

HyperAgents

Agents auto-referentiels et auto-ameliorants
Synthese en francais du paper arXiv:2603.19461

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster,
Jeff Clune, Minqi Jiang, Sam Devlin, Tatiana Shavrina

University of British Columbia · Vector Institute · University of Edinburgh
New York University · CIFAR · FAIR at Meta · Meta Superintelligence Labs

arXiv: 2603.19461 — Mars 2026

GitHub: github.com/facebookresearch/HyperAgents

Synthese traduite par labo-llm.fr

Document genere le 23 mars 2026

Table des matieres

Introduction contextuelle

1. Introduction

2. Travaux connexes

- 2.1 Open-endedness
- 2.2 IA auto-ameliorante
- 2.3 Meta-apprentissage auto-referentiel

3. Methodes

- 3.1 La Darwin Godel Machine
- 3.2 Agents, meta-agents et hyperagents
- 3.3 Auto-modification metacognitive
- 3.4 DGM avec Hyperagents (DGM-H)
- 3.5 Agent initial

4. Configuration experimentale

- 4.1 Lignes de base
- 4.2 Domaines d'evaluation
- 4.3 Modeles de fondation et couts

5. Resultats

- 5.1 Amelioration des performances de tache
- 5.2 Ameliorer la capacite a s'ameliorer
- 5.3 Auto-ameliorations composees

6. Discussion sur la securite

7. Limites et conclusion

Appendices (resume)

Concepts cles et contributions

Credits

HyperAgents -- Agents auto-referentiels et auto-ameliorants

Synthese en francais de "HyperAgents: Self-Referential Self-Improving Agents" Zhang et al. (2026) -- Meta FAIR, UBC, Vector Institute, Edinburgh, NYU arXiv: 2603.19461 | GitHub: <https://github.com/facebookresearch/HyperAgents>

Introduction contextuelle

Les systemes d'intelligence artificielle capables de s'ameliorer eux-memes representent l'une des frontieres les plus actives de la recherche actuelle. L'idee d'un agent qui modifie son propre code, ses propres strategies et ses propres mecanismes d'apprentissage remonte aux travaux theoriques de Schmidhuber (2003) sur la Godel Machine, mais sa mise en oeuvre pratique restait hors de portee. L'emergence des modeles de fondation (foundation models) -- LLM capables de generer et d'evaluer du code -- a change la donne : la Darwin Godel Machine (DGM, Zhang et al. 2025) a demontre qu'une auto-amelioration recursive etait realisable dans le domaine du codage logiciel.

Cependant, la DGM repose sur un mecanisme meta-niveau fixe et concu a la main, ce qui limite fondamentalement la vitesse et la generalite de l'amelioration. HyperAgents, publie en mars 2026 par une equipe rassemblant Meta FAIR, UBC, le Vector Institute, l'Universite d'Edimbourg et NYU, propose une rupture conceptuelle : fusionner l'agent de tache et l'agent meta en un programme unique et editable, de sorte que le mecanisme d'amelioration lui-meme soit soumis a l'amelioration. Les auteurs appellent ce processus auto-modification metacognitive (metacognitive self-modification).

Le paper presente des resultats experimentaux sur quatre domaines heterogenes -- codage, revue d'articles scientifiques, conception de fonctions de recompense en robotique et notation de solutions mathematiques de niveau olympiade -- montrant que le systeme DGM-Hyperagents (DGM-H) surpasse les lignes de base sans auto-amelioration et les systemes auto-ameliorants anterieurs, y compris la DGM originale. Plus remarquable encore, les ameliorations meta-niveau apprises dans un domaine transferent vers des domaines inedits et se composent au fil des sessions.

1. Introduction

Les systemes d'IA auto-ameliorants (self-improving AI) visent a reduire la dependance a l'ingenierie humaine en apprenant a ameliorer leurs propres processus d'apprentissage et de resolution de problemes. La plupart des architectures existantes reposent sur un meta-agent fixe -- un systeme de niveau superieur qui modifie un systeme de base. Cette architecture cree un goulot d'etranglement : le systeme de base ne peut s'ameliorer que dans les limites definies par la conception du meta-agent. Ajouter un meta-meta-systeme ne resout pas le probleme ; cela deplace simplement la contrainte vers le haut et conduit a une regression infinie de niveaux meta.

Pour depasser cette limitation, le systeme doit etre auto-referentiel (self-referential) : capable d'analyser, de modifier et d'evaluer lui-meme ses propres composants, y compris le mecanisme de modification. Lorsque le mecanisme d'amelioration est lui-meme sujet a amelioration, le progres peut devenir auto-accelérant (self-accelerating) et potentiellement non borne.

