Sujet de projet
Agentification et motifs de conception agentiques

Projet par groupe de 4 étudiant.e.s
Date de rendu : 20 dec. 2025

Résumé

Les agents, ou systèmes agentiques, sont des programmes autonomes capables de prendre des décisions, de planifier des actions et d'interagir avec des outils ou des utilisateurs. Ils constituent un élément central des architectures d'IA modernes, représentant une évolution significative par rapport aux systèmes classiques qui se contentent de répondre à des requêtes isolées. Un agent peut percevoir son environnement, maintenir un état interne, raisonner sur les actions à entreprendre, et exécuter ces actions de manière séquentielle ou itérative jusqu'à atteindre son objectif.

Dans ce projet, réalisé **en groupes de quatre étudiants**, vous explorerez la transformation, ou *agentification*, d'un service ou d'un workflow en un système composé d'un ou plusieurs agents. Cette démarche d'agentification consiste à identifier les étapes d'un processus qui peuvent bénéficier d'une autonomie décisionnelle, d'une capacité de planification ou d'une interaction avec des ressources externes. Vous appliquerez des motifs de conception agentiques abordés en cours, qui sont des patrons architecturaux éprouvés permettant de structurer efficacement le comportement et les interactions des agents.

Les objectifs pédagogiques sont doubles. D'une part, vous devez maîtriser l'application de ces motifs dans une implémentation concrète, en comprenant non seulement leur fonctionnement théorique mais aussi leurs implications pratiques. D'autre part, vous mesurerez quantitativement l'impact des choix architecturaux sur les performances du système, y compris les coûts en tokens associés à l'utilisation de modèles de langage. Cette dimension économique est cruciale dans un contexte professionnel où l'efficacité d'un système se mesure aussi à sa viabilité financière.

1 Objectifs pédagogiques

Le projet vise à développer plusieurs compétences clés qui vous prépareront à concevoir et déployer des systèmes agentiques dans un cadre professionnel.

Premièrement, vous devrez comprendre et appliquer des motifs de conception agentiques tels que Planner–Executor, réflexion, module de mémoire, délégation, agents hiérarchiques et utilisation d'outils. Ces motifs ne sont pas de simples recettes à appliquer mécaniquement, mais des paradigmes qui structurent la manière dont un agent perçoit, raisonne et agit. Par exemple, le motif Planner–Executor sépare clairement la phase de planification stratégique de l'exécution tactique, ce qui améliore la traçabilité et permet d'adapter le plan en cours de route.

Deuxièmement, vous concevrez et implémenterez un prototype fonctionnel d'un service agentifié adapté à un cas d'usage réaliste. L'accent est mis sur la fonctionnalité et la robustesse plutôt que sur la sophistication technologique. Un prototype simple mais fiable qui résout effectivement un problème concret vaut mieux qu'un système complexe qui échoue fréquemment.

Troisièmement, vous définirez des métriques d'évaluation objectives pour effectuer une analyse quantitative. Ces métriques peuvent inclure la précision des réponses, le taux de succès des tâches accomplies, la latence du système, sa robustesse face aux erreurs, et bien sûr les coûts en tokens. L'objectif est de pouvoir comparer différentes approches de manière scientifique et reproductible.

Quatrièmement, vous mettrez en place un suivi reproductible des coûts via des logs détaillés par appel d'API ou de modèle, et vous produirez une analyse économique. Dans un environnement où l'utilisation de modèles de langage est facturée au token, il est essentiel de pouvoir quantifier précisément combien coûte chaque interaction et d'identifier les opportunités d'optimisation.

Enfin, vous rédigerez un rapport didactique et préparerez une démonstration qui permettront de communiquer vos résultats de manière claire et convaincante à un public technique.

2 Consignes générales

Le projet se réalise en groupes de **quatre étudiants**. La collaboration est un aspect fondamental de ce travail, et chaque membre doit contribuer à la conception, à l'implémentation et à l'évaluation.

Une répartition équitable des tâches est attendue, et il est recommandé d'utiliser des outils de gestion de projet pour coordonner le travail de l'équipe.

Chaque groupe sélectionne un service ou un besoin utilisateur concret qu'il souhaite agentifier. Les possibilités sont nombreuses et variées. Le choix du cas d'usage doit être guidé par sa pertinence fonctionnelle et par la possibilité d'y appliquer plusieurs motifs de conception agentiques.

