Université Paris Nanterre

ENS
ÉCOLE NORMALE SUPÉRIEURE

Mémoire de M1

Master MIAGE (apprentissage)

Les agents IA dans la réalisation de projets de développement

Entreprise d'accueil : École Normale Supérieure (ENS - PSL)
Mémoire réalisé du 2 septembre 2024 au 28 mai 2025

présenté et soutenu par

Kevin SOARES

le 10 juin 2025

Jury de la soutenance

M. François DELBOT,	Maître de conférences	Responsable du master / Tuteur
M. Stéphane POULAIN,	Directeur Financier	Maître d'apprentissage

Remerciements

Je tiens tout d'abord à exprimer ma sincère gratitude à **M. François Delbot**, Maître de conférences et responsable du Master, pour son accompagnement tout au long de mon parcours. Déjà tuteur enseignant lors de mon stage de fin d'année précédente, il m'a de nouveau soutenu cette année, en m'aidant à définir une problématique pertinente pour ce mémoire. Je le remercie chaleureusement pour son engagement constant et la qualité de son encadrement.

Mes remerciements vont également à **M. Karim El-Mahou**, qui a été mon tuteur lors mon arrivée à l'ENS. Son soutien technique comme humain a grandement facilité mon intégration et orienté mes premiers travaux.

Je souhaite enfin remercier **M. Stéphane Poulain**, directeur du département, pour l'accueil chaleureux qu'il m'a réservé et pour les conditions de travail qu'il a mises en place. Sa disponibilité et ses conseils avisés ont contribué à faire de cette expérience un moment riche tant sur le plan professionnel que personnel.

Résumés (1 page)

Résumé

Les progrès récents des Large Language Models (LLMs) et des architectures multi-agents soulèvent une question audacieuse : peut-on remplacer une équipe de développement logicielle par un collectif d'agents IA ? Afin d'éclairer ce débat, ce mémoire dresse d'abord l'état de l'art des LLMs appliqués au génie logiciel, puis cartographie les fonctions clés, de l'analyse d'exigences à la mise en production, qu'une équipe doit assumer. L'examen de trois prototypes représentatifs (*CodePori*, *Agent-Driven Automatic Software Improvement* et un pipeline Auto-DevOps multi-agents) montre que la génération, les tests et le débogage peuvent déjà être automatisés avec un taux de précision compris entre 80 et 90%. Cependant, les risques de biais, d'hallucinations et de fuites de données obligent à maintenir des garde-fous et une supervision humaine, notamment pour l'architecture et la gouvernance. Une matrice SWOT met en évidence des forces (exécution 24 / 7, coûts réduits), mais aussi des menaces (responsabilité légale, attaques backdoor). Trois scénarios d'évolution sont proposés : *développement assisté* à court terme, *équipes hybrides* à moyen terme, puis *autonomie quasi complète* sous audits périodiques. La conclusion affirme que le remplacement total est techniquement concevable, mais conditionné à l'alignement, à la sécurité et à un cadre réglementaire robuste. Ce travail offre ainsi un outil d'aide à la décision pour chercheurs, industriels et législateurs confrontés à l'émergence des équipes 100% IA.

Abstract

Recent advances in *Large Language Models* (LLMs) and multi-agent systems raise a bold question : can a software development team be entirely replaced by AI agents ? To address this, the thesis first surveys state-of-the-art LLM techniques for software engineering and maps essential roles, from requirements analysis to production deployment, that any team must cover. Three representative prototypes (*CodePori*, *Agent-Driven Automatic Software Improvement*, and an Auto-DevOps multi-agent pipeline) reveal that code generation, testing and debugging can already be automated with 80 to 90% accuracy. Yet bias, hallucinations and data-leak risks still demand human oversight for architecture and governance. A SWOT matrix highlights strengths (24 / 7 execution, lower recurring costs) and threats (legal accountability, backdoor attacks). Building on this analysis, three evolution scenarios are outlined : *AI-assisted pair programming* in the short term, *lean hybrid teams* mid-term, and near-full autonomy under periodic audits in the long run. The thesis concludes that full replacement is technically plausible but contingent upon reliable alignment, strong security guarantees and a clear regulatory framework. Overall, the work provides a decision-making tool for researchers, practitioners and policymakers facing the rise of "all-AI" development teams.

Table des matières

Chapitre 1

Introduction

L'essor des modèles de langage de grande taille (**Large Language Models** ou LLMs) a bouleversé la façon dont nous interagissons avec le texte : un simple prompt suffit désormais à générer une page de prose, un résumé d'article ou, plus surprenant encore, un extrait de code compilable. Dans la foulée, une nouvelle génération d'agents IA, capables de raisonner sur plusieurs tours, d'appeler des outils externes et de **collaborer entre eux**, prétend couvrir l'ensemble du cycle logiciel : rédaction des spécifications, écriture des tests, production du code, déploiement continu. Face à ces annonces, une question surgit : **Une équipe de développement peut-elle, à terme, être remplacée entièrement par des agents IA ?**

Ce mémoire propose de répondre à cette interrogation en adoptant une démarche bibliographique structurée. Nous passons d'abord en revue les principes des LLMs et les architectures multi-agents, puis nous confrontons leurs performances à des benchmarks de génération de code. Nous analysons ensuite trois prototypes représentatifs : *CodePori*, *Agent-Driven Automatic Software Improvement* et un pipeline Auto-DevOps multi-agents, qui illustrent le potentiel, mais aussi les limites, d'équipes 100 % IA (ou presque). Enfin, nous synthétisons ces observations via une matrice SWOT et nous esquissons trois scénarios d'évolution, de la programmation assistée d'aujourd'hui à l'autonomie (quasi) complète de demain.

La contribution essentielle de ce travail est double. D'une part, il offre une **cartographie des tâches** qu'une équipe de développement doit couvrir et un état de la couverture actuelle par les agents IA. D'autre part, il identifie les **conditions critiques** (alignement, sécurité, gouvernance) sans lesquelles le remplacement total demeurerait théorique. L'objectif n'est pas de trancher définitivement, mais de fournir au lecteur une grille d'analyse argumentée pour évaluer la faisabilité, les risques et les opportunités d'un tel bouleversement.

Le plan suit une logique progressive : après l'état de l'art 3 et l'étude de la dynamique équipe humaine/agents 4, nous passons aux cas d'usage concrets 5, puis à une discussion de faisabilité et de perspectives 6. La conclusion 7 revient sur les points clés et ouvre sur les recherches futures, qu'elles soient techniques, organisationnelles ou éthiques.

