FICHE DE COURS

Formation Professionnelle en

AGENTIC AI

Conception, orchestration et mise en production d'agents IA

Programme sur 3 mois • 12 séances de 3 h • 36 heures

Syllabus complet — Modules, contenus & laboratoires pratiques

Document pédagogique de référence
Version 1.0 — Mai 2026

Table des matières

1. Présentation de la formation

Cette formation de trois mois, à raison d'une séance de 3 heures par semaine, forme des praticiens capables de concevoir, développer, évaluer et déployer des agents IA en production. Elle combine des fondations théoriques ciblées et une pédagogie résolument pratique : chaque séance amorce un laboratoire concret, finalisé en travail personnel, produisant un livrable réutilisable dans le projet final.

Le programme suit une progression logique en quatre modules — des fondamentaux des LLM jusqu'à la mise en production sécurisée — et culmine avec un projet Capstone déployé et soutenu devant un jury.

Informations clés

Intitulé	Formation Professionnelle en Agentic AI
Durée	3 mois — 12 séances — 36 heures en présentiel/synchrone
Rythme	1 séance de 3 heures par semaine + travail personnel
Format	Présentiel ou distanciel synchrone, fortement orienté pratique
Langue	Français (ressources techniques en anglais)
Effectif conseillé	8 à 16 participants
Évaluation	Labs notés (40 %) + Projet Capstone (50 %) + participation (10 %)
Certification	Attestation de réussite délivrée sur validation du Capstone

2. Public cible & prérequis

Public cible

- Développeurs et ingénieurs logiciels souhaitant se spécialiser dans l'IA agentique.
- Data scientists et ingénieurs ML voulant passer du modèle à l'agent en production.
- Architectes techniques et tech leads évaluant des solutions agentiques.
- Product managers techniques impliqués dans des produits à base d'agents.

Prérequis

- **Programmation :** maîtrise de Python (fonctions, classes, gestion d'environnement, packages).
- **Bases techniques :** notions d'API REST, JSON, ligne de commande, Git.
- **IA générative :** exposition préalable aux LLM souhaitée mais non obligatoire.
- **Matériel :** ordinateur capable d'exécuter Python 3.11+ et un accès Internet stable.

> **Engagement attendu**
>
> • Environ 4 à 6 heures de travail personnel par semaine : les laboratoires sont amorcés en séance puis finalisés en autonomie.
>
> • Chaque lab produit un livrable versionné qui alimente le projet Capstone.
>
> • La participation active aux revues de code et soutenances fait partie de l'évaluation.

3. Objectifs pédagogiques & compétences visées

À l'issue de la formation, le participant sera capable de :

1. Concevoir l'architecture d'un agent adaptée à un cas d'usage donné.
2. Implémenter le tool-calling, la mémoire et la récupération (RAG) de façon fiable.
3. Construire des systèmes multi-agents orchestrés et raisonner sur leur pertinence.
4. Maîtriser au moins deux frameworks d'agents et choisir avec discernement.
5. Évaluer un agent avec des métriques rigoureuses et une observabilité complète.
6. Sécuriser un agent (guardrails, moindre privilège) et optimiser ses coûts.
7. Déployer un agent en production avec monitoring et plan d'amélioration.

Cartographie des modules

Module	Séances	Thème	Compétence clé acquise
Module 1	1 – 3	Fondamentaux	LLM, prompting, tool-calling
Module 2	4 – 6	Architectures d'agents	ReAct, mémoire, RAG
Module 3	7 – 9	Frameworks & multi-agents	LangGraph, CrewAI, orchestration
Module 4	10 – 12	Production & déploiement	Évaluation, sécurité, mise en prod

4. Plan détaillé des 12 séances

Chaque séance de 3 heures est structurée de façon identique : objectifs pédagogiques, contenu théorique, laboratoire pratique avec livrable, puis ressources et lectures. Le temps en séance est consacré à la théorie et au démarrage guidé du lab ; la finalisation du livrable se fait en travail personnel. Les laboratoires sont cumulatifs et préparent progressivement le projet Capstone.