Vous devrez intégrer au moins trois motifs de conception vus en cours et justifier leur pertinence pour le cas choisi. Cette justification doit aller au-delà d'une simple description théorique et expliquer concrètement comment chaque motif améliore le système. Par exemple, si vous choisissez d'intégrer un module de mémoire, vous devez expliquer quelles informations sont mémorisées, comment elles sont récupérées, et en quoi cela améliore la cohérence ou l'efficacité du système.

Le prototype que vous développerez doit être opérationnel pour des scénarios d'usage représentatifs, avec une interface minimale. Cette interface peut prendre la forme d'une ligne de commande (CLI), d'une API REST accessible via des requêtes HTTP, ou d'une interface utilisateur simple. L'important est que le prototype puisse être testé et évalué de manière reproductible.

L'évaluation constitue un aspect central du projet. Vous définirez des jeux de tests et des scénarios d'usage, collecterez des mesures quantitatives. Cette comparaison peut porter sur différents aspects, par exemple un agent simple versus un agent équipé d'un module de mémoire, ou encore un planificateur centralisé versus une architecture où plusieurs agents délégués collaborent. L'objectif est de démontrer empiriquement l'impact de vos choix architecturaux.

Le suivi des coûts en tokens est obligatoire et doit être rigoureux. Vous enregistrerez tous les appels aux modèles de langage, en calculant précisément le nombre de tokens utilisés dans les prompts, les complétions et au total. Ces données seront compilées dans une table récapitulative et visualisées sous forme de graphiques pour faciliter l'analyse et l'identification des goulots d'étranglement.

Enfin, le reporting comprend plusieurs éléments complémentaires : un rapport PDF structuré et détaillé, le code source hébergé dans un dépôt Git avec des instructions claires, une démonstration vidéo de cinq à dix minutes montrant le prototype en action, des slides pour une soutenance orale de dix minutes, et un fichier de coûts au format CSV contenant toutes les données de consommation de tokens.

3 Livrables attendus

Le code source sera hébergé sur la plateforme de gestion de versions GitLab. Il doit être accompagné d'un fichier README complet qui explique l'architecture du projet, les prérequis d'installation, les commandes pour exécuter le système, et des exemples d'utilisation. Le code doit être structuré, commenté et respecter les bonnes pratiques de développement.

Le rapport PDF, d'une longueur de huit à douze pages, doit couvrir l'ensemble du projet de manière didactique. Il commencera par une présentation de l'architecture du système avec des diagrammes explicatifs, suivie d'une description détaillée des motifs de conception utilisés et de leur intégration. Le protocole d'évaluation sera ensuite exposé, en précisant les métriques choisies et la méthodologie employée. Les résultats seront présentés sous forme de tableaux et de graphiques, accompagnés d'une discussion qui analyse les forces et faiblesses de votre approche. Enfin, une section sur les limitations et les perspectives d'amélioration conclura le rapport.

Le fichier de suivi des coûts, nommé `costs.csv` ou `costs.json`, doit inclure au minimum les champs suivants : l'horodatage de chaque appel (`timestamp`), l'identifiant du scénario testé (`scenario_id`), l'identifiant unique de l'appel (`call_id`), le nom du modèle utilisé (`model`), l'endpoint ou le service appelé (`endpoint`), le nombre de tokens dans le prompt (`prompt_tokens`), le nombre de tokens dans la complétion (`completion_tokens`), le total de tokens (`total_tokens`), et éventuellement des notes ou métadonnées supplémentaires (`notes`). Ce fichier est essentiel pour l'analyse économique et doit être généré automatiquement par votre système de logging.

Un script d'analyse, sous forme de notebook Jupyter ou de script Python, doit accompagner le fichier de coûts. Ce script agrège les données, calcule des statistiques descriptives, produit des métriques synthétiques par scénario ou par variante architecturale, et génère des visualisations comme des histogrammes de distribution des tokens, des courbes cumulées, ou des comparaisons par boxplot.

La vidéo de démonstration, d'une durée de cinq à dix minutes, présente le prototype en action. Elle doit montrer des exemples d'utilisation concrets, illustrer le fonctionnement des différents motifs de conception, et résumer les principaux résultats de l'évaluation. Cette vidéo servira également de support pour la soutenance orale.

Les slides pour la soutenance de dix minutes doivent être concises et visuellement claires. Elles résument le contexte, les choix architecturaux, les résultats clés et les leçons apprises. La soutenance sera l'occasion de défendre vos choix techniques et de répondre aux questions sur votre méthodologie et vos résultats.