En somme, ce mémoire vise à montrer pourquoi le sujet est crucial, comment nous l'avons abordé, et en quoi nos résultats peuvent éclairer décideurs, chercheurs et praticiens. Le lecteur n'a pas besoin d'être expert en IA : il trouvera ici les repères nécessaires pour comprendre les promesses, les limites et les implications d'une **éventuelle équipe de développement sans développeurs**.

Chapitre 2

Méthodologie

2.1 Systematic Mapping Study (SMS)

La **Systematic Mapping Study** (SMS) est une méthode de revue de littérature formelle visant à cartographier et classer l'état de l'art d'un domaine de recherche de façon exhaustive et reproductible. Elle se déroule en quatre grandes phases :

1. **Définition des questions de recherche** (QR) et des objectifs de la carte ;
2. **Recherche systématique** des publications pertinentes dans les bases de données (ex. IEEE Xplore, Scholar, ACM) selon des chaînes de requêtes préétablies ;
3. **Sélection et filtrage** des articles via des critères d'inclusion/exclusion, combinant filtrage automatique (titres, abstracts, scoring) puis examen manuel (abstract, introduction, table des matières, conclusion) ;
4. **Classification et synthèse** des résultats : attribution de catégories thématiques, visualisation sous forme de matrice ou de graphique, identification des lacunes et des tendances.

Dans ce mémoire, nous appliquons la démarche SMS pour structurer notre recherche bibliographique sur la génération automatique de code par agents IA. Chaque étape (définition des requêtes, filtrage, scoring, classification) suit le protocole SMS afin d'assurer fiabilité, transparence et reproductibilité des travaux existants.

2.2 Méthode PICO

Pour structurer la question de recherche, nous utilisons la méthode **PICO**. Cette approche, acronyme de *Population*, *Intervention*, *Comparison* et *Outcomes*, exige de définir explicitement ces quatre composantes. Elle aide ainsi à formuler une question précise et à guider la recherche .

Notre question de recherche est formulée de la manière suivante : "Est-il possible de remplacer entièrement une équipe de développement par des agents IA ?"

Le tableau 2.1 présente la décomposition de notre problématique selon la méthode PICO.

Élément	Application au mémoire
Population	Équipe de développement
Intervention	Remplacement par des agents IA
Comparison	Équipe humaine traditionnelle
Outcomes	Faisabilité technique d'un remplacement total

TABLE 2.1 – Formulation PICO de la question de recherche

2.3 Formulation des requêtes

La méthode PICO décrite ci-dessus nous permet ainsi de formuler les requêtes nous permettant d'interroger les bases de données scientifiques et de pouvoir en extraire des papiers utiles à notre recherche. Pour répondre à celle-ci, nous divisons notre recherche en quatre requêtes :

— Définition de LLM
— Définition d'agent IA
— Taille et composition idéales d'une équipe de développement
— Remplacement d'une équipe de développement par des agents IA

Ces quatre requêtes vont nous permettre de définir ce que sont des LLM ainsi que des agents IA, pour ensuite examiner les papiers consacrés à la taille et composition idéales d'une équipe. Enfin, nous explorerons les travaux sur la génération automatique de code par les agents IA ce qui nous fournira les éléments nécessaires pour croiser ces informations et déterminer dans quelle mesure une équipe humaine peut ou non être remplacée par des agents IA.

Pour la recherche, nous utilisons Google Scholar qui permet de chercher des papiers dans différentes bases (e.g. IEEE ou ArXiv) ce qui en fait l'outil le plus polyvalent pour nos besoins.

2.3.1 Définition de LLM

Cette requête (Listing 2.1) nous met à disposition les papiers permettant de définir le terme "LLM".

```
1
2   allintitle:"large␣language␣model" ("systematic␣literature␣review" OR survey OR taxonomy)
```

Listing 2.1 – Requête - LLM

2.3.2 Définition d'Agents IA

Cette requête (Listing 2.2) recense les papiers proposant des définitions d'agents IA (ou " LLM agents "), afin de cerner précisément ce concept.

```
1   allintitle:(agent OR "LLM␣agent") (taxonomy OR definition)
```

Listing 2.2 – Requête - Agents IA

2.3.3 Taille et composition idéales d'une équipe de développement

Cette requête (Listing 2.3) vise la littérature traitant de la taille idéale et de la composition optimale des équipes de développement logiciel.

```
intitle:"team size" AND ideal AND (software development OR agile OR scrum)
```

Listing 2.3 – Requête - Equipe de développement

2.3.4 Remplacement d'une équipe de développement par des Agents IA

Cette requête (Listing 2.4) rassemble les travaux étudiant dans quelle mesure des agents IA sont capables de générer automatiquement du code et peuvent remplacer une équipe de développement humaine.

```
("automatic code generation" OR "code synthesis" OR "AI-generated code") AND (intitle:agent OR intitle:"
    software agent" OR intitle:"AI agent")
```

Listing 2.4 – Requête - Remplacement équipe de développement

2.4 Filtrage

Avant de lire entièrement chaque papier, nous appliquons plusieurs étapes de filtrage afin de conserver uniquement les papiers essentiels et qui auront une réelle utilité au développement de nos propos.

2.4.1 Filtrage sur l'année

Notre premier critère de filtrage, appliqué directement lors de la recherche depuis Google Scholar, est l'année.

Pour chaque recherche, un filtre sur l'année a été appliqué : Pour la définition de LLM (Listing 2.1) et l'automatisation du développement avec agents IA (Listing 2.4) nous recherchons uniquement les papiers publiés à partir de 2024. Ce choix est dû au grand nombre de papiers traitant des LLM et à l'essor récent de ceux-ci ces dernières années, cela nous permet donc premièrement de retenir moins de papiers et deuxièmement d'avoir des études plus récentes et actuelles de ceux-ci. On passe donc de 121 à 101 papiers pour la définition des LLM et de 325 à 63 papiers pour l'automatisation du développement avec agents IA.

Concernant la définition des agents IA (Listing 2.2) et la composition et taille idéales d'une équipe de développement (Listing 2.3) nous recherchons uniquement les papiers publiés à partir de 2024. On passe donc de 174 à 22 papiers pour la définition des agents IA et de 127 à 30 papiers pour l'équipe de développement.