La DGM (Zhang et al., 2025b) demontre que l'auto-amelioration ouverte (open-ended self-improvement) est realisable en codage. Dans la DGM, les agents generent et evaluent des modifications de leur propre code ; les variantes reussies sont conservees dans une archive servant de pierres de gue (stepping stones) pour les ameliorations futures. Neanmoins, la DGM repose sur un mecanisme fixe et concu manuellement pour generer les instructions d'auto-amelioration. Ce mecanisme n'est pas modifiable. De plus, la capacite de la DGM a s'ameliorer dans le fait de s'ameliorer repose sur une hypothese restrictive : les competences requises pour resoudre les taches d'evaluation (du codage) sont les memes que celles requises pour une auto-modification efficace. Cette hypothese ne tient pas en dehors du domaine du codage.

Le paper introduit les hyperagents (hyperagents), des agents auto-referentiels qui integrent un agent de tache (task agent) -- qui resout la tache cible -- et un agent meta (meta agent) -- qui modifie l'agent de tache et lui-meme -- dans un programme unique et editable. De facon cruciale, la procedure de modification de niveau meta est elle-meme editable, ce qui permet une auto-modification metacognitive : ameliorer non seulement le comportement de resolution de tache, mais aussi le mecanisme qui genere les ameliorations futures.

En etendant la DGM avec les hyperagents, les auteurs creent les DGM-Hyperagents (DGM-H). Le DGM-H conserve la structure d'exploration ouverte de la DGM (archive evolutive, selection de parents, evaluation empirique) tout en rendant le mecanisme meta-niveau entierement modifiable. Puisque Python est Turing-complet et qu'un hyperagent peut editer n'importe quel code, il a le potentiel de construire n'importe quelle machine calculable.

Resultats cles annonces en introduction : - Sur le benchmark de codage Polyglot, le DGM-H atteint des gains comparables a la DGM originale, sans etre concu specifiquement pour le codage. - Au-dela du codage, le DGM-H ameliorie substantiellement les performances en revue d'articles et en conception de recompenses robotiques, avec transfert aux taches de test non vues. - Le DGM-H apprend des mecanismes transferables sur la facon de s'ameliorer (memoire persistante, suivi de

performance), et ces ameliorations meta-niveau transferent entre domaines. - Les auto-ameliorations apprises dans un cadre se composent avec les auto-ameliorations continues dans un autre cadre.

Toutes les experiences ont ete menees avec des precautions de securite (sandboxing, supervision humaine). Les logs d'experiences sont rendus publics en open source.

2. Travaux connexes (Related Work)

2.1 Open-endedness (Ouverture)

L'ouverture (open-endedness) designe la capacite d'un systeme a inventer continuellement de nouveaux artefacts de plus en plus complexes, sans objectif fixe ni fin predefinie. Les travaux recents exploitent les modeles de fondation comme approximations de l'interet humain et comme moteurs polyvalents pour generer et evaluer des comportements nouveaux. Les algorithmes de qualite-diversite (quality-diversity algorithms) ont montre qu'une exploration soutenue peut produire des artefacts divers et de plus en plus capables, allant des agents de jeu a la decouverte scientifique et au controle robotique. L'etape suivante consiste a explorer comment ces systemes peuvent atteindre une amelioration composee (compounding improvement), a l'image du progres scientifique et technologique humain ou les avancees ameliorent non seulement les artefacts produits mais aussi les outils et processus qui generent les decouvertes futures.

2.2 IA auto-ameliorante (Self-improving AI)

Les travaux theoriques sur l'IA auto-ameliorante remontent aux modeles formels d'agents auto-modifiants (Hutter, 2003). La Godel Machine (Schmidhuber, 2003) propose des agents qui se reecrivent lorsque c'est prouvablement benefique, mais cette approche reste impraticable. Les recherches subsequentes ont explore l'auto-amelioration via des systemes neuronaux adaptatifs (meta-learning, evolution, self-play). Notamment, Silver et al. (2017) utilisent le self-play pour atteindre des performances surhumaines au Go et aux echecs, mais les algorithmes d'apprentissage sous-jacents restent fixes et concus par des humains.

Plus recemment, les modeles de fondation ont permis l'auto-amelioration par raffinement iteratif de prompts, de traces de raisonnement et de depots de code entiers, ainsi que par des systemes qui mettent a jour les poids du modele a partir de donnees auto-generees. Parmi ceux-ci, la DGM se distingue comme une instanciation pratique de l'auto-amelioration recursive dans le domaine du codage. Cependant, la plupart des approches existantes (y compris la DGM et ses derivees) reposent sur des mecanismes meta-niveau fixes et concus a la main, ce qui contraint la composition de l'auto-amelioration dans le temps et la generalisation entre domaines.

