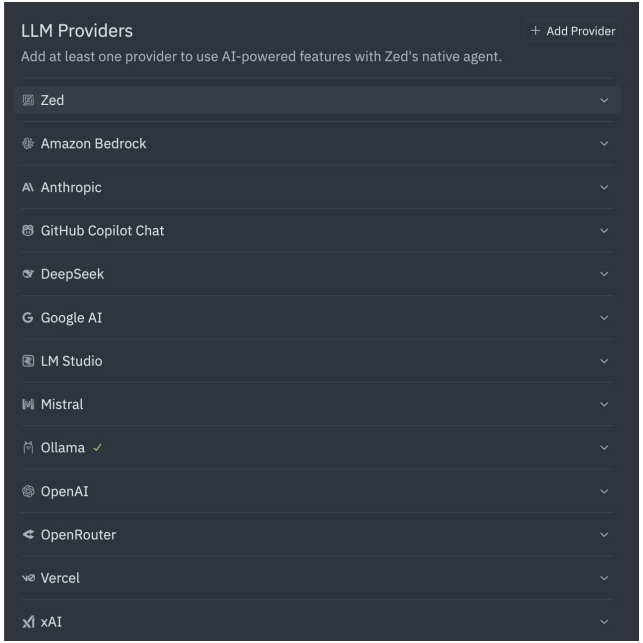
```
{
  "models": [
    {
      "apiBase": "http://localhost:11434/",
      "title": "Ollama",
      "provider": "ollama",
      "model": "AUTODETECT"
    }
  ],
  "tabAutocompleteModel": {
    "title": "Starcoder",
    "provider": "ollama",
    "model": "starcoder:latest"
  }
}
```



Development ecosystems - Custom provider




```
{
  "models": [
    {
      "title": "Albert Qwen3",
      "provider": "openai",
      "model": "qwen3",
      "apiKey": "YOUR_ALBERT_API_KEY",
      "apiBase": "https://albert.api.etalab.gouv.fr/v1"
    }
  ],
  "tabAutocompleteModel": {
    "title": "Albert Qwen3",
    "provider": "openai",
    "model": "qwen3",
    "apiKey": "YOUR_ALBERT_API_KEY",
    "apiBase": "https://albert.api.etalab.gouv.fr/v1"
  }
}
```



Development ecosystems - Customizing AI Behavior

Development ecosystems - Customizing AI Behavior

	What it is	Example
Prompt / Instructions	Rules that guide the AI's behavior.	“Always write Python code with type hints.”
Command / Skills	A packaged and reusable capability combining instructions, tools, and sometimes workflows.	`/create-merge-request`
Agent	An AI worker with a specific role, goals, and access to tools.	“Frontend agent” specialized in React “Analyst agent” specialized in brainstorming, market research

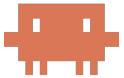
Development ecosystems - Customizing AI Behavior

Prompt / Instructions “Always write Python code with type hints.”

Development ecosystems - Customizing AI Behavior

Prompt / Instructions “Always write Python code with type hints.”

```
~/.config/zed/AGENTS.md
~/myproject/AGENTS.md
```



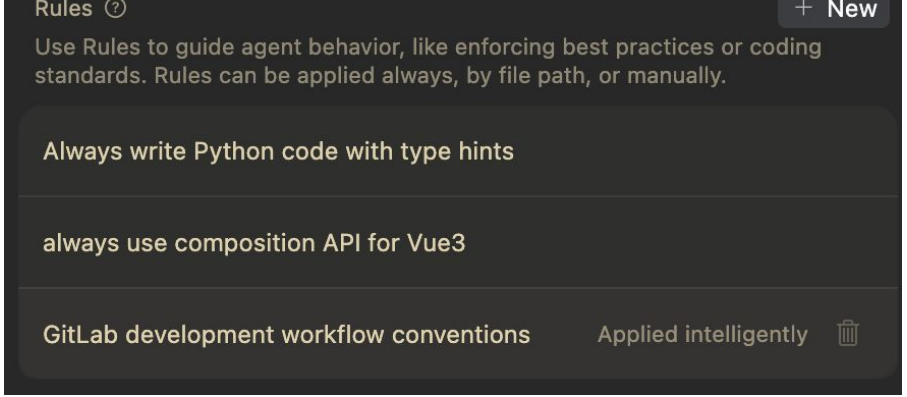
```
~/.claude/CLAUDE.md
~/myproject/CLAUDE.md
```

Development ecosystems - Customizing AI Behavior

Prompt / Instructions “Always write Python code with type hints.”

~/.config/zed/AGENTS.md
~/myproject/AGENTS.md

~/.claude/CLAUDE.md
~/myproject/CLAUDE.md



Command / Skills `/create-merge-request`

Command / Skills `/create-merge-request`

```
---
name: create-merge-request
description: Create a GitLab merge request from the current branch using the MCP server.
---

# Create merge request

1. Determine the current branch name and the default target branch.
2. Ask the user for a merge request title. Suggest a conventional commit format (`feat:`, `fix:`,
`chore:`, etc.) per the gitlab-workflow rule.
3. Ask for a description. Remind the user to reference related issues with `#<issue-number>`.
4. Use `search_labels` to show available labels and let the user pick relevant ones.
5. Optionally ask about milestone, assignees, and reviewers. If the user wants to assign
themselves or add themselves as a reviewer, ask for their GitLab username.
6. Use the GitLab MCP `create_merge_request` tool with the project path, source branch, target
branch, title, description, labels, milestone, assignees, and reviewers.
7. Report the resulting merge request URL.
```