Séance 1 — Introduction à l'Agentic AI & fondations des LLM

Module : Module 1 — Fondamentaux | Durée : 3 h (1 h 30 théorie / 1 h 30 lab) | Niveau : Débutant

Objectifs pédagogiques

- Définir ce qu'est un agent IA et le distinguer d'un simple chatbot ou d'un pipeline LLM.
- Comprendre la boucle perception → raisonnement → action → observation.
- Maîtriser le fonctionnement des LLM : tokenisation, fenêtre de contexte, température, paramètres de génération.
- Identifier les cas d'usage pertinents (et non pertinents) pour une approche agentique.

Contenu théorique

- Panorama de l'IA agentique : du prompt unique aux systèmes autonomes ; définitions, taxonomie (assistants, copilotes, agents autonomes).
- Anatomie d'un LLM : architecture transformer (vue d'ensemble), embeddings, inférence, coût et latence.
- Niveaux d'autonomie : assisté, semi-autonome, autonome avec supervision humaine (human-in-the-loop).
- Composants d'un agent : modèle, instructions/persona, outils, mémoire, boucle de contrôle.
- Limites et risques : hallucination, dérive d'objectif, dépendance aux données, coûts.

Lab pratique

> **Lab 1 — Premier agent minimal**
> - Mettre en place l'environnement Python (venv, clés API, gestion des secrets).
> - Appeler un LLM via API et explorer les paramètres (température, max_tokens, top_p).
> - Construire une boucle agentique « naïve » : question → réponse → reformulation.
> - Mesurer latence et coût par requête, journaliser les échanges.

Livrable attendu : Notebook fonctionnel d'un mini-agent conversationnel avec journalisation des coûts.

Stack technique : Python 3.11+, SDK Anthropic / OpenAI, Jupyter, python-dotenv

Ressources & lectures

- « A practical guide to building agents » (Anthropic / OpenAI docs).
- Documentation officielle de l'API du fournisseur LLM utilisé.
- Article fondateur : « Attention Is All You Need » (lecture conceptuelle, non mathématique).

Séance 2 — Prompt engineering avancé & sorties structurées

Module : Module 1 — Fondamentaux | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Débutant

Objectifs pédagogiques

- Concevoir des prompts robustes : rôle, contexte, contraintes, exemples (few-shot).
- Forcer des sorties structurées et validables (JSON, schémas).
- Appliquer le raisonnement étape par étape (chain-of-thought, décomposition).
- Gérer les erreurs de format et mettre en place des stratégies de re-prompting.

Contenu théorique

- Anatomie d'un prompt système efficace : instructions positives/négatives, ton, garde-fous.
- Techniques : zero-shot, few-shot, chain-of-thought, self-consistency, prompt chaining.
- Sorties structurées : JSON mode, schémas (Pydantic / JSON Schema), function-calling comme contrat.
- Évaluation de prompts : jeux de tests, métriques, itération systématique.
- Anti-patterns : prompts trop longs, instructions contradictoires, sur-spécification.

Lab pratique

> **Lab 2 — Extraction structurée fiable**
>
> - Construire un extracteur d'information transformant un texte libre en JSON validé par schéma.
> - Ajouter une couche de validation Pydantic et un mécanisme de réessai automatique.
> - Comparer plusieurs variantes de prompts sur un jeu de test de 20 cas.
> - Produire un tableau de scores (précision, taux d'échec de format).

Livrable attendu : Module Python d'extraction structurée + rapport comparatif des prompts.

Stack technique : Python, Pydantic, instructor / structured outputs, pytest

Ressources & lectures

- Guide officiel de prompt engineering du fournisseur LLM.
- Documentation Pydantic v2 (validation de schémas).
- Référentiel de patterns de prompts (cookbook officiel).

Séance 3 — Tool use & function calling

Module : Module 1 — Fondamentaux | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Débutant à intermédiaire

Objectifs pédagogiques

- Comprendre le mécanisme de tool/function calling de bout en bout.
- Concevoir des outils sûrs, idempotents et bien documentés pour un agent.
- Orchestrer plusieurs appels d'outils dans une même boucle de raisonnement.
- Gérer les erreurs d'outils et les boucles infinies.