2.4.2 Filtrage sur le type de document

Après extraction des différents papiers à l'aide de Zotero, nous commençons les étapes de filtrage plus précises passant par différents scripts python. La première d'entre elles est un filtrage sur le type de document. Nous pouvons voir dans nos papiers différents types de documents, en voici la liste exhaustive ainsi que leur nombre :

- journalArticle : 108
- preprint : 66
- conferencePaper : 28
- bookSection : 7
- thesis : 3

— document : 2
— report : 1
— book : 1

Nous décidons donc d'exclure les papiers de type *report* et *document* étant des types de documents sans description ne nous permettant pas de savoir au premier abord de quel papier il s'agit. Le tableau 2.2 montre le nombre de papiers pour chaque requête avant et après le filtrage sur le type de document.

Requête	**Papiers avant filtrage**	**Papiers après filtrage**
Définition LLMs	101	101
Définition agents IA	22	21
Taille et composition d'équipe	30	29
Automatisation par Agents IA	63	62
Total	216	213

TABLE 2.2 – Filtrage sur le type de document

2.4.3 Filtrage sur le titre

Nous continuons le filtrage en procédant à une sélection sur le titre avec une liste de mots éliminatoires tels que 'political', 'philosophy' ou encore 'religion' (une liste exhaustive est disponible en annexe .1).

Le tableau 2.3 montre le nombre de papiers pour chaque requête avant et après filtrage sur le titre.

Requête	**Papiers avant filtrage**	**Papiers après filtrage**
Définition LLMs	101	98
Définition agents IA	21	21
Taille et composition d'équipe	29	26
Automatisation par Agents IA	62	62
Total	213	207

TABLE 2.3 – Filtrage sur le titre : mots éliminatoires

2.4.4 Filtrage sur l'abstract

Notre dernière étape de filtrage *"automatique"*, avant le filtrage manuel, sera un système de points sur l'abstract. Pour cela, on établit un système à points qui sont attribués lorsque les mots d'un groupe de mots prédéfini sont présents dans l'abstract. Le maximum étant de 3 points si tous les termes sont compris dans l'abstract et le minimum de 0, si aucun terme n'est présent. Le tableau 2.4 présente pour chaque requête la répartition des papiers en fonction de leur score d'abstract.

Score / Requête	3pts	2pts	1pts	0pts
Définition LLMs	5	54	29	10
Définition agents IA	5	3	6	7
Taille et composition d'équipe	2	13	10	1
Automatisation par Agents IA	1	5	26	30
Total	**14**	**76**	**71**	**48**

TABLE 2.4 – Filtrage sur l'abstract : score

Suite aux résultats de ce filtrage, nous garderons les papiers ayant obtenu un score de 3 points. Cependant, nous prenons la décision de garder également ceux ayant obtenu 2 points pour les requêtes *Taille et composition d'équipe* et *Automatisation par Agents IA*.

Il nous reste ainsi un total de 31 papiers à trier manuellement.

2.4.5 Filtrage manuel

Les étapes de filtrage *"automatiques"* sont utiles grâce à leur simplicité et rapidité d'exécution, surtout sur de gros volumes de papiers, mais restent cependant assez limitées au niveau de la compréhension du texte. Nous devons donc procéder à un filtrage manuel en lisant d'abord l'abstract (première étape), puis en passant en revue l'introduction, les titres de sections et la conclusion (deuxième étape) des articles sélectionnés à l'étape précédente.

Le tableau 2.5 présente le nombre de papiers obtenus après chaque étape de filtrage manuel.

Requêtes	Nombre initial de papiers	1ère étape	2e étape
Définition LLMs	5	2	**2**
Définition agents IA	5	2	**2**
Taille et composition d'équipe	15	2	**2**
Automatisation par Agents IA	6	6	**6**
Total	**31**	**12**	**12**

TABLE 2.5 – Filtrage manuel

Nous conservons donc pour notre recherche un total de **12** papiers dont :
- **2** pour la définition des LLM ;
- **2** pour la définition des agents IA ;
- **2** pour la taille et la composition d'équipe ;
- **6** pour l'automatisation par Agents IA.

Évidemment, bien que chaque article soit rattaché à une catégorie ou à une question de recherche précise, les informations qu'il contient peuvent être mobilisées dans différentes parties de ce mémoire.

Chapitre 3

État de l'art des LLMs appliqués au développement informatique

3.1 Définitions fondamentales : IA, LLM et Agent IA

3.1.1 Intelligence artificielle (IA)

L'**intelligence artificielle** (**IA**) est un domaine de l'informatique visant à concevoir des programmes capables de comprendre leur environnement, raisonner et agir de façon à maximiser la réussite d'objectifs (tels que la reconnaissance d'images, la planification de trajectoires ou, comme dans ce mémoire : **la génération de code**).

3.1.2 Large Language Model (LLM)

Un **Large Language Model** (**LLM**) est un réseau de neurones de type transformeur entraîné sur une large gamme de données. (Cui et al., 2024). Son objectif est d'estimer la probabilité du prochain mot afin de générer du texte cohérent. Inclure du code source dans le corpus permet au modèle de créer, compléter ou modifier des programmes informatiques. Le LLM est donc essentiellement une "brique" de génération et de raisonnement probabiliste : il produit du **texte** mais n'interagit pas par lui-même avec un environnement logiciel.

3.1.3 Agent IA

Nous appelons **Agent IA** une entité logicielle semi-autonome voire autonome qui observe un état (par exemple un dépôt git / un projet ou un rapport de tests), planifie une séquence d'actions et exécute ces actions pour atteindre un objectif. Un agent IA s'appuie sur un ou plusieurs LLMs pour la génération ou la prise de décision et a donc un certain set de compétences, un rôle défini et une mémoire qui lui est propre (Händler, 2023a), mais entoure ceci d'une logique d'orchestration : mémoire de travail, appels d'API, exécution de tests, boucles de rétroaction.

3.1.4 Différences essentielles entre IA, LLM et Agent IA

Maintenant que nous avons défini ces 3 termes, nous pouvons aborder les différences essentielles entre ceux-ci :

— **Périmètre :** l'IA est l'ensemble. Le LLM est un sous-ensemble spécialisé dans la modélisation du langage naturel entraîné sur des ensembles de données. L'agent

IA est un système logiciel qui intègre potentiellement un ou plusieurs LLMs tout en y ajoutant une couche d'autonomie (planification, action, raisonnement) (Cui et al., 2024).

— **Architecture :** le LLM est un modèle de type transformeur ne conservant pas d'état externe entre les appels, tandis que l'agent IA maintient un état persistant et une mémoire de long terme pour effectuer plusieurs tours d'itération (Händler, 2023a).