4 Contraintes techniques (suggestions)

Il est recommandé d'utiliser Python comme langage principal en raison de sa richesse en bibliothèques et de sa facilité d'intégration avec les modèles de langage. Plusieurs bibliothèques peuvent faciliter votre travail. `tiktoken` est particulièrement utile pour le comptage précis des tokens avant ou après les appels aux modèles, ce qui est essentiel pour votre analyse des coûts. Pour l'interface utilisateur, des frameworks comme `Streamlit` permettent de créer rapidement des interfaces web interactives, tandis que `FastAPI` est idéal pour construire une API REST performante et bien documentée.

Pour les modèles de langage, vous disposez de plusieurs options. **LM Studio** permet d'exécuter localement des modèles open-source, ce qui offre un contrôle total et évite les coûts d'API, mais nécessite des ressources matérielles significatives. **Ollama** est une solution conviviale pour gérer facilement des modèles comme LLaMA, Mistral ou Phi sur une machine locale, avec une installation simple et une utilisation intuitive. Enfin, l'**API d'inférence de Hugging Face** donne accès à une large gamme de modèles hébergés dans le cloud, ce qui simplifie le déploiement mais implique des coûts potentiels.

Quelle que soit la solution choisie, il est impératif de logger précisément chaque requête envoyée aux modèles. Ce logging doit capturer non seulement les tokens consommés, mais aussi le contexte de l'appel, le temps de réponse, et le statut de la requête. De plus, prévoyez des tests automatisés sur des scénarios reproductibles, ce qui vous permettra de valider votre système et de mesurer l'impact de vos modifications de manière fiable.

5 Design Pattern Agentic

Pour chaque motif de conception que vous intégrez, vous devez expliquer son rôle dans votre architecture, décrire précisément comment il est implémenté, et définir la manière dont vous mesurez son impact.

Le motif **Planner–Executor** repose sur une séparation claire entre la planification et l'exécution. Un module de planification, généralement alimenté par un modèle de langage, génère une séquence structurée de sous-tâches nécessaires pour accomplir l'objectif global. L'exécuteur prend ensuite ces sous-tâches et les réalise une par une, en vérifiant à chaque étape que le résultat est conforme aux attentes. Cette séparation améliore la traçabilité des décisions, facilite le débogage, et permet d'adapter le plan en cours de route si une étape échoue ou produit un résultat inattendu.

La **réflexion ou mécanisme de critique** introduit une boucle d'évaluation et de révision des sorties produites par l'agent. Après avoir généré une réponse ou accompli une action, l'agent soumet son résultat à un module critique qui évalue sa qualité, identifie les erreurs potentielles, et suggère des améliorations. L'agent peut alors réviser sa sortie en tenant compte de ces retours. Ce processus itératif améliore significativement la qualité des résultats, mais augmente également la consommation de tokens et la latence.

Le **module de mémoire** permet à l'agent de stocker et de récupérer des informations contextuelles provenant d'interactions passées. Cette mémoire peut être à court terme, conservant uniquement les échanges récents, ou à long terme, persistant des connaissances accumulées au fil du temps. L'utilisation d'un module de mémoire améliore la cohérence des réponses, évite la répétition d'informations déjà fournies, et permet d'optimiser les prompts en ne réinjectant que les informations pertinentes plutôt que l'intégralité de l'historique.

La **délégation et l'approche multi-agent** consistent à répartir les responsabilités entre plusieurs agents spécialisés qui communiquent entre eux pour accomplir une tâche complexe. Chaque agent se concentre sur un domaine d'expertise ou une sous-tâche spécifique, ce qui améliore la modularité et la maintenabilité du système. Par exemple, dans un assistant de recherche documentaire, un agent pourrait être responsable de la recherche d'informations, un autre de la synthèse, et un troisième de la vérification factuelle. La coordination entre ces agents nécessite un mécanisme de communication clair et des protocoles d'échange bien définis.

L'**utilisation d'outils et l'orchestration d'APIs** permettent à l'agent d'étendre ses capacités en interagissant avec des ressources externes. Ces outils peuvent inclure des moteurs de recherche, des calculatrices, des bases de données, des services de traduction, ou tout autre API pertinent pour la tâche. L'intégration sécurisée de ces outils nécessite une validation rigoureuse des entrées et sorties, une gestion appropriée des erreurs, et idéalement une abstraction qui facilite l'ajout de nouveaux outils.