Contenu théorique

- Le contrat d'un outil : nom, description, schéma d'arguments, valeur de retour.
- Cycle complet : intention du modèle → appel d'outil → exécution → observation → suite.
- Bonnes pratiques de conception d'outils : granularité, déterminisme, messages d'erreur exploitables.
- Sécurité des outils : validation des entrées, sandboxing, principe du moindre privilège.
- Limites : nombre d'outils, ambiguïté de sélection, coûts de contexte.

Lab pratique

> **Lab 3 — Agent outillé (calculatrice, recherche, fichiers)**
> - Définir 3 à 4 outils (calcul, recherche web simulée, lecture/écriture de fichiers).
> - Implémenter la boucle de tool-calling avec gestion d'erreurs et limite d'itérations.
> - Ajouter une journalisation des appels d'outils (trace lisible).
> - Tester l'agent sur des tâches multi-étapes nécessitant plusieurs outils.

Livrable attendu : Agent outillé avec 3+ outils, traces d'exécution et tests.

Stack technique : Python, SDK LLM avec tool-calling, requests, logging

Ressources & lectures

- Documentation officielle « tool use / function calling ».
- Patterns de conception d'outils pour agents (cookbook).
- Article : « ReAct: Synergizing Reasoning and Acting » (introduction).

Séance 4 — Architectures d'agents : ReAct, Plan-and-Execute, Reflexion

Module : Module 2 — Architectures d'agents | Durée : 3 h (1 h 30 théorie / 1 h 30 lab) | Niveau : Intermédiaire

Objectifs pédagogiques

- Comparer les principales architectures de raisonnement agentique.
- Implémenter le pattern ReAct (raisonnement + action entrelacés).
- Mettre en œuvre un agent planificateur (décomposition de tâches).
- Ajouter une boucle d'auto-critique et de réflexion (Reflexion).

Contenu théorique

- ReAct : alternance pensée/action/observation ; forces et limites.
- Plan-and-Execute : planification globale puis exécution séquentielle.
- Reflexion & self-refine : critique itérative, mémoire d'erreurs.
- Tree/Graph of Thoughts : exploration de plusieurs chemins de raisonnement.
- Critères de choix d'architecture selon la tâche (déterministe vs exploratoire).

Lab pratique

> **Lab 4 — Comparaison d'architectures sur une même tâche**
> - Implémenter trois agents (ReAct, Plan-and-Execute, Reflexion) résolvant le même problème.
> - Mettre en place un harnais d'évaluation commun (succès, étapes, coût).
> - Analyser les traces pour comprendre les modes d'échec de chacun.
> - Rédiger une recommandation argumentée d'architecture.

Livrable attendu : Trois agents comparables + note d'analyse argumentée (1-2 pages).

Stack technique : Python, LangGraph (ou implémentation maison), pandas pour l'analyse

Ressources & lectures

- Articles : ReAct, Reflexion, Plan-and-Solve (résumés conceptuels).
- Documentation LangGraph (graphes d'états d'agents).
- Études de cas publiques d'architectures agentiques.

Séance 5 — Mémoire & gestion d'état

Module : Module 2 — Architectures d'agents | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Intermédiaire

Objectifs pédagogiques

- Distinguer mémoire à court terme (contexte) et à long terme (persistante).
- Implémenter une mémoire vectorielle et une mémoire épisodique.
- Gérer la compression et le résumé de l'historique de conversation.
- Concevoir un état d'agent persistant et reprenable.

Contenu théorique

- Types de mémoire : tampon, résumé, vectorielle, épisodique, sémantique.
- Stratégies de troncature et de résumé du contexte (sliding window, summarization).
- Stores de mémoire : bases vectorielles, clé-valeur, bases relationnelles.
- Gestion d'état : checkpoints, reprise après interruption, threads de conversation.
- Risques : fuite de contexte, mémoire obsolète, coûts de stockage et de relecture.