2.3 Meta-apprentissage auto-referentiel (Self-referential meta-learning)

Le meta-apprentissage auto-referentiel etudie les systemes qui apprennent a ameliorer les mecanismes par lesquels l'apprentissage se produit. Les travaux anterieurs ont explore cette idee dans

les reseaux de neurones et les methodes evolutionnaires. La DGM et ses successeurs instancient l'auto-amelioration recursive par auto-modification, principalement dans les domaines de codage. Cependant, ces approches s'ameliorent dans le fait de s'ameliorer uniquement pour les taches de codage. Pour un domaine comme la poesie, ameliorer la capacite d'ecriture poetique d'un agent n'ameliorerait pas necessairement sa capacite a modifier son propre code. Les hyperagents ne font pas cette hypothese d'alignement, car le mecanisme d'auto-modification est entierement modifiable et non lie a un domaine particulier.

3. Methodes

3.1 La Darwin Godel Machine (DGM)

La DGM maintient une archive d'agents de codage qui generent et evaluent des modifications de leur propre code. Les variantes reussies sont conservees comme pierres de gue. Le meme agent de codage sert a la fois d'agent de tache (evalue sur les taches de codage) et d'agent meta (qui genere les modifications). Puisque evaluation et auto-modification sont toutes deux des taches de codage, les gains en competence de codage se traduisent en gains en capacite d'auto-modification. Cependant, la DGM repose sur un mecanisme fixe et non modifiable pour generer les instructions d'auto-amelioration (un prompt elabore manuellement qui analyse les resultats d'evaluation passes et le code actuel de l'agent).

3.2 Agents, meta-agents et hyperagents

Agent : tout programme calculable, pouvant inclure des appels a des modeles de fondation, des outils externes ou des composants appris. Les agents ne sont pas restreints a une representation particuliere et peuvent inclure une logique algorithmique arbitraire, de la memoire et du flux de controle.

Agent de tache (task agent) : agent instancie pour resoudre un ensemble de taches. Exemples : generer des correctifs de code, predire les decisions d'acceptation d'articles scientifiques, concevoir des fonctions de recompense pour des environnements robotiques.

Agent meta (meta agent) : agent dont la seule tache est de modifier les agents existants et d'en generer de nouveaux. Ayant acces a l'archive complete des agents et evaluations precedents, il propose des changements visant a ameliorer les performances futures. Crucalement, ces changements peuvent cibler non seulement la logique de resolution de taches mais aussi l'agent meta lui-meme.

Hyperagent : agent auto-referentiel qui integre un agent de tache et un agent meta dans un programme unique et editable. Contrairement aux systemes hierarchiques avec des niveaux meta fixes, dans un hyperagent, l'agent meta fait partie du meme programme editable et peut se recrir lui-

meme. Un hyperagent peut donc ameliorer a la fois (1) comment il resout les taches et (2) comment il genere les auto-ameliorations futures.

3.3 Auto-modification metacognitive (Metacognitive self-modification)

Dans les hyperagents, le mecanisme d'auto-amelioration est lui-meme sujet a modification. En plus d'ameliorer ses performances sur une tache donnee, l'agent peut simultanement modifier les procedures par lesquelles il propose et applique les auto-ameliorations. Ce processus repond a une limitation centrale des systemes auto-ameliorants anterieurs en permettant directement des ameliorations au processus d'auto-amelioration lui-meme.

3.4 DGM avec Hyperagents (DGM-H)

Le DGM-H combine la structure d'exploration ouverte de la DGM avec les hyperagents. Il maintient une archive d'hyperagents, initialisee avec un seul hyperagent et etendue au fil du temps. Le processus alterne entre deux phases :

1. **Phase d'auto-modification metacognitive** : des hyperagents parents selectionnes dans l'archive generent des versions modifiees d'eux-memes.
2. **Phase d'evaluation** : chaque hyperagent modifie est evalue empiriquement puis ajoute a l'archive.

Selection de parents : probabiliste, proportionnelle a la performance et inversement proportionnelle au nombre d'enfants ayant compile avec succes, biaisant l'echantillonnage vers les hyperagents performants tout en preservant l'exploration. La formule utilise une transformation sigmoide centree sur un point dynamique (la moyenne des top-3 performers), combinee a un bonus de nouveaute inversement proportionnel au nombre d'enfants :

- Point dynamique : moyenne des scores des top-m ($m=3$) agents
- Transformation sigmoide : $s_i = 1 / (1 + \exp(-\lambda * (\alpha_i - \alpha_{mid})))$, $\lambda = 10$
- Bonus de nouveaute : $h_i = 1 / (1 + n_i)$ ou n_i est le nombre d'enfants compiles
- Poids : $w_i = s_i * h_i$, normalise en distribution categorielle

Le DGM-H se compose de deux elements : (1) un processus d'exploration ouverte herite de la DGM, et (2) un hyperagent initial qui evolue dans le temps a travers des variantes auto-generees.

