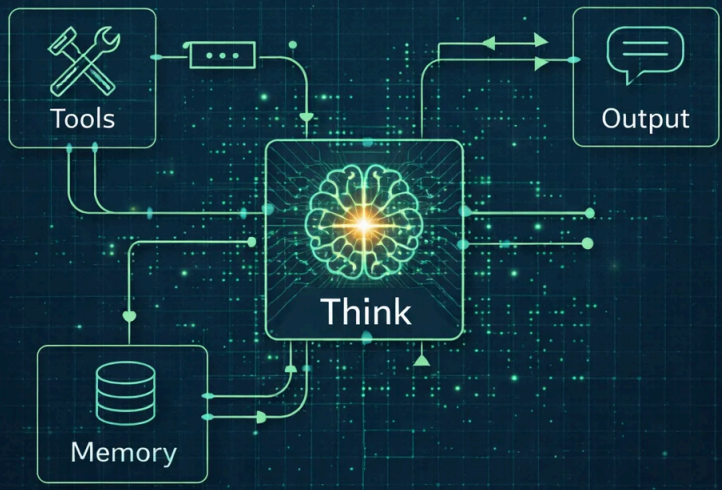
AGENTS LLM EN PYTHON

Guide Technique



Patrice HUETZ

Agents LLM en Python

Patrice Huetz

patrice-huetz.fr

patrice-huetz.fr · contact@patrice-huetz.fr

Préface

Ce livre n'est pas pour tout le monde

Si vous cherchez une introduction théorique aux Transformers ou l'histoire de l'IA depuis Turing, passez votre chemin. Ces informations sont gratuites sur YouTube.

Ce livre est pour vous si :

- ✅ Vous avez déjà joué avec ChatGPT/Claude et vous voulez aller plus loin
- ✅ Vous voulez construire des **outils qui travaillent pour vous**, pas des chatbots qui bavardent
- ✅ Vous êtes prêt à **coder** (Python) et pas juste à lire des concepts
- ✅ Vous voulez éviter les erreurs qui m'ont coûté des semaines et des centaines de dollars

Ce que vous allez construire

À la fin de ce livre, vous aurez construit :

1. Un Agent de Développement (~Chapitre 1-4)

Un assistant qui lit votre code, propose des modifications, et vérifie que ça compile. Comme GitHub Copilot, mais que vous contrôlez.

2. Un Système RAG Intelligent (~Chapitre 5)

Pas un RAG basique qui retourne des chunks aléatoires. Un système qui comprend les **dépendances** entre vos fichiers et retourne le contexte complet.

3. Un Agent Financier (~Chapitre 9-10)

Le projet fil rouge : un agent qui analyse des données boursières, génère des graphiques, et produit des rapports. Code complet et testé.

4. Un Système Multi-Agents (~Chapitre 13)

Une équipe d'agents qui collaborent : un planifie, un code, un review. Le pattern utilisé par les outils comme Devin et Claude Code.

Ce que ce livre va vous éviter

L'erreur à $847

Un de mes premiers agents est parti en boucle infinie pendant 6 heures. Facture OpenAI : $847. Chapitre 1, section 1.2 : comment ne jamais reproduire cette erreur.

Les 3 semaines de refactoring

J'ai construit un agent avec le mauvais framework. Trois semaines à tout réécrire. Chapitre 7 : le tableau de décision pour choisir le bon framework du premier coup.

Le bug "Lost in the Middle"

Mon agent "oubliait" des informations cruciales placées au milieu du contexte. J'ai perdu 2 semaines à comprendre pourquoi. Chapitre 14 : le phénomène scientifique et les solutions.

La facture mensuelle de $3,200

Premier mois en production : $3,200 de tokens. Après optimisation : $890 pour la même qualité. Chapitre 2 : le router hybride qui économise 70%.

Comment ce livre est différent

Livre classique	Ce livre
"Un agent est composé de..."	**Le code minimal qui marche** (50 lignes)
"Le paradigme ReAct stipule..."	**Le template copier-coller** testé en production
"Considérons les avantages de..."	**Le problème concret → la solution**
Exercice : "Réfléchissez à..."	Exercice : **Construisez X en 30 minutes**

Chaque chapitre commence par **le résultat que vous obtiendrez** et le **temps estimé**.

Prérequis

Obligatoire

- Python intermédiaire (classes, async/await, packages)
- Un compte OpenAI ou Anthropic (ou Ollama installé)
- ~$20 de crédits API pour les exercices (ou utiliser les modèles locaux)

Recommandé (pas obligatoire)

- Notions de base en Machine Learning
- Expérience avec les APIs REST
- Git pour récupérer le code des exercices

Ce que vous n'avez PAS besoin de savoir

- Deep Learning / Réseaux de neurones
- Les maths derrière les Transformers
- Comment entraîner un LLM

Ce livre est sur l'**utilisation** des LLM, pas sur leur construction.