Lab pratique

> **Lab 5 — Agent à mémoire long terme**
>
> - Intégrer une base vectorielle pour la mémoire à long terme.
> - Implémenter un résumé automatique de l'historique au-delà d'un seuil.
> - Ajouter des checkpoints d'état permettant de reprendre une session.
> - Démontrer la persistance sur deux sessions distinctes.

Livrable attendu : Agent conversationnel mémoire persistante avec démonstration multi-session.

Stack technique : Python, base vectorielle (Chroma/FAISS/pgvector), LangGraph checkpoints

Ressources & lectures

- Documentation de la base vectorielle choisie.
- Patterns de gestion de mémoire d'agents (LangChain/LangGraph memory).
- Article sur la mémoire à long terme des agents conversationnels.

Séance 6 — RAG pour agents (Retrieval-Augmented Generation)

Module : Module 2 — Architectures d'agents | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Intermédiaire

Objectifs pédagogiques

- Construire un pipeline RAG complet : ingestion, chunking, indexation, récupération.
- Intégrer la récupération comme un outil agentique (agentic RAG).
- Améliorer la pertinence : reranking, requêtes multiples, citations.
- Évaluer la qualité d'un système RAG.

Contenu théorique

- Pipeline RAG : extraction, découpage, embeddings, index, recherche, génération.
- Stratégies de chunking et leur impact sur la pertinence.
- Agentic RAG : l'agent décide quoi et quand récupérer, requêtes multi-tours.
- Amélioration : reranking, HyDE, query rewriting, fusion de sources.
- Évaluation RAG : fidélité, pertinence du contexte, exactitude des citations.

Lab pratique

> **Lab 6 — Assistant documentaire agentique**
>
> - Ingérer un corpus de documents (PDF/Markdown) et construire l'index vectoriel.
> - Exposer la recherche comme un outil que l'agent appelle de façon autonome.
> - Ajouter un mécanisme de citations vérifiables dans les réponses.
> - Évaluer le système sur un jeu de questions de référence.

Livrable attendu : Assistant RAG agentique avec citations + rapport d'évaluation.

Stack technique : Python, base vectorielle, embeddings, framework RAG, jeu d'évaluation

Ressources & lectures

- Guides RAG officiels (cookbooks fournisseurs).
- Documentation d'un framework d'évaluation RAG (p. ex. Ragas).
- Articles : RAG, HyDE, reranking (résumés).

Séance 7 — Frameworks d'agents I — LangChain & LangGraph

Module : Module 3 — Frameworks & multi-agents | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Intermédiaire à avancé

Objectifs pédagogiques

- Maîtriser les abstractions clés de LangChain (modèles, outils, chaînes).
- Construire des agents comme graphes d'états avec LangGraph.
- Implémenter des branches conditionnelles, cycles et human-in-the-loop.
- Comprendre quand utiliser un framework vs une implémentation maison.

Contenu théorique

- Écosystème LangChain : composants, interfaces, intégrations.
- LangGraph : nœuds, arêtes, état partagé, persistance, interruptions.
- Patterns : routeur, superviseur, boucle de validation humaine.
- Compromis des frameworks : vitesse de développement vs contrôle et dette.
- Bonnes pratiques de structuration de projet d'agent.

Lab pratique

> **Lab 7 — Agent métier avec LangGraph**
>
> - Modéliser un workflow métier en graphe d'états (nœuds et transitions).
> - Ajouter une étape de validation humaine (human-in-the-loop) avant action sensible.
> - Implémenter la persistance d'état et la reprise.
> - Tester les chemins nominaux et les chemins d'erreur.

Livrable attendu : Agent métier basé sur un graphe d'états documenté.

Stack technique : Python, LangChain, LangGraph, persistance (SQLite/Postgres)

Ressources & lectures

- Documentation officielle LangChain et LangGraph.
- Tutoriels officiels d'agents LangGraph.
- Exemples de référence (templates d'agents).