— **Entrées/Sorties :** le LLM prend un prompt textuel et retourne du texte. L'agent IA accepte des objectifs de haut niveau, invoque des outils (exécution de tests, CI/CD, IDE) et valide les résultats avant de produire un livrable logiciel (Rasheed et al., 2024).

— **Responsabilité :** un LLM ne porte pas d'intention propre ; toute responsabilité incombe à l'utilisateur. L'agent IA en revanche inclut une couche de prise de décision et d'alignement dont la sûreté doit être évaluée (Cui et al., 2024).

En résumé, le LLM est le moteur de génération et de raisonnement, tandis que l'agent IA en est le conducteur capable d'exploiter ce moteur pour naviguer, de manière semi-autonome / autonome, dans le cycle de vie de l'atteinte de l'objectif défini (dans notre cas : pour le développement logiciel).

3.2 Capacités actuelles des LLMs pour la génération de code

Les benchmarks de complétion et de correction sur code ont rapidement servi d'indicateurs clés pour évaluer les LLMs. Par exemple, **HumanEval** (par OpenAI) : Rasheed et al. (2024) rapportent que *CodePori*, un système d'agents LLM dont nous ferons l'étude de cas au Chapitre 5.1, atteint jusqu'à 89% d'amélioration de précision du code comparé à une dizaine de modèles de génération de code sur ce benchmark et 85% lors d'une évaluation manuelle de projets complexes.

Au-delà des scores, la qualité du code généré (lisibilité, absence de bugs) s'améliore lorsqu'on applique des cycles itératifs d'exécution de tests et de correction automatique :

— Dans Vallecillos Ruiz (2024), des agents LLM orchestrés en pipeline [génération → tests → amélioration] démontrent une réduction significative des erreurs "last-mile" (problème présents lors du fin de processus de distribution) grâce à l'apprentissage par rétroaction.

— Zahid et Hussain (2024) montrent enfin que la collaboration de plusieurs LLMs spécialisés ("boilerplate writer", "doc generator") réduit le temps de développement et augmente la couverture de tests dans des scénarios de projet réels.

3.3 Architectures multi-agents basées LLM

Les architectures multi-agents tirent parti de plusieurs LLMs collaborant selon des rôles et des topologies variées pour accomplir des tâches complexes de génération de code.

— **Dimension composition d'agents** :
Chaque agent se spécialise dans un sous-ensemble de fonctions (par exemple, analyse des spécifications, génération de code, tests, relecture) et possède sa propre mémoire et accès à des outils externes (Händler, 2023a).

— **Modes de collaboration** :

  - Pipeline itératif : les agents s'enchaînent selon un flux génération → tests → amélioration, exploitant les boucles de rétroaction pour corriger par exemple les "last-mile" problems (Vallecillos Ruiz, 2024).
  - Collaboration distribuée : plusieurs agents LLM, spécialisés (boilerplate writer, doc generator), travaillent en parallèle et consolident leurs résultats pour produire un livrable cohérent (Zahid & Hussain, 2024).
- **Équilibrage de charge et orchestration** :
  Pour optimiser la performance et la cohérence, des stratégies dynamiques redistribuent les tâches entre agents selon la charge actuelle et leur spécialisation tout en veillant à la convergence des résultats (Händler, 2023b).
- **Niveaux d'autonomie** :
  Chaque agent peut opérer de manière plus ou moins autonome selon le degré de supervision humaine et les guardrails définis de l'exécution strictement guidée à l'initiative complète sur des sous-tâches (Händler, 2023a).

3.4 Limites techniques et risques inhérents

L'utilisation de LLMs et d'agents IA soulève plusieurs enjeux critiques pouvant compromettre la fiabilité, la sécurité et la maintenabilité des solutions logicielles.

3.4.1 Biais et hallucinations

Les LLMs, entraînés sur des jeux de données massifs, intègrent souvent des biais socio-culturels ou sectoriels. Ces biais peuvent conduire à des généralisations inappropriées ou à des décisions discriminatoires dans le code produit. De plus, les modèles peuvent **halluciner** : générer du code incorrect ou non compilable, voire dangereux (plus généralement, Wikipedia, 2025a : générer des réponses fausses à partir de modèles ou données inexistantes) (Cui et al., 2024).

3.4.2 Sécurité et confidentialité

Les processus de formation et d'utilisation des LLMs exposent à des menaces spécifiques tant au niveau de la sécurité que de la confidentialité :

- **Vulnérabilités induites** : du code généré pourrait comporter des backdoors ou exploiter des librairies vulnérables, ouvrant la porte à des attaques ciblées, comme indiqué dans la taxonomy des menaces de (Wang et al., 2024).
- **Fuite de données sensibles** : les LLMs peuvent mémoriser et régurgiter des extraits de leur corpus d'entraînement, y compris des secrets de configuration ou des clés d'API accidentellement indexées.

3.5 Conclusion

Nous avons défini les briques fondamentales — IA, LLM et Agent IA — et exploré les capacités actuelles des LLMs pour la génération de code, ainsi que les architectures multi-agents et leurs limites techniques. Avant de passer à l'analyse comparative IA vs humain, il est utile de préciser deux critères clés d'évaluation :

— **Degré d'autonomie** : capacité d'un agent à planifier et exécuter des tâches sans intervention humaine, mesurée par la fréquence et l'importance des points de contrôle nécessaires pour éviter les erreurs critiques (tests, ...).
— **Niveau d'alignement** : adéquation des décisions de l'agent avec les objectifs de haut niveau et les contraintes du projet (sécurité, style, normes), évaluée à l'aide de métriques qualitatives (revues humaines) et quantitatives (taux de conformité aux tests).

Ces deux critères permettront de juger si et comment une équipe d'agents IA peut réellement se substituer à une équipe humaine.

Chapitre 4

Effort humain, dynamique d'équipe et substitution potentielle

4.1 Taille optimale et productivité des équipes humaines

À présent, nous cherchons à quantifier comment des agents IA peuvent remplacer ou épauler une équipe humaine en simulant des workflows réels de développement.

Dans un projet de développement, la constitution de l'équipe est un facteur déterminant de réussite. Il faut qu'elle soit : ni trop petite (risque de surcharge -> retards sur les livraisons, burn-outs, ... et de dépendance à quelques individus), ni trop grande (coûts de coordination élevés, manque de communication et complexité organisationnelle). Pour mesurer et comparer la productivité des équipes, on utilise notamment :

— **SLOC** (Source Line Of Code) et **defect density** pour évaluer le rendement et la qualité / présence de défauts (Wikipedia, 2025b), (Wikipedia, 2025c) ;

— **Bus factor** pour mesurer la vulnérabilité organisationnelle (Wikipedia, 2025d).