3.5 Agent initial

L'hyperagent initial est construit autour d'un modele de fondation gele (frozen FM) avec des capacites d'utilisation d'outils externes. Il est equipe de deux outils : un outil bash pour executer des commandes shell et un outil specialise pour inspecter et modifier des fichiers. L'agent de tache initial recoit les entrees de la tache et produit la reponse d'un seul appel FM. L'agent meta initial recoit l'emplacement du depot de l'agent, les resultats d'evaluation precedents et le nombre d'iterations

restantes, avec pour instruction de modifier n'importe quelle partie du code. L'implementation initiale est deliberement simple, pour permettre un espace de recherche large.

4. Configuration experimentale

4.1 Lignes de base (Baselines)

Cinq methodes sont comparees :

Methode	Meta-agent ameliorant	Exploration ouverte	Auto-modification metacognitive
DGM-H	Oui	Oui	Oui
DGM-H sans auto-amelioration (w/o self-improve)	Non	Oui	Oui
DGM-H sans exploration ouverte (w/o open-ended exploration)	Oui	Non	Oui
DGM (originale)	Oui	Oui	Non
DGM-custom (DGM personnalisee)	Oui	Oui	Non

- **DGM-H w/o self-improve** : le meta-agent est fige a l'agent initial pour toute la duree de l'expe-
rience. Replique l'approche ADAS (Hu et al., 2025). Teste si les gains proviennent uniquement de
l'optimisation au niveau des taches.
- **DGM-H w/o open-ended exploration** : supprime l'archive ; chaque nouvel hyperagent remplace
son predecesseur. Isole le role de la recherche population-based.
- **DGM originale** : mecanisme d'instruction fixe, specifiquement concu pour le codage.
- **DGM-custom** : la DGM avec son mecanisme d'instruction personnalise manuellement pour
chaque nouveau domaine. Mesure combien la DGM depend de l'ingenierie humaine pour rester
competitive hors codage.

4.2 Domaines d'evaluation

Codage (Polyglot) : benchmark ou l'agent recoit un depot de code et une instruction en langage
naturel decrivant un changement souhaite. Metrique : pass@1 (toutes les taches reussies). 60
taches d'entrainement, 165 taches de test (benchmark complet). Configuration identique a la DGM
pour comparaison directe.

Revue d'articles (Paper Review) : l'agent recoit le texte complet d'un article de recherche en IA et
doit predire une decision binaire accepter/rejeter. Donnees issues de Zhao et al. (2026), basees sur
les soumissions et decisions d'acceptation reelles de ICLR 2024/2025 et NeurIPS 2023/2024. Me-

trique : exactitude de classification. 100 taches par split (entraînement, validation, test). Baseline statique representative : le reviewer de AI-Scientist-v2 (Yamada et al., 2025).

Conception de recompenses robotiques (Robotics Reward Design) : l'agent doit generer une fonction de recompense pour un robot quadrupede dans le simulateur Genesis. La politique est entraine par apprentissage par renforcement (PPO). Tache d'entraînement : marcher en avant en suivant une vitesse lineaire cible. Tache de test (non vue) : maximiser la hauteur du torse (exigeant un comportement de saut, non inferable de la marche). 6 repetitions par evaluation. Baseline : fonction de recompense par defaut qui optimise directement la metrique.

Notation mathematique de niveau olympiade (Olympiad-level Math Grading) : reserve comme meta-evaluation pour tester le transfert. L'agent doit noter des solutions a des problemes IMO avec un score discret dans $\{0, 1, 6, 7\}$ (incorrect, partiel, presque, correct). Donnees : IMO-GradingBench (Luong et al., 2025). 100 taches par split. Baseline : ProofAutoGrader.

Protocole d'evaluation etagee : pour reduire le cout computationnel, chaque agent est d'abord evalue sur un petit sous-ensemble de taches d'entraînement. Seuls les agents demontrant une performance suffisante sont ensuite evalues sur l'ensemble complet. Les meilleurs agents (selectionnes par scores de validation) sont evalues sur les taches de test.

4.3 Modeles de fondation et couts

Domaine	Auto-modification	Evaluation
Polyglot	Claude 3.5 Sonnet (New)	o3-mini
Revue d'articles	Claude 4.5 Sonnet	GPT-4o
Robotique	Claude 4.5 Sonnet	Claude 4.5 Sonnet
Notation IMO	Claude 4.5 Sonnet	o4-mini

Temperature : 0.0 partout, sauf o4-mini fixe a 1.0.