Séance 8 — Frameworks d'agents II — CrewAI, AutoGen, Agents SDK

Module : Module 3 — Frameworks & multi-agents | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Intermédiaire à avancé

Objectifs pédagogiques

- Comparer plusieurs frameworks d'agents et leurs philosophies.
- Construire une équipe d'agents avec rôles et tâches (CrewAI).
- Mettre en place des conversations multi-agents (AutoGen).
- Choisir le bon framework selon le besoin du projet.

Contenu théorique

- CrewAI : agents, rôles, tâches, processus séquentiel/hiérarchique.
- AutoGen : agents conversationnels, groupes, orchestration par dialogue.
- Agents SDK / frameworks légers : handoffs, guardrails intégrés.
- Grille de comparaison : courbe d'apprentissage, contrôle, observabilité, communauté.
- Critères de décision et risques de verrouillage technologique.

Lab pratique

> **Lab 8 — Même cas d'usage, deux frameworks**
> - Implémenter une équipe d'agents (recherche + rédaction + revue) avec CrewAI.
> - Réimplémenter le même cas avec un second framework au choix.
> - Comparer code, contrôle, observabilité et coût.
> - Rédiger une recommandation de framework pour le projet final.

Livrable attendu : Deux implémentations comparables + grille de décision argumentée.

Stack technique : Python, CrewAI, AutoGen ou Agents SDK

Ressources & lectures

- Documentation officielle CrewAI et AutoGen.
- Comparatifs de frameworks d'agents (sources récentes).
- Exemples officiels d'équipes d'agents.

Séance 9 — Systèmes multi-agents & patterns d'orchestration

Module : Module 3 — Frameworks & multi-agents | Durée : 3 h (1 h 30 théorie / 1 h 30 lab) | Niveau : Avancé

Objectifs pédagogiques

- Concevoir des architectures multi-agents (superviseur, pairs, hiérarchique).
- Définir des protocoles de communication et de délégation entre agents.
- Gérer les conflits, la coordination et l'arrêt d'un système multi-agents.
- Évaluer quand le multi-agents apporte (ou non) de la valeur.

Contenu théorique

- Patterns : orchestrateur-travailleurs, hiérarchique, marché/enchères, débat.
- Communication : passage de messages, mémoire partagée, handoffs.
- Coordination : décomposition de tâches, agrégation de résultats, consensus.
- Modes d'échec : boucles inter-agents, explosion des coûts, diffusion d'erreurs.
- Quand préférer un agent unique bien outillé à un système multi-agents.

Lab pratique

> **Lab 9 — Système multi-agents orchestré**
> - Concevoir un système superviseur + 2-3 agents spécialisés.
> - Implémenter le protocole de délégation et d'agrégation des résultats.
> - Ajouter des garde-fous anti-boucle et un budget d'itérations/coût.
> - Évaluer la qualité globale vs un agent unique de référence.

Livrable attendu : Système multi-agents fonctionnel + analyse comparative vs agent unique.

Stack technique : Python, LangGraph ou CrewAI, observabilité des échanges

Ressources & lectures

- Articles et guides sur les architectures multi-agents.
- Documentation des patterns d'orchestration des frameworks.
- Études de cas industrielles de systèmes multi-agents.

Séance 10 — Évaluation, observabilité & debugging

Module : Module 4 — Production & déploiement | Durée : 3 h (1 h théorie / 2 h lab) | Niveau : Avancé

Objectifs pédagogiques

- Construire un jeu d'évaluation et des métriques pour agents.
- Mettre en place le tracing complet (LLM, outils, étapes).
- Détecter et diagnostiquer les modes d'échec récurrents.
- Intégrer l'évaluation dans une boucle d'amélioration continue.

Contenu théorique

- Évaluation des agents : succès de tâche, qualité, coût, latence, robustesse.
- Datasets d'évaluation : construction, jeux adverses, régression.
- Observabilité : tracing distribué, spans, attributs, coûts par étape.
- LLM-as-a-judge : forces, biais, calibration.
- Debugging systématique : rejouer les traces, isoler les composants.