Enfin, les mécanismes de **robustesse comme les fallbacks et les retries** assurent que le système peut gérer les échecs de manière gracieuse. Lorsqu'un appel à un modèle échoue, que ce soit en raison d'une indisponibilité temporaire, d'un dépassement de timeout, ou d'une réponse invalide, le système peut automatiquement réessayer l'opération, basculer vers un modèle alternatif, ou adopter une stratégie de dégradation gracieuse qui fournit une réponse partielle plutôt qu'un échec total.

6 Protocole d'évaluation quantitative

L'évaluation quantitative de votre système agentifié repose sur la définition de métriques pertinentes et leur mesure systématique sur un ensemble de scénarios représentatifs.

Le **taux de succès** mesure le pourcentage de scénarios pour lesquels le système accomplit correctement la tâche demandée. Cette métrique binaire est simple à calculer mais nécessite de définir clairement ce qu'est un succès, ce qui peut être non trivial pour des tâches ouvertes ou subjectives.

La **latence moyenne** mesure le temps moyen nécessaire pour traiter un scénario, exprimé en secondes. Cette métrique est cruciale pour l'expérience utilisateur, car un système trop lent sera perçu comme inefficace même s'il est précis.

Le nombre de **tokens par scénario** correspond à la somme des tokens utilisés dans tous les appels aux modèles pour un scénario donné. Cette métrique est directement liée au coût économique et doit être analysée en conjonction avec le taux de succès.

Le **coût estimé par scénario** est calculé en multipliant le nombre de tokens par le tarif du fournisseur de modèles. Cette estimation permet de comparer la viabilité économique de différentes approches architecturales.

Le ratio **tokens par succès** divise le nombre total de tokens consommés par le nombre de scénarios réussis. Ce ratio est particulièrement utile pour comparer l'efficacité de différentes variantes du système, car il pénalise à la fois les systèmes gourmands en tokens et ceux qui échouent fréquemment.

La **robustesse** peut être quantifiée par le taux de pannes, c'est-à-dire le pourcentage d'exécutions qui se terminent par une erreur, ainsi que par le nombre moyen de tentatives nécessaires avant d'obtenir un résultat. Un système robuste devrait avoir un taux de pannes faible et rarement nécessiter plus d'une ou deux tentatives.

Pour chaque métrique, il est recommandé d'inclure un intervalle de confiance ou un écart-type lorsque vous effectuez des répétitions multiples. Cela permet de quantifier la variabilité des résultats et de déterminer si les différences observées entre variantes sont statistiquement significatives.

7 Procédure de suivi des coûts en tokens

Le suivi rigoureux des coûts en tokens nécessite une instrumentation appropriée de votre code dès le début du projet. Attendez-vous à ce que cette instrumentation soit complexe à ajouter rétroactivement, d'où l'importance de la mettre en place dès les premières itérations.

L'**instrumentation** consiste à implémenter un logger qui capture systématiquement les informations suivantes pour chaque appel à un modèle : l'horodatage précis de l'appel (`timestamp`), l'identifiant du scénario en cours d'exécution (`scenario_id`), un identifiant unique pour cet appel spécifique

(`call_id`), le nom et la version du modèle utilisé (`model`), l'endpoint ou le service appelé (`endpoint`), éventuellement un hash du texte du prompt pour faciliter l'analyse sans stocker le prompt complet (`prompt_text_hash`), le nombre de tokens dans le prompt (`prompt_tokens`), le nombre de tokens dans la complétion générée (`completion_tokens`), le total de tokens (`total_tokens`), le temps d'exécution de l'appel en millisecondes (`call_time_ms`), et le statut de la réponse indiquant succès ou type d'erreur (`response_status`).

Le **comptage des tokens** doit être précis et cohérent. Utilisez une bibliothèque comme `tiktoken` qui implémente la même tokenisation que les modèles OpenAI, ou une bibliothèque équivalente pour d'autres familles de modèles. Vous pouvez calculer les tokens localement avant l'appel pour estimer le coût, et après l'appel en utilisant les valeurs retournées par l'API pour obtenir le compte exact.

Le **fichier récapitulatif** `costs.csv` doit être exporté avec un en-tête explicite comprenant au minimum : `timestamp`, `scenario_id`, `call_id`, `model`, `endpoint`, `prompt_tokens`, `completion_tokens`, `total_tokens`, `latency_ms`, `status`, `notes`. Ce fichier servira de base pour toutes vos analyses ultérieures.

Dans un **notebook d'analyse**, vous calculerez les tokens totaux par scénario en agrégeant tous les appels associés, estimerez le coût en multipliant par les tarifs applicables (souvent différents pour les tokens de prompt et de complétion), et comparerez les variantes architecturales en générant des boxplots qui montrent la distribution des tokens consommés pour chaque approche.