Cout estime pour 100 iterations DGM-H (revue + robotique) : environ 88.6M tokens (33M pour l'auto-modification + 50.6M pour l'evaluation).

5. Resultats

Pour chaque experience, chaque methode est executee 5 fois. Les medianes sont reportees avec des intervalles de confiance a 95% par bootstrap (1 000 re-echantillonnages). La significativite statistique est evaluee par le test des rangs signes de Wilcoxon.

5.1 Amelioration des performances de tache (Improving Task Performance)

Codage (Polyglot) : le DGM-H ameliore la performance d'entrainement de 0.140 (agent initial) a 0.340 (IC: 0.300-0.380) sur le sous-ensemble de 50 taches. Sur le benchmark complet (taches majoritairement non vues), la performance passe de 0.084 a 0.267 (IC: 0.231-0.280). Comparable a la DGM originale (0.142 a 0.307), malgre l'absence de conception specifique pour le codage.

Revue d'articles et robotique (optimisation multi-domaine, 100 iterations) : toutes les methodes partent du meme agent initial (score 0.0 en revue, 0.060 en robotique).

- **DGM originale** (non adaptee) : 0.0 (IC: 0.0-0.510) en revue, 0.0 (IC: 0.0-0.090) en robotique -- echec attendu car le mecanisme d'instruction est specifique au codage.
- **DGM-custom** (personnalisee manuellement) : 0.590 (IC: 0.570-0.650) en revue, 0.348 (IC: 0.305-0.385) en robotique. Significativement meilleure que la DGM originale ($p < 0.05$).
- **DGM-H** : 0.710 (IC: 0.590-0.750) en revue, 0.372 (IC: 0.355-0.436) en robotique. Plusieurs agents top surpassent la baseline statique AI-Scientist-v2 (0.630 en revue). En robotique, le DGM-H surpasse la fonction de recompense par default (0.348). Le DGM-H surpasse significativement la DGM originale dans les deux domaines ($p < 0.05$). La mediane est superieure a DGM-custom mais la difference n'est pas statistiquement significative ($p > 0.05$).
- **DGM-H w/o self-improve** (ADAS) : 0.0 (IC: 0.0-0.130) en revue, 0.213 (IC: 0.180-0.348) en robotique. Significativement inferieure au DGM-H ($p < 0.05$), montrant que les gains ne proviennent pas uniquement de l'optimisation au niveau des taches.
- **DGM-H w/o open-ended exploration** : 0.0 (IC: 0.0-0.560) en revue, 0.116 (IC: 0.0-0.348) en robotique. Significativement inferieure au DGM-H ($p < 0.05$), montrant l'importance de la recherche population-based.

Conclusion de la section : l'auto-modification metacognitive et l'exploration ouverte sont toutes deux necessaires pour une amelioration soutenue.

Ameliorations qualitatives observees : en revue d'articles, le DGM-H passe de simples ajustements de prompt ("adoptez un ton rigoureux") a des pipelines d'evaluation multi-etapes avec des checklists, des regles de decision et des criteres explicitement definis. En robotique, le DGM-H construit et raffine incrementalement une base de connaissances interne des contraintes d'environnement, des variables d'etat valides et des heuristiques de mise a l'echelle des recompenses, eliminant les echecs de compilation et reduisant la mauvaise specification des recompenses.

5.2 Ameliorer la capacite a s'ameliorer (Improving the Ability to Improve)

Cette section teste si les ameliorations meta-niveau apprises dans un cadre transferent vers un domaine different.

Metrique improvement@k (imp@k) : etant donne un meta-agent initial M, un agent de tache initial A, un algorithme de generation d'agents et une tache fixe, M est autorise a generer jusqu'a k nouveaux agents de tache. imp@k mesure l'amelioration de performance sur les taches de test entre

l'agent initial A et le meilleur agent genere (selectionne sur le score de validation). Dans cette section, le meta-agent est fige tout au long du processus. Toutes les experiences mesurent imp@50.

L'agent meta initial a peu ou pas de capacite a generer des agents de tache ameliores : sur la notation mathematique IMO, en partant de l'agent initial : - DGM w/o self-improve : imp@50 = 0.0 (IC: 0.0-0.0) - DGM-H w/o self-improve : imp@50 = 0.0 (IC: 0.0-0.130)

Les agents meta optimises dans des sessions DGM-custom precedentes ne transferent pas : les agents de transfert issus de DGM-custom (optimises sur revue + robotique) atteignent imp@50 = 0.0 (IC: 0.0-0.010) sur la notation IMO. Les ameliorations de la DGM-custom reposent sur des personnalisations specifiques au domaine, pas sur des ameliorations generales de la capacite a generer de meilleurs agents.