Lab pratique

> **Lab 10 — Harnais d'évaluation & tracing**
>
> - Instrumenter un agent existant avec une plateforme de tracing.
> - Construire un dataset d'évaluation de 25-50 cas avec attendus.
> - Mettre en place un évaluateur automatique (LLM-as-a-judge + règles).
> - Produire un tableau de bord de performance et de coûts.

Livrable attendu : Pipeline d'évaluation reproductible + tableau de bord de résultats.

Stack technique : Python, LangSmith / Phoenix / Langfuse, dataset d'évaluation

Ressources & lectures

- Documentation de la plateforme d'observabilité choisie.
- Guides d'évaluation d'agents (cookbooks fournisseurs).
- Article sur LLM-as-a-judge et ses limites.

Séance 11 — Sécurité, guardrails, gouvernance & optimisation des coûts

Module : Module 4 — Production & déploiement | Durée : 3 h (1 h 30 théorie / 1 h 30 lab) | Niveau : Avancé

Objectifs pédagogiques

- Identifier les risques de sécurité propres aux agents (injection, exfiltration).
- Mettre en place des guardrails d'entrée et de sortie.
- Appliquer le principe du moindre privilège aux outils et données.
- Optimiser les coûts et la latence sans dégrader la qualité.

Contenu théorique

- Surface d'attaque des agents : prompt injection (directe/indirecte), outils malveillants.
- Guardrails : filtrage d'entrée/sortie, validation, modération, allow-lists.
- Confidentialité & gouvernance : minimisation des données, journalisation, traçabilité, conformité.
- Human-in-the-loop pour actions à fort impact ; approbations.
- Optimisation : choix de modèle, mise en cache, routage, réduction de contexte.

Lab pratique

> **Lab 11 — Durcissement d'un agent**
> - Auditer un agent et lister ses vulnérabilités (injection indirecte via documents).
> - Implémenter des guardrails d'entrée/sortie et une allow-list d'outils.
> - Ajouter une étape d'approbation humaine pour les actions sensibles.
> - Mesurer l'impact des optimisations (cache, routage) sur coût et latence.

Livrable attendu : Agent durci + rapport de sécurité et d'optimisation des coûts.

Stack technique : Python, bibliothèque de guardrails, cache, métriques de coût

Ressources & lectures

- OWASP Top 10 pour applications LLM.
- Guides officiels de sécurité et de guardrails des agents.
- Bonnes pratiques de réduction des coûts LLM.

Séance 12 — Déploiement en production & soutenance du projet Capstone

Module : Module 4 — Production & déploiement | Durée : 3 h (30 min cours / 2 h 30 projet & soutenances) | Niveau : Avancé

Objectifs pédagogiques

- Déployer un agent comme service (API) avec configuration et secrets.
- Mettre en place le monitoring, les alertes et la gestion des incidents.
- Présenter et défendre le projet Capstone devant un jury.
- Établir une feuille de route d'amélioration post-formation.

Contenu théorique

- Architecture de déploiement : API, files d'attente, conteneurisation, scaling.
- Gestion de configuration, secrets, versioning de prompts et de modèles.
- Monitoring en production : santé, coûts, dérive, alertes, runbooks.
- Boucle d'amélioration continue : retours utilisateurs, ré-évaluation, itération.
- Synthèse de la formation et perspectives (tendances de l'écosystème).

Lab pratique

> **Lab 12 — Mise en production & soutenance**
> - Conteneuriser l'agent du projet et l'exposer via une API documentée.
> - Activer monitoring, traces et alertes de coûts.
> - Présenter le projet Capstone (démo + architecture + résultats d'évaluation).
> - Recueillir le feedback du jury et formaliser la feuille de route.

Livrable attendu : Projet Capstone déployé + présentation + feuille de route.

Stack technique : Python, FastAPI, Docker, plateforme d'observabilité, fournisseur cloud

Ressources & lectures

- Guides de mise en production d'applications LLM.
- Patterns de déploiement et de monitoring d'agents.
- Checklists de readiness production.