Des études récentes confirment et affinent ces constats :

— Olivares et al. (2024) appliquent des algorithmes bio-inspirés et montrent que la taille optimale d'une **équipe Agile** se situe généralement **entre 5 et 9 membres**. Au-delà, les retours sont décroissants, surtout en l'absence de structures de communication formalisées.

— Bodaragama et al. (2023) démontrent que malgré qu'une équipe plus petite produit du code avec moins de défauts et qu'une plus grande équipe produirait un code plus complexe qui pourrait donc conduire à des problèmes de maintenance, la relation taille d'équipe–qualité dépend de facteurs modérateurs (communication, expérience, complexité), sans seuil universel, et soulignent l'importance d'utiliser plusieurs métriques de qualité pour en rendre compte et donc choisir d'une taille d'équipe optimale.

Ces résultats soulignent donc :

— l'importance cruciale des compétences transverses (revue de code, tests, DevOps) pour maintenir la qualité ;

— la nécessité d'adapter la taille des équipes aux besoins du projet et de mettre en place des canaux de communication formalisés, afin de prévenir la surcharge organisationnelle et de garantir une qualité de code élevée.

En conséquence, pour des projets exigeant une maintenabilité constante et soumis à des évolutions fréquentes, il est préférable de privilégier une équipe restreinte, capable de conserver une vision partagée, de fluidifier la communication et de limiter la complexité organisationnelle.

4.2 Simulations d'équipes multi-agents

Pour évaluer la capacité des agents IA à prendre en charge des workflows de développement logiciel et mesurer leur substitution potentielle à une équipe humaine, plusieurs études récentes ont mis en place des cadres expérimentaux qui reproduisent des interactions de projet réelles.

— Ashraf et Talavera (2025) présentent un cadre expérimental où des agents LLM spécialisés (analyse des exigences, génération de code, débogage, exécution des tests, documentation) collaborent sur un même projet, et mesurent l'accélération des livraisons et la réduction des erreurs humaines.

— Zahid et Hussain (2024) explorent des architectures multi-agents où différents LLMs collaborent sur des phases de génération, test et correction de code, et rapportent une amélioration du débit global de production ainsi qu'une meilleure détection de certaines classes d'erreurs.

Les indicateurs clés retenus sont :

— l'accélération des livraisons et la réduction des erreurs humaines (Ashraf & Talavera, 2025) ;

— l'amélioration du débit global de production de code et de la détection d'erreurs (Zahid & Hussain, 2024).

Les conclusions principales sont les suivantes :

— Ashraf et Talavera (2025) démontrent que la coopération de plusieurs agents LLM accélère significativement les livraisons et réduit les erreurs humaines.

— Zahid et Hussain (2024) montrent que ces agents, en automatisant les tâches répétitives (code boilerplate, documentation), allègent la charge de travail et améliorent la productivité, tout en contribuant à l'identification des inefficacités et vulnérabilités du code.

Ces simulations démontrent que, bien que la parallélisation des rôles augmente sensiblement la productivité, la qualité finale reste dépendante d'une supervision formelle (revue et tests approfondis), soulignant l'importance d'un équilibre entre autonomie des agents et contrôles de fiabilité.

Chapitre 5

Études de cas et prototypes d'équipes IA autonomes

5.1 Cas d'étude : *CodePori*

CodePori, proposé par Rasheed et al. (2024), vise à automatiser la génération de code pour des projets complexes à partir d'une description fonctionnelle en langage naturel. Le système orchestre **six Agents IA** possédant chacun un rôle attitré bien défini, visible dans le tableau 5.1 :

No	Agent	Rôle principal	Contenu du Prompt
1	Manager	Segmentation des tâches	Prend la description du projet et segmente les tâches en modules plus petits et structurés
2	Dev_01	Génération du code initial	Génère le code initial en se basant sur la description donnée
3	Dev_02	Optimise le code généré	Affine et optimise le code généré par Dev_01
4	Finalized_01	Assurance qualité du code	Joue un rôle d'assurance qualité et affine le code pour l'utilisation finale
5	Finalized_02	Revue de qualité du code	Suggère un processus de révision itératif pour améliorer la qualité du code
6	Verification	Vérification Finale du Code	Se concentre sur l'identification et la rectification des défauts de code que l'agent ci-dessus aurait pu manquer

Table 5.1 – Tâche principales du Multi-agent, (Rasheed et al., 2024)

Flux de travail. La figure 5.1 du papier illustre le pipeline : après avoir reçu la description du projet, le Manager décompose le projet, les Dev agents itèrent pour la génération du code, les Finalized font la revue croisée, puis le Verification_Agent valide l'exécutable en cherchant les défauts que les Finalized auraient pu manquer. Toutes les requêtes sont acheminées par HTTPS vers l'API OpenAI, garantissant la traçabilité des échanges.

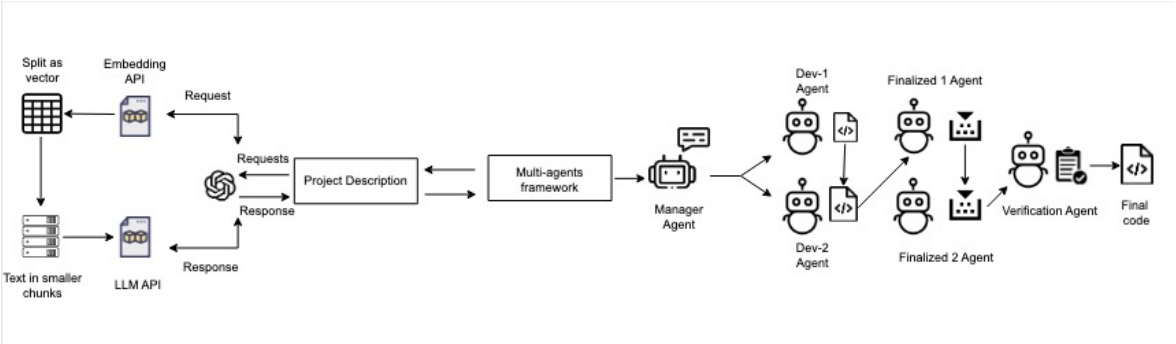

FIGURE 5.1 – Diagramme du workflow d'un système multi agent dirigé par IA pour la génération automatique de code (RASHEED et al., 2024)

Évaluation.