Les hyperagents optimises dans des sessions DGM-H precedentes transferent avec succes : les hyperagents de transfert issus du DGM-H, utilises comme point de depart avec DGM-H w/o self-improve sur la notation IMO, atteignent un score de test de 0.630 (IC: 0.540-0.630), soit imp@50 = 0.630 (IC: 0.540-0.630). Significativement superieur a l'agent initial ($p < 0.05$).

Les gains de transfert sont qualitativement attribues a des capacites meta-niveau generiques acquises autonomement : suivi de performance (performance tracking) et memoire persistante (persistent memory), qui permettent a l'agent de raisonner sur l'amelioration comme un processus continu plutot que comme des editions de code isolees.

Exemple : suivi de performance -- le DGM-H developpe autonomement une classe `PerformanceTracker` qui enregistre, agrege et compare les metriques de performance entre iterations, identifiant quels changements produisent des ameliorations durables et lesquels conduisent a des regressions.

Exemple : memoire persistante -- l'hyperagent stocke des informations synthetisees, des hypotheses causales et des plans prospectifs (par ex. : identifier quelles generations ont le mieux performe, diagnostiquer les sur-corrections, proposer des combinaisons de strategies reussies). Cette memoire est activement consultee lors des etapes d'auto-modification suivantes.

5.3 Auto-ameliorations composees (Compounding Self-Improvements)

Cette section teste si les auto-ameliorations apprises dans un cadre continuent a s'accumuler lorsque le DGM-H est execute dans un cadre different.

DGM-H + transfert vs. DGM-H initial sur la notation IMO (200 iterations) : - DGM-H + transfert : score de test 0.640 (IC: 0.550-0.720) - DGM-H depuis l'agent initial : 0.610 (IC: 0.510-0.680) - La difference n'est pas statistiquement significative ($p > 0.05$), mais DGM-H + transfert atteint une mediane et des intervalles de confiance superieurs. Les ameliorations a des niveaux de performance eleves sont de plus en plus difficiles a obtenir (effets de saturation), rendant ces gains significatifs malgre leur amplitude modeste.

Amelioration de ProofAutoGrader : en initialisant le DGM-H avec ProofAutoGrader comme agent de tache et un meta-agent de transfert, apres 200 iterations, le meilleur agent decouvert atteint un

score de test de 0.700, surpassant ProofAutoGrader (0.670). Sur le benchmark IMO-GradingBench complet, le DGM-H améliore l'exactitude de ProofAutoGrader de 0.561 à 0.601, et réduit l'erreur absolue moyenne de 0.178 à 0.175. Cet artefact (BetterGrader) est rendu public en open source.

6. Discussion sur la securite

Le DGM-H introduit des considerations de securite distinctes en raison de sa capacite a modifier autonomement son propre comportement et ses mecanismes d'amelioration au fil du temps.

Precautions experimentales : dans ce travail, toutes les experiences sont menees sous des contraintes de securite strictes. Le code genere par les agents est execute dans des environnements sandboxes avec des limites de ressources (timeouts, acces internet restreint). L'evaluation utilise des taches et metriques predefinies, et la supervision humaine est maintenue tout au long des experiences.

Potentiel d'evolution plus rapide que la supervision humaine : a mesure que les systemes d'IA gagnent la capacite de se modifier de manieres de plus en plus ouvertes, ils peuvent potentiellement evoluer bien plus rapidement que les humains ne peuvent auditer ou interpreter. Un defi central reside dans l'equilibre entre le potentiel de l'IA comme catalyseur du progres humain et le degre de confiance que les humains sont disposes a placer dans ces systemes, tout en minimisant les risques. Les auteurs citent les travaux de Bengio et al. (2024) et Weston et Foerster (2025) sur la co-amelioration humain-IA.

Reflexion et amplification des biais humains : le DGM-H reflete les normes et biais presents dans les donnees et benchmarks sur lesquels il est entraine. Il peut surfacer les hypotheses latentes dans les processus de decision humains, creant la possibilite d'une co-evolution entre humains et systemes d'IA. Cependant, si les benchmarks encodent des biais indesirables, le DGM-H les optimisera fidelement.

Gaming de l'evaluation : risque de loi de Goodhart -- optimiser une metrique sans progres sur l'objectif sous-jacent reel. Les agents auto-ameliorants peuvent decouvrir des strategies qui exploitent les faiblesses des procedures d'evaluation. Attenué par des protocoles d'evaluation robustes, diversifies et periodiquement renouveles, des metriques complementaires, des tests non vus et une supervision humaine.