Structure du livre

Partie 1 : Démarrage Rapide (Chapitres 1-3)

Objectif : Avoir un agent fonctionnel en moins de 2 heures. - Chapitre 1 : Votre premier agent qui marche - Chapitre 2 : Configurer le LLM pour éviter les factures surprises - Chapitre 3 : Les templates de prompts qui fonctionnent

Partie 2 : Les Briques (Chapitres 4-6)

Objectif : Maîtriser les composants essentiels. - Chapitre 4 : Les outils (le code complet) - Chapitre 5 : La mémoire (RAG intelligent) - Chapitre 6 : Le raisonnement (pas la théorie, les patterns)

Partie 3 : L'Écosystème (Chapitres 7-8)

Objectif : Choisir les bons outils sans se tromper. - Chapitre 7 : LangChain vs LlamaIndex vs CrewAI (la vraie comparaison) - Chapitre 8 : Plugins et MCP (l'avenir des agents)

Partie 4 : Le Projet (Chapitres 9-10)

Objectif : Un agent complet de A à Z. - Chapitre 9 : Conception de l'Analys-Bot Financier - Chapitre 10 : Implémentation complète (code, tests, déploiement)

Partie 5 : La Production (Chapitres 11-17)

Objectif : Passer du prototype au produit. - Chapitre 11 : Tracer et débugger un agent - Chapitre 12 : Sécurité (les attaques, les défenses) - Chapitre 13 : Multi-agents (quand un seul ne suffit plus) - Chapitre 14-16 : Optimisations avancées - Chapitre 17 : Ce qui arrive en 2025

Accompagnement

Le Code Source

Tout le code du livre est disponible sur GitHub :

```
github.com/[auteur]/agent-llm-python
```

Chaque chapitre a son dossier avec : - Le code complet et commenté - Les tests unitaires - Un `README.md` avec les instructions

Les Mises à Jour

Les APIs évoluent vite. Le repo GitHub est mis à jour régulièrement avec : - Les changements d'API (OpenAI, Anthropic, LangChain) - Les nouveaux modèles et leurs configurations - Les corrections de bugs signalés par les lecteurs

Note de l'auteur

"J'ai construit mon premier agent LLM il y a 18 mois. Il ne marchait pas. Le deuxième non plus. Le troisième a presque fonctionné, mais il coûtait une fortune.

Ce livre contient tout ce que j'aurais voulu savoir avant de commencer. Pas la théorie que vous trouvez dans les papiers de recherche. Les solutions aux vrais problèmes que vous allez rencontrer.

Si après avoir lu ce livre vous évitez ne serait-ce qu'une erreur à $100, il aura été rentable."

— Patrice Huetz

Commencez Maintenant

Passez directement au Chapitre 1 pour construire votre premier agent en 30 minutes.

Sommaire →

CHAPITRE 1

Votre Premier Agent en 30 Minutes

1. Le Problème

"Créer un agent LLM" semble simple. En pratique : boucle infinie à $847, agent qui casse vos fichiers, hallucinations d'outils inexistants.

L'erreur classique : Construire un agent alors qu'un simple prompt suffirait.

Question	OUI = Agent	NON = Prompt
Appeler des APIs / lire des fichiers ?	✅	❌
Vérifier son travail (tests, lint) ?	✅	❌
Nombre d'étapes variable ?	✅	❌
Action concrète (pas juste texte) ?	✅	❌

Score : 0-1 OUI = prompt simple. 2-3 = agent léger. 4 = agent complet.

2. La Solution Rapide : Agent Minimal

```
from dataclasses import dataclass
from typing import Callable
import json

@dataclass
class Tool:
    name: str
    description: str
    function: Callable[..., str]
    parameters: dict  # JSON Schema

class MinimalAgent:
    """Agent avec les 3 garde-fous essentiels."""

    def __init__(self, llm_client, tools: list[Tool], sy
stem_prompt: str, max_iterations: int = 15):
        self.llm = llm_client
        self.tools = {t.name: t for t in tools}
        self.system_prompt = system_prompt
        self.max_iterations = max_iterations
        self.history: list[dict] = []

    def run(self, user_message: str) -> str:
         self.history.append({"role": "user", "content":
user_message})

        for iteration in range(self.max_iterations):
            # 1. Demander au LLM
            response = self._call_llm()
            action = self._parse_response(response)

            # 2. Réponse finale ?
            if action["type"] == "final_answer":
                return action["content"]

            # 3. Exécuter l'outil
            if action["type"] == "tool_call":
                 result = self._execute_tool(action["tool
_name"], action["tool_args"])
                 self.history.append({"role": "tool", "co
ntent": f"Résultat: {result}"})
```