— **Benchmark HumanEval** : 89% de précision au test **Pass@1** (évaluant le taux de réussite au premier coup), surpassant plusieurs modèles de référence comme MetaGPT ou encore ChatDev.
— **Test manuel** sur 20 descriptions de projets : 85% de code exécutable sans modifications (17 résultats concluants).
— **Coût et latence** : génération d'un projet complet "en quelques minutes" pour "quelques dollars" [1]. RASHEED et al., 2024.

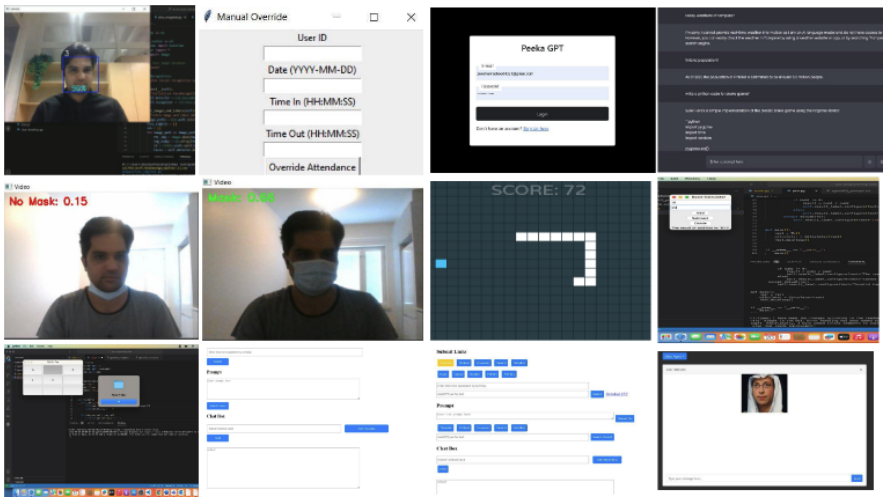

FIGURE 5.2 – Logiciels développés par CodePori (RASHEED et al., 2024)

Forces et limites. CodePori démontre la faisabilité d'une équipe IA couvrant tout le cycle de développement (analyse → codage → tests → assurance qualité), mais l'étude souligne la dépendance à l'orchestration centrale et la nécessité d'itérations multiples pour obtenir un code pleinement fonctionnel.

1. Par exemple, pour un projet de détection de masque (visible sur la figure 5.2) sur le visage, on a : 18 minutes pour 399 lignes de code et un coût de 1.02$

5.2 Cas d'étude : Agent-Driven Automatic Software Improvement

Vallecillos Ruiz (2024) présente un projet doctoral qui vise à améliorer la maintenance logicielle en combinant **LLMs** et **agents IA** dans un cadre itératif de génération–révision.

Concept et architecture. La figure 5.3 introduit un framework conceptuel composé d'un Agent LLM (Agent IA) et d'un critique (humain, outil ou autre LLM) échangeant sur des boucles continues pour affiner le code généré en donnant un retour entre chaque sortie (Output).

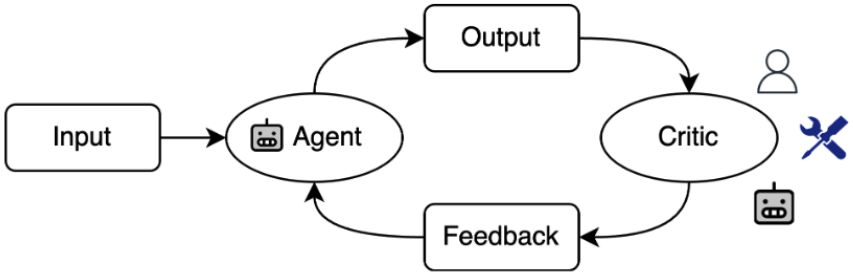

Figure 5.3 – Framework conceptuel pour un Agent IA, (Vallecillos Ruiz, 2024)

Trois scénarios d'interaction sont distingués (figure 5.4) :

1. **Agent unique** agissant en autocontrôle ;
2. **Agents multiples** coopérant ou se spécialisant ;
3. **Human-in-the-loop** combinant expertise humaine et puissance de l'agent.

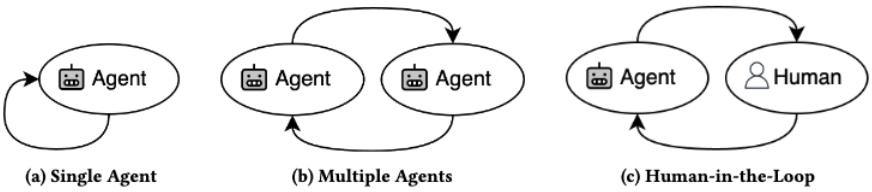

Figure 5.4 – Scenarios d'interactions entre Agents IA, (Vallecillos Ruiz, 2024)

Objectifs et contributions attendues. Le projet se déploie en trois phases correspondant aux questions de recherche :

- **RQ1** : comparer agent unique vs. usage one-shot d'un LLM sur des tâches d'amélioration de code ;
- **RQ2** : étudier l'effet synergique d'équipes multi-agents spécialisées pour des scénarios complexes ;
- **RQ3** : proposer de nouvelles méthodes de *fine-tuning* tirant parti du processus itératif.

Boucles itératives et *last-mile* problems. Les agents itératifs sont conçus pour résoudre les *last-mile errors*, c'est-à-dire les bogues subtils apparaissant en toute fin de génération fonctionnelle. Chaque cycle d'itération permet :

1. de recevoir un retour du critic ;
2. de régénérer ou patcher la portion fautive ;
3. d'améliorer progressivement sécurité, efficacité et style.

Positionnement par rapport à l'état de l'art. L'auteur souligne que les approches mono-shot échouent souvent sur des codes complexes ou vulnérables, alors que des agents multi-itératifs sont plus compétents et plus aptes, par exemple, à régler un "last-mile problem".

Forces et limites.

— **Forces :** cadre générique, possibilité d'exploiter des experts spécialisés, amélioration continue via feedback loops.
— **Limites :** orchestration complexe, besoin d'une comparaison systématique des approches multi-agent, dépendance aux coûts d'appels API et à la qualité des critiques.

5.3 Autres frameworks récents

5.3.1 Pipelines Auto-DevOps multi-agents

Khan et Daviglus (2025) décrivent un cadre multi-agent intégré aux pratiques DevOps : des agents LLM distincts s'occupent de la planification de sprint, de la génération de code, de l'exécution automatique de tests et du déclenchement de pipelines CI/CD. Les auteurs rapportent une réduction du temps de développement et une meilleure cohérence entre le backlog et le code livré grâce à la distribution dynamique des tâches et au feedback continu entre agents.