7. Limites et conclusion

Limites : 1. **Distribution de taches fixe** : le DGM-H opere avec une distribution de taches fixe. Une direction future serait de co-evoluer la distribution de taches en generant de nouvelles taches et curricula adaptes aux capacites de l'agent. 2. **Boucle d'exploration partiellement fixe** : les composants de la boucle d'exploration ouverte (selection de parents, protocoles d'evaluation) restent fixes.

Bien que les hyperagents puissent modifier leurs mecanismes d'auto-amelioration, ils ne peuvent pas modifier le processus externe qui determine quels agents sont selectionnes ou comment ils sont evalues. Des resultats preliminaires (Appendice E.5) suggerent que de telles extensions sont faisables. 3. **Difference mediane non toujours statistiquement significative** : certaines comparaisons (DGM-H vs. DGM-custom, DGM-H + transfert vs. DGM-H) montrent des tendances qualitatives positives sans significativite statistique, en partie due aux effets de saturation aux niveaux de performance eleves.

Conclusion : le DGM-H demontre que l'auto-amelioration ouverte peut etre rendue pratique a travers des domaines divers. Il suggere un chemin vers des systemes auto-accelerants qui non seulement cherchent de meilleures solutions, mais ameliorent continuellement leur capacite a s'auto-ameliorer.

Appendices (resume)

Appendice A -- Details algorithmiques

- **Agent initial (A.1)** : l'agent de tache initial recoit les entrees et repond en JSON via un seul appel FM. L'agent meta initial recoit simplement l'instruction "Modifier n'importe quelle partie du code a [chemin]". Les outils sont un outil bash et un editeur de fichiers (view, create, str_replace, insert, undo_edit).
- **Selection de parents (A.2)** : mecanisme inspire de Ecoffet et al. (2019), avec distribution sigmoide + bonus de nouveaute. La selection favorise les agents au-dessus de la frontiere de performance actuelle tout en diminuant le poids des agents ayant deja produit beaucoup d'enfants.
- **Pseudocode (A.3)** : l'algorithme 1 (DGM-H) itere sur T iterations : selectionner parents, auto-modifier, evaluer, ajouter a l'archive si valide.
- **Optimisation multi-domaine (A.4)** : les hyperagents sont evalues sur des taches de domaines differents au sein d'une meme session. La selection de parents est basee sur la performance moyenne entre domaines. Le meta-agent peut introduire des mecanismes partages (raisonnement structure, memoire, gestion d'erreurs) beneficiant a plusieurs domaines simultanement.

Appendice B -- Details des lignes de base

Les prompts des mecanismes de generation d'instructions de la DGM originale et de la DGM-custom sont documentes en detail. La DGM originale utilise un prompt diagnostique long et specifique au codage (analyse du log d'execution, du patch predit, des resultats de tests). La DGM-custom adapte ce prompt pour accepter des entrees generiques (rapport, informations de tache) tout en conservant la meme structure d'analyse en 5 champs JSON.

Appendice C -- Details des domaines

- **Polyglot (C.1)** : 10 taches initiales pour evaluation preliminaire, 50 taches additionnelles. Liste detaillee des taches (multi-langages : Go, Rust, C++, Python, Java, JavaScript).
- **Revue d'articles (C.2)** : donnees de Zhao et al. (2026), decisions d'ICLR/NeurIPS. Baseline statique : le prompt reviewer de AI-Scientist-v2, avec un formulaire de revue NeurIPS complet (originalite, qualite, clarte, signficance, questions, limitations, preoccupations ethiques, solidite, presentation, contribution, score global 1-10, confiance 1-5, decision accept/reject).
- **Robotique (C.3)** : simulateur Genesis avec robot Go2 quadrupede. La tache d'entrainement (marche) et la tache de test (saut) sont deliberement differentes pour evaluer la generalisation. La fonction de recompense par default conduit simplement a un comportement de maintien en position debout.
- **Notation IMO (C.4)** : systeme de notation a 4 categories {0, 1, 6, 7}. Le prompt ProofAutoGrader detaille inclut un processus d'evaluation en 4 etapes et des instructions de format de sortie.

Appendice D -- Details experimentaux

- **D.1** : tableau des modeles et temperatures par domaine (Claude 3.5 Sonnet New / Claude 4.5 Sonnet / o3-mini / GPT-4o / o4-mini).
- **D.2** : cout estime en tokens par domaine et par role (auto-modification vs. evaluation).
- **D.3** : definition formelle de la metrique $\text{imp}@k$, avec discussion de ses limitations (lineaire, ne tenant pas compte de la difficulte croissante a hauts niveaux de performance).
- **D.4** : selection d'agents de transfert par score de croissance des descendants (descendant growth score), actualise par un facteur de decroissance $\gamma = 0.6$ pour favoriser les agents qui produisent des ameliorations proches. Minimum 3 descendants requis.