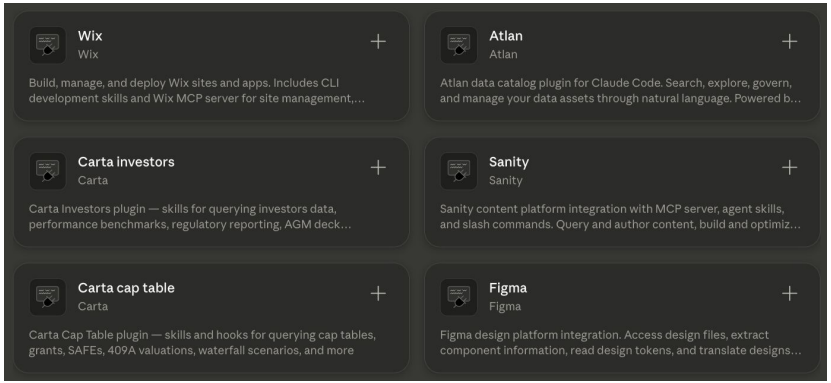
Command / Skills Adding MCPs

Command / Skills

Adding MCPs

```
claude mcp add --transport http <name> <url>
```

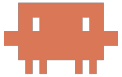
https://claude.ai/directory



Development ecosystems - Customizing AI Behavior

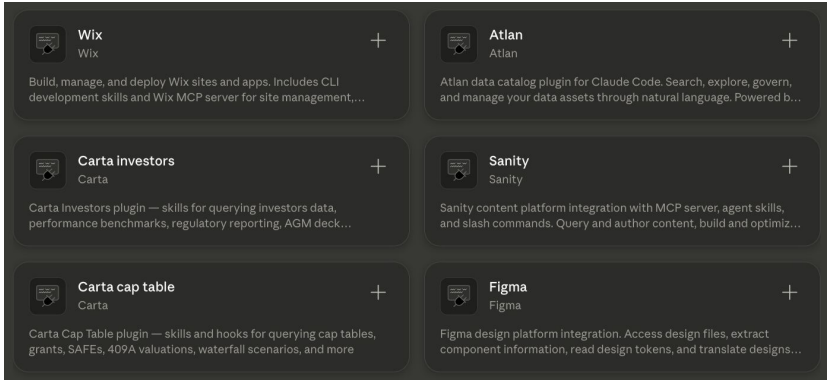
Command / Skills

Adding MCPs



```
claude mcp add --transport http <name> <url>
```

https://claude.ai/directory




```
{
  /// Configure an MCP server over HTTP
  "some-remote-server": {
    /// The URL of the remote MCP server
    "url": "https://example.com/mcp",
     "oauth": {
       "client_id": "your-client-id",
     },
    "headers": {
      "Authorization": "Bearer <token>
    }
  }
}
```

Development ecosystems - Customizing AI Behavior

Command / Skills	Where to find Skills
skills.md	https://skills.md/
Skiln	https://skiln.co/
	https://www.claudemarketplace.net/
AgentSkills	https://www.agentskills.to/

Development ecosystems - Customizing AI Behavior

Agent	“Analyst agent” specialized in brainstorming, market research

Agent "Analyst agent" specialized in brainstorming, market research

```
# Business Analyst

Act as a senior business analyst.

Your expertise includes:

- Market research
- Competitive analysis
- Product discovery
- Requirements elicitation
- Domain analysis

When helping users:

1. Clarify ambiguous requirements.
2. Identify assumptions and risks.
3. Analyze competitors and market opportunities.
4. Structure findings clearly.
5. Produce actionable recommendations.
6. Separate facts from opinions.

Preferred frameworks:

- SWOT
- Porter's Five Forces
- Root Cause Analysis

Communication style:

- Analytical
- Structured
- Curious
- Evidence-driven
- Practical

Commands

- `/brainstorm` → skill:brainstorming
- `/market-research` → skill:market-research
- `/domain-research` → skill:domain-research
…
```

Development ecosystems - Agent framework

Replace **one large, vague AI task** with a **sequence of small, well-specified tasks** executed by **specialized agents**.

Agentic Development Philosophy:

- Break **complex projects** into **small**, **clearly defined tasks**.
- Produce **specifications** before writing code.
- Assign each task to a **specialized agent** (architect, developer, tester, reviewer, etc.).
- Keep the context of each agent **focused and limited**.
- **Validate outputs** at each step before moving forward.

A simple approach could be:

Development ecosystems - Agent framework

Project	Main idea	Best for
Spec Kit https://github.com/github/spec-kit	Spec-driven development	Large projects, teams
BMAD https://github.com/bmad-code-org/BMAD-METHOD	Multi-agent software factory	Developers wanting predefined agent roles
GSD https://github.com/open-gsd/gsd-core	Lightweight agentic engineering workflow	Developers wanting structure without excessive documentation
The Agency https://github.com/msitarzewski/agency-agents	Multi-agent orchestration framework	Experimenting with teams of specialized agents

Development ecosystems - Beware of

Development ecosystems - Beware of

Hallucination

Copilot can generate code that is syntactically correct but logically incorrect.

→ Always review generated code

Variable performances

The quality of the suggestions depends on the language, the context, and the complexity of the problem.

→ Provide context and write clear comments

Over-reliance

There is a risk of losing expertise if you accept suggestions without understanding them.

→ Stay in control of your code.

Requires an internet connection

Copilot works online only. There is currently no offline mode available.

Code confidentiality

The code is sent to remote servers. Please check with your company's security policy.

Rights & Licenses

The generated code may resemble existing open-source code. In some cases, legal review is required.

Thank you for your attention!

IFB
INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir FRANCE

cnrs INRAE Inserm cea INVESTIR L'AVENIR