La **visualisation** est essentielle pour communiquer vos résultats. Créez des histogrammes montrant la distribution des tokens par appel ou par scénario, des courbes cumulées illustrant comment les tokens s'accumulent au fil des appels, et des tableaux triés par `total_tokens` qui identifient les scénarios les plus coûteux et méritent une attention particulière pour l'optimisation.

8 Livrables et modalités de rendu

Le projet donne lieu à trois livrables principaux qui doivent être soumis avant la date limite indiquée.

Le premier livrable est un **rapport PDF de 12 pages** maximum qui documente l'ensemble de votre travail. Ce rapport doit présenter de manière structurée votre architecture système avec des diagrammes explicatifs, les motifs de conception choisis et leur justification détaillée, le protocole d'évaluation mis en place en précisant les métriques et la méthodologie, les résultats obtenus sous forme de tableaux et graphiques accompagnés d'une analyse approfondie, et une discussion critique sur les forces, faiblesses et perspectives d'amélioration de votre approche.

Le deuxième livrable est un **dépôt GitLab** contenant l'intégralité de votre projet. Ce dépôt doit être structuré de manière claire et inclure le code source avec un fichier README détaillé expliquant l'architecture, les prérequis d'installation et les commandes d'exécution, les fichiers de données et scénarios de test nécessaires à la reproduction de vos expériences, le fichier de suivi des coûts `costs.csv` ou `costs.json` généré automatiquement par votre système de logging, les scripts d'analyse (notebooks Jupyter ou scripts Python) permettant de reproduire vos résultats et visualisations, et tous les graphiques et tableaux présentés dans le rapport.

Le fichier de suivi des coûts peut inclure au minimum les champs suivants : `timestamp`, `scenario_id`, `call_id`, `model`, `endpoint`, `prompt_tokens`, `completion_tokens`, `total_tokens`, `latency_ms`, `status`, et `notes`. Ce fichier est essentiel pour l'analyse économique et permet de calculer les tokens totaux par scénario, d'estimer les coûts, et de comparer les variantes architecturales.

Le troisième livrable est une **vidéo de démonstration** de 5 à 10 minutes qui présente votre prototype en action et résume vos principaux résultats. Cette vidéo doit montrer des exemples d'utilisation concrets, illustrer le fonctionnement des différents motifs de conception, et mettre en valeur les aspects les plus importants de votre évaluation. La vidéo doit être déposée dans le dépôt GitLab, idéalement dans un dossier dédié nommé `demo/` ou `video/`. Vous pouvez l'héberger sur une plateforme comme YouTube ou Vimeo et inclure le lien dans le README, ou bien la déposer directement dans le dépôt si sa taille le permet.

L'accès au dépôt GitLab doit être partagé avec les enseignants avant la date limite. Assurez-vous que tous les membres du groupe ont contribué de manière visible dans l'historique Git, ce qui permettra d'évaluer la répartition du travail au sein de l'équipe.

9 Ressources suggérées

La documentation officielle de **LMStudio**, disponible sur leur site web, explique comment installer et configurer l'environnement pour exécuter des modèles localement. Elle contient également des guides pour optimiser les performances en fonction de votre matériel.

La documentation d'**Ollama** fournit des tutoriels pas à pas pour installer et utiliser différents modèles open-source. Son approche simplifiée en fait un excellent point de départ pour ceux qui découvrent l'exécution locale de modèles de langage.

L'**API d'inférence de Hugging Face** est documentée de manière exhaustive avec des exemples de code dans plusieurs langages. Consultez également le Hub de Hugging Face pour explorer les milliers de modèles disponibles et leurs caractéristiques.

Les bibliothèques de comptage de tokens comme `tiktoken` sont accompagnées d'exemples d'utilisation et de notebooks Jupyter qui illustrent comment calculer précisément les tokens pour différents modèles.

Les articles et notes de cours sur les motifs de conception agentiques, partagés pendant les séances de TD, constituent votre référence principale pour comprendre les fondements théoriques et les bonnes pratiques d'implémentation de ces motifs.

Enfin, les forums et communautés en ligne dédiés à l'IA et aux agents, comme les discussions sur GitHub, Stack Overflow, ou les serveurs Discord spécialisés, peuvent être des sources précieuses d'inspiration et d'aide pour résoudre des problèmes techniques spécifiques.

Bonne réalisation et bon apprentissage !