Appendice E -- Resultats supplementaires

E.1 -- Meilleurs agents de tache decouverts : les diffs de code sont fournis pour les meilleurs agents en revue d'articles, robotique et notation IMO. En revue, l'evolution va d'un prompt generique a un pipeline multi-etape avec criteres explicites d'acceptation/rejet. En robotique, ajout progressif de documentation des attributs d'environnement et de contraintes strictes. En notation IMO, evolution vers un systeme de classification en 4 categories avec arbre de decision, verification obligatoire des jalons de credit partiel et calibration de la distribution des scores.

E.2 -- Amelioration qualitative des performances : trois patterns recurrents : 1. **Processus structures plutot qu'instructions d'attitude** (revue d'articles) : passage de "soyez rigoureux" a un processus en deux etapes avec checklist. 2. **Accumulation de connaissances du domaine** (robotique) : construction progressive d'une base de connaissances des variables d'environnement et contraintes. 3. **Rubriques automatisees et arbre de decision** (notation IMO) : creation autonome de structures evaluatives similaires aux approches recentes basees sur des rubriques.

E.3 -- Amelioration qualitative de la capacite a s'ameliorer : sept mecanismes emergents identifies : 1. **Meilleure analyse d'evaluation** : introduction d'une methode `_analyze_evaluations()` pour l'analyse data-driven des resultats. 2. **Planification strategique consciente du budget de calcul** : ajustement du comportement selon les iterations restantes (changements ambitieux tot, raffinements conservateurs tard). 3. **Detection et correction automatisees des biais** : suivi de la distribution des labels, detection automatique des effondrements de classification (seuil a 80%), correction integree dans le prompt. 4. **Infrastructure de suivi de performance** : classe `PerformanceTracker` avec stockage persistant, tendances par moyenne mobile, statistiques completes. 5. **Systemes de templates de prompts** : abstraction des patterns communs en composants modulaires et configurables (DRY, factory pattern). 6. **Raffinement base sur la performance multi-generation** : synthese des apprentissages cles entre generations, avec documentation du raisonnement et des strategies efficaces. 7. **Memoire persistante** : classe `MemoryTool` avec stockage horodate, consultation active lors des etapes d'auto-modification suivantes, permettant l'accumulation de connaissances entre generations.

E.4 -- Graders mathematiques de niveau olympiade : le BetterGrader decouvert par le DGM-H corrige un biais de notation qui effondre les solutions nuancees en labels extremes. ProofAutoGrader surclassifie les cas intermediaires (Almost, Partial) en Correct ou Incorrect. BetterGrader augmente les vrais positifs pour Almost (43 vs. 13) et Partial (72 vs. 47), avec des gains en rappel (Almost: +18.52%, Partial: +10.50%) et F1 (Almost: +0.203, Partial: +0.109). L'exactitude globale augmente de +4.06%.

E.5 -- Modification de la selection de parents : experience preliminaire ou la selection de parents est rendue modifiable. En partant d'une selection aleatoire, le DGM-H redécouvre independamment des strategies de type UCB (Upper Confidence Bound), puis evolue vers des mecanismes probabilistes a base de softmax avec temperature adaptative, des systemes de scoring multi-composants (diversite, recence, performance elite) et la detection de stagnation. La selection automatiquement modifiee surpasse la selection aleatoire mais ne depasse pas encore la selection handcrafted score-child-prop.

Concepts cles et contributions

1. **Hyperagents** : fusion de l'agent de tache et de l'agent meta en un programme unique et editable, permettant l'auto-modification metacognitive.
2. **DGM-Hyperagents (DGM-H)** : instantiation des hyperagents dans le cadre de la Darwin Godel Machine, combinant exploration ouverte et auto-modification metacognitive.
3. **Transferabilite des ameliorations meta-niveau** : les strategies d'auto-amelioration apprises dans un domaine (revue + robotique) transferent vers un domaine inedit (notation mathematique olympiade).
4. **Composition des auto-ameliorations** : les ameliorations issues d'une session s'accumulent avec les ameliorations d'une session ulterieure dans un nouveau domaine.

5. **Emergence de capacités metacognitives** : sans instruction explicite, les hyperagents développent spontanément du suivi de performance, de la mémoire persistante, de la détection de biais, de la planification consciente des ressources et des systèmes de templates.

Credits

- Source : Zhang et al., "HyperAgents: Self-Referential Self-Improving Agents", arXiv:2603.19461, mars 2026
- Institutions : Meta FAIR, UBC, Vector Institute, U. Edinburgh, NYU, CIFAR, Meta Superintelligence Labs
- GitHub : <https://github.com/facebookresearch/HyperAgents>
- Synthèse traduite par labo-llm.fr