5.3.2 Retours d'expérience industriels ou open-source

Dans un contexte industriel, Abbas et Wahab (2024) montrent qu'une adoption progressive d'agents LLM pour la revue de code, la priorisation du backlog et l'automatisation des tests permet d'alléger la charge des développeurs et d'augmenter la productivité globale des équipes. Les agents interviennent notamment pour :

— générer la documentation et détecter les risques en temps réel ;
— automatiser la revue de code et proposer des correctifs.

5.4 Évaluation empirique

La comparaison des prototypes et retours d'expérience fait ressortir trois indicateurs récurrents :

— **Réduction du temps de développement** : les pipelines multi-agents permettent d'accélérer la planification et l'intégration continue (Khan & Daviglus, 2025).
— **Amélioration de la productivité et de la détection d'erreurs** : l'automatisation des tâches répétitives (documentation, tests, revue) libère du temps pour les activités à forte valeur ajoutée (Abbas & Wahab, 2024).

— **Cohérence backlog ↔ code livré** : la coordination d'agents spécialisés assure une meilleure traçabilité des exigences fonctionnelles jusqu'au code (Khan & Daviglus, 2025).

Chapitre 6

Faisabilité et perspectives du remplacement total

6.1 Cartographie des tâches d'une équipe de développement

Les articles de notre bibliographie font apparaître un ensemble récurrent de **huit fonctions** que toute équipe de développement — humaine ou multi-agents — doit couvrir :

- **Analyse des exigences & priorisation du backlog** (Ashraf & Talavera, 2025 ; Abbas & Wahab, 2024).
- **Planification de sprint & management de projet** (Khan & Daviglus, 2025).
- **Génération et refactorisation du code** (Rasheed et al., 2024 ; Zahid & Hussain, 2024).
- **Tests automatisés et validation fonctionnelle** (Rasheed et al., 2024 ; Khan & Daviglus, 2025).
- **Débogage et correction de bugs** (Vallecillos Ruiz, 2024 ; Ashraf & Talavera, 2025).
- **Documentation et génération de rapports** (Zahid & Hussain, 2024 ; Ashraf & Talavera, 2025).
- **Revue de code / Assurance qualité** (Rasheed et al., 2024 ; Abbas & Wahab, 2024).
- **Intégration & déploiement continus (CI/CD)** (Khan & Daviglus, 2025).

6.2 Couverture actuelle par les agents IA

Les études analysées montrent un niveau d'automatisation contrasté :

- **Développement :** fort (85 à 89% de précision HumanEval avec CodePori) (Rasheed et al., 2024).
- **Refactorisation / bug fixing :** amélioration notable via pipelines itératifs (Vallecillos Ruiz, 2024).
- **Tests générés automatiquement :** partiel, dépend de la couverture de benchmarks (Abbas & Wahab, 2024).
- **Architecture et priorisation produit :** faible ; décisions stratégiques requièrent toujours une validation humaine (Händler, 2023a).

— **DevOps :** automatisation des pipelines CI/CD prometteuse mais encore supervisée (Khan & Daviglus, 2025).

6.3 Analyse SWOT d'une équipe 100 % IA

Avant de statuer sur la faisabilité d'un remplacement total, il est utile de synthétiser, sous forme SWOT, les **forces** (Strength), **faiblesses** (Weaknesses), **opportunités** (Opportunities) et **menaces** (Threats) dégagées par les études de cas et simulations précédentes. Le tableau 6.1 résume ces facteurs :

Strength	Vitesse d'exécution quasi 24/7 (Ashraf & Talavera, 2025)	Réduction des coûts unitaires (Rasheed et al., 2024)
Weaknesses	Hallucinations, biais, dettes techniques (Cui et al., 2024)	Dépendance aux API propriétaires (Rasheed et al., 2024)
Opportunities	Nouveaux modèles SaaS sans humains	Personnalisation rapide pour marchés verticaux
Threats	Fuite de données / attaques backdoor (Wang et al., 2024)	Éthique et responsabilité légale

Table 6.1 – Analyse SWOT d'une équipe 100% IA

6.4 Scénarios d'évolution

Les retours d'expérience et prototypes étudiés suggèrent une trajectoire en trois paliers pour l'adoption des agents IA dans le développement logiciel.

6.4.1 Court terme : collaboration homme–IA

Dans la phase actuelle, les agents agissent avant tout en supplément, comme copilotes :

— génération ou complétion de code, recherche de snippets, écriture de tests unitaires ;

— suggestions de refactorisation et détection précoce de failles courantes.

Le développeur reste le **reviewer critique**, relit et valide chaque changement avant fusion. Zahid et Hussain (2024) soulignent que malgré les capacités des agents en terme de détection de bug, l'intervention humaine reste essentielle pour gérer des problèmes complexes et fortement contextualisés.

6.4.2 Moyen terme : équipes hybrides réduites

Dans une équipe hybride, on pourrait viser un ratio d'un humain pour cinq agents IA, l'humain restant responsable de l'architecture et de la validation métier.

L'humain se concentrerait sur :

— la définition de l'architecture / des priorités produit ;

— l'arbitrage des décisions critiques (sécurité, budget).

— éventuellement la relecture avant déploiement

Les agents IA, orchestrés dans un workflow continu, prennent en charge :

1. le codage et la documentation ;

2. l'exécution automatique de tests et le débogage ;
3. la mise à jour des pipelines CI/CD et du monitoring.

Khan et Daviglus (2025) rapportent que cette configuration réduit le temps de développement tout en maintenant un feedback loop humain aux points de contrôle clés (déploiement, ...).

6.4.3 Long terme : autonomie complète sous supervision minimale

À horizon plus lointain, on peut envisager une **équipe 100 % IA** livrant en continu, avec :

— un contrôle a posteriori : audits périodiques et tests de conformité ;
— des guardrails de sûreté (policies, sandboxing) pour limiter les dérives (Cui et al., 2024).

Les tâches humaines se déplacent vers :

— la gouvernance (définition d'objectifs, gestion des risques) ;
— la validation légale et réglementaire ;
— l'optimisation des coûts d'infrastructure LLM et la mise à jour des modèles.

Ce scénario demeure conditionné à la résolution des menaces identifiées : fuites de données et attaques backdoor (Wang et al., 2024), ainsi qu'à l'instauration d'un cadre de responsabilité clair pour les décisions prises par les agents.