5. Projet Capstone

Le projet Capstone est le fil rouge de la formation. Démarré dès le Module 2, il consiste à concevoir et déployer un agent (ou système multi-agents) résolvant un problème réel choisi par le participant ou imposé par un sujet.

Jalons du projet

Jalon	Échéance	Livrable
J1 — Cadrage	Séance 4	Note de cadrage : problème, périmètre, architecture cible
J2 — Prototype	Séance 7	Agent fonctionnel minimal avec outils et mémoire
J3 — Évaluation	Séance 10	Jeu d'évaluation + résultats et observabilité
J4 — Durcissement	Séance 11	Guardrails, sécurité et optimisation des coûts
J5 — Soutenance	Séance 12	Agent déployé + démo + présentation + feuille de route

Critères d'évaluation du Capstone

Critère	Poids	Description
Pertinence & conception	20 %	Adéquation de l'architecture au problème
Qualité technique	25 %	Code, robustesse, gestion d'erreurs, tests
Évaluation & observabilité	20 %	Rigueur des métriques et du tracing
Sécurité & coûts	15 %	Guardrails, moindre privilège, optimisation
Déploiement & démo	10 %	Mise en production effective et fonctionnelle
Présentation	10 %	Clarté, argumentation, réponses au jury

6. Modalités d'évaluation

L'évaluation est continue et orientée production de livrables. Elle combine la qualité des laboratoires hebdomadaires, le projet Capstone et la participation.

Composante	Pondération	Modalité
Laboratoires (12)	40 %	Notation des livrables hebdomadaires
Projet Capstone	50 %	Jalons + soutenance finale devant jury
Participation	10 %	Revues de code, entraide, contributions

Conditions de réussite

- Obtenir une moyenne générale ≥ 60 %.
- Valider au moins 9 des 12 laboratoires.
- Présenter et défendre le projet Capstone déployé.

7. Environnement technique & outils

Les outils sont sélectionnés pour leur pertinence pédagogique. Des alternatives équivalentes sont acceptées si elles couvrent les mêmes concepts.

Catégorie	Outils recommandés
Langage & environnement	Python 3.11+, venv/poetry, Jupyter, Git
Accès LLM	API d'un fournisseur LLM majeur (clé fournie par le participant)
Frameworks d'agents	LangChain, LangGraph, CrewAI, AutoGen / Agents SDK
Mémoire & RAG	Base vectorielle (Chroma, FAISS ou pgvector), embeddings
Évaluation & tracing	LangSmith, Phoenix ou Langfuse ; Ragas pour le RAG
Sécurité	Bibliothèque de guardrails, validation de schémas (Pydantic)
Déploiement	FastAPI, Docker, un fournisseur cloud au choix

Note importante

- L'écosystème de l'IA agentique évolue très rapidement : les versions exactes des bibliothèques et certaines API peuvent changer.
- Les concepts enseignés (boucles agentiques, mémoire, orchestration, évaluation, sécurité) restent stables et transférables.
- Vérifier systématiquement la documentation officielle à jour de chaque outil avant les labs.

8. Ressources pédagogiques complémentaires

Lectures fondatrices (concepts)

- ReAct : raisonnement et action entrelacés dans les agents.
- Reflexion / Self-Refine : auto-amélioration itérative.
- RAG : génération augmentée par récupération.
- Architectures multi-agents et patterns d'orchestration.

Documentation & guides pratiques

- Guides « building agents » et cookbooks des principaux fournisseurs de LLM.
- Documentation officielle des frameworks (LangChain/LangGraph, CrewAI, AutoGen).
- OWASP Top 10 pour applications LLM (sécurité).
- Documentation des plateformes d'observabilité (tracing et évaluation).

Pratique continue

- Reproduire chaque lab dans un dépôt Git personnel et documenté.
- Tenir un journal de bord technique des décisions d'architecture.
- Participer à des communautés et suivre les évolutions de l'écosystème.

Fin du document — Fiche de cours Agentic AI