Ouverture éthique Ces perspectives soulèvent néanmoins une question fondamentale : *remplacer entièrement les développeurs humains par des agents IA serait-il éthique, juste et socialement acceptable ?*

Chapitre 7

Conclusion

Ce mémoire est parti d'une interrogation simple : l'automatisation par agents IA peut-elle aller jusqu'à remplacer entièrement une équipe de développement ? Pour y répondre, nous avons d'abord parcouru l'état de l'art des LLMs, décrit les architectures multi-agents et analysé trois prototypes — CodePori (89 % de réussite sur HumanEval), le cadre itératif Agent-Driven Automatic Software Improvement, et les pipelines Auto-DevOps présentés par Khan et Daviglus (2025). Ces études de cas montrent qu'une grande partie des tâches techniques — génération, tests, documentation ou débogage — est déjà couverte avec un niveau de fiabilité oscillant entre 80% et 90% de précision.

L'examen systématique des fonctions clés d'une équipe (de l'analyse d'exigences à la mise en production) révèle toutefois que deux domaines échappent encore largement à l'autonomie : la définition stratégique du produit et la gouvernance qualité. Les agents sont rapides : ils livrent du code exécutable en quelques minutes pour quelques dollars (Rasheed et al., 2024), mais demeurent vulnérables aux hallucinations et aux biais (Cui et al., 2024), ainsi qu'aux attaques de type backdoor ou fuite de données (Wang et al., 2024). Tant que ces risques ne sont pas entièrement maîtrisés, une supervision humaine reste indispensable, au moins a posteriori, pour valider les livraisons et assumer la responsabilité légale.

Notre analyse conduit donc à une réponse nuancée. Oui, le remplacement total est envisageable sur le plan technique ; non, il n'est pas encore acceptable sans trois garanties :

— un dispositif d'alignement solide (guardrails, traçabilité),
— une gestion rigoureuse de la sécurité et de la confidentialité,
— un cadre de gouvernance qui précise qui ou quoi porte la responsabilité des choix d'architecture, de budget et de conformité

En proposant une cartographie fonctionnelle, une synthèse critique des prototypes et une matrice SWOT, ce travail apporte un panorama structuré des points de bascule où l'IA devance déjà l'humain, et des verrous qui subsistent. Ses limites résident principalement dans la rareté de données industrielles ouvertes et l'incertitude sur les coûts d'inférence à grande échelle. Les perspectives logiques sont doubles : d'un côté, la création de benchmarks plus représentatifs (intégrant dette technique et sécurité), de l'autre, l'étude des impacts organisationnels et éthiques d'équipes entièrement IA.

En définitive, la question n'est plus tant "peut-on" automatiser, mais plutôt dans quelles conditions et avec quelles garanties nous choisirons (ou non) de le faire.

Bibliographie

Cui, T., Wang, Y., Fu, C., Xiao, Y., Li, S., Deng, X., Liu, Y., Zhang, Q., Qiu, Z., Li, P., Tan, Z., Xiong, J., Kong, X., Wen, Z., Xu, K., & Li, Q. (2024, janvier 11). Risk Taxonomy, Mitigation, and Assessment Benchmarks of Large Language Model Systems. https://doi.org/10.48550/arXiv.2401.05778

Händler, T. (2023a). A Taxonomy for Autonomous LLM-Powered Multi-Agent Architectures. *KMIS*, 85-98. https://doi.org/10.48550/arXiv.2310.03659

Rasheed, Z., Sami, M. A., Kemell, K.-K., Waseem, M., Saari, M., Systä, K., & Abrahamsson, P. (2024, septembre 17). CodePori : Large-Scale System for Autonomous Software Development Using Multi-Agent Technology. https://doi.org/10.48550/arXiv.2402.01411

Vallecillos Ruiz, F. (2024). Agent-driven automatic software improvement. *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 470–475. https://doi.org/10.1145/3661167.3661171

Zahid, I., & Hussain, I. (2024). Multi-Agent AI Collaboration : Advancing Software Engineering with Autonomous LLMs. https://doi.org/10.13140/RG.2.2.18585.74080

Händler, T. (2023b). Balancing autonomy and alignment : a multi-dimensional taxonomy for autonomous LLM-powered multi-agent architectures. *arXiv preprint arXiv :2310.03659*. https://doi.org/10.48550/arXiv.2310.03659

Wang, S., Zhu, T., Liu, B., Ding, M., Guo, X., Ye, D., Zhou, W., & Yu, P. S. (2024, juin 18). Unique Security and Privacy Threats of Large Language Model : A Comprehensive Survey. https://doi.org/10.48550/arXiv.2406.07973

Olivares, R., Noel, R., Guzmán, S. M., Miranda, D., & Munoz, R. (2024). Intelligent learning-based methods for determining the ideal team size in agile practices [Publisher : MDPI]. *Biomimetics*, *9*(5), 292. https://doi.org/10.3390/biomimetics9050292

Bodaragama, B. D. T., Vipulasiri, D., DI, D. S., & JayakodY, T. (2023). Exploring the Impact of Team Size on Software Quality [Publisher : Authorea]. *Authorea Preprints*. https://doi.org/10.22541/au.168431236.62929306/v1

Ashraf, B., & Talavera, G. (2025). Autonomous Agents in Software Engineering : A Multi-Agent LLM Approach. https://doi.org/10.13140/RG.2.2.22360.61448

Khan, S., & Daviglus, M. (2025). AI-Driven Automation in Agile Development : Multi-Agent LLMs for Software Engineering. https://doi.org/10.13140/RG.2.2.20682.89281

Abbas, G., & Wahab, A. (2024). AI-Driven Agile Development : How Multi-Agent LLMs Optimize Engineering Workflows. https://doi.org/10.13140/RG.2.2.23618.90566

Webographie

Wikipedia. (2025a). *Hallucination (IA)*. https://fr.wikipedia.org/wiki/Hallucination_(intelligence_artificielle)
Wikipedia. (2025b). *SLOC*. https://en.wikipedia.org/wiki/Source_lines_of_code
Wikipedia. (2025c). *Software Metrics*. https://en.wikipedia.org/wiki/Software_metric
Wikipedia. (2025d). *Bus Factor*. https://fr.wikipedia.org/wiki/Facteur_d%27autobus

Chapitre 8

Annexes

Annexe .1 : Mots éliminatoires pour les titres

Afin de garantir la pertinence des articles retenus, les titres comportant l'un des mots suivants sont exclus :

```
ELIMINATORY_WORDS = ["philosophy","sociology","ethics","
   political","policy","society","healthcare","clinical","gender
   ","history","art","religion","linguistics","psychology"]
```