

De l’outil au scénario : génération automatique de benchmarks pour agents conversationnels

Alexandre Gallot¹ Louis Jourdain¹

(1) ChapsVision, Paris, France

{agallot, ljourdain}@chapsvision.com

RÉSUMÉ

L’évaluation des agents conversationnels dotés d’outils réels reste coûteuse et difficile à généraliser. Nous proposons un système de génération automatique de scénarios d’évaluation synthétiques, fondé sur la modélisation de l’espace d’exécution comme un graphe de transition pondéré. À partir des seuls outils de l’agent, le pipeline audite chaque outil, construit un automate à états finis, pondère les transitions par une chaîne de Markov du premier ordre, puis hydrate des trajectoires échantillonnées en scénarios complets via des appels réels aux outils. Appliqué à delhIA (7 outils Météo-France), le système produit 50 tâches couvrant 6 outils (5,7 appels en moyenne) et est déployé sur 10 agents, 60 outils dans des domaines variés.

ABSTRACT

From Tool to Scenario : Automatic Benchmark Generation for Conversational Agents

Evaluating conversational agents equipped with real tools remains costly and hard to generalize. We propose a system for the automatic generation of synthetic evaluation scenarios, based on modeling an agent’s execution space as a weighted transition graph. Given only the agent’s tools, our pipeline audits each tool, builds a finite-state automaton, weights transitions via a first-order Markov chain, and hydrates sampled trajectories into complete scenarios via live tool calls. Applied to delhIA (7 Météo-France tools), the system produces 50 tasks covering 6 tools (avg. 5.7 calls), deployed across 10 agents and 60 tools in diverse domains.

MOTS-CLÉS : agent conversationnel, évaluation, données synthétiques, chaîne de Markov, LLM.

KEYWORDS: conversational agent, evaluation, synthetic data, Markov chain, LLM.

1 Introduction

Les grands modèles de langue (LLMs) ont étendu leur domaine d’action bien au-delà de la génération de texte : capables d’appeler des API externes, de raisonner sur des résultats intermédiaires et de dialoguer sur plusieurs tours, ils constituent le cœur de systèmes agentiques déployés en production (Wei *et al.*, 2022; Yao *et al.*, 2023). Ces systèmes sont cependant difficiles à évaluer : les métriques statiques (pass@k, latence) ne capturent pas la richesse d’une trajectoire multi-tours, et l’annotation humaine ne suit plus le développement des modèles (Yehudai *et al.*, 2025). Les *benchmarks* existants restent liés à des domaines *ad hoc* (*sandbox*) dont les scénarios ont été conçus et annotés à la main (Yao *et al.*, 2024; Patil *et al.*, 2025) : toute nouvelle configuration d’agent à évaluer exige un effort de construction de corpus hors de portée d’une équipe ordinaire. *Peut-on générer*

automatiquement un benchmark à partir de la seule spécification des outils ? Notre réponse repose sur une observation structurelle : les séquences d’appels d’un agent conversationnel forment des *chemins* dans un automate fini sur l’espace des outils, dont les transitions peuvent être pondérées par estimation markovienne. On peut donc construire ce graphe à partir des seules spécifications, y échantillonner des trajectoires diversifiées et hydrater chacune en scénario complet (persona, appels attendus, critères de succès) à partir des vraies réponses API. Déployé sur **10 domaines open source** (Yao *et al.*, 2024; Barres *et al.*, 2025) dont **delhIA** (7 outils Météo-France), il produit 50 tâches évaluées sur 3 modèles.

Nos contributions s’articulent autour de trois axes. Sur le plan de la **formalisation**, nous modélisons l’espace de comportements d’un agent comme un graphe de transition markovien construit automatiquement à partir des spécifications d’outils. Nous proposons ensuite un **pipeline bout-en-bout** — de la spécification au scénario d’évaluation complet — sans annotation humaine et reproductible sur toute API documentée. Enfin, des **résultats empiriques** obtenus sur delhIA (50 tâches, 3 modèles) montrent que les trajectoires synthétiques révèlent des lacunes invisibles aux métriques agrégées.

2 Contexte et travaux connexes

2.1 Du modèle à l’agent

L’arrivée des modèles d’IA générative a multiplié le nombre de tâches de NLP réalisables sans entraînement supplémentaire (Wei *et al.*, 2022), mais l’effort d’évaluation n’a pas suivi cette diversification rapide, laissant sans garanties le déploiement de systèmes génératifs sur des tâches complexes. Ce décalage s’est accentué avec l’essor de l’IA agentique. Depuis l’introduction de la génération structurée (OpenAI, 2024), les LLMs ne sont plus seulement des générateurs de texte mais des prédicteurs d’action (Yao *et al.*, 2023) : un agent IA se définit comme un système autonome qui perçoit son environnement, le modifie au moyen d’outils, et retourne une réponse à l’utilisateur (Jiang *et al.*, 2026). Ce paradigme déplace l’objet même de l’évaluation : il ne s’agit plus de juger un résultat final, mais une *trajectoire*, la séquence de décisions et d’appels d’outils par laquelle l’agent atteint (ou manque) son objectif. Or une trajectoire n’admet pas de solution optimale unique, et les agents évoluent dans des environnements dynamiques (APIs, MCP) qui résistent à la constitution de jeux de données annotés stables. À ces difficultés générales s’ajoute la propagation de l’erreur dans le système, qui impose une évaluation à la fois *intrinsèque* (par composant) et *extrinsèque* (résultat global), tandis que l’observabilité des décisions prises par le LLM au cœur de l’agent devient un enjeu central.

2.2 Évaluation des agents conversationnels

Parmi la diversité croissante des architectures agentiques (LLM seul, LLM avec outils, agent ReAct, systèmes multi-agents), l’agent conversationnel occupe une position singulière : la plus demandée par l’industrie, la plus adaptable à des domaines variés, et la plus étudiée, les travaux antérieurs en NLP sur le dialogue ayant produit des corpus et métriques partiellement transposables. Ces ressources restent néanmoins insuffisantes, et les surveys récents (Mohammadi *et al.*, 2025; Yehudai *et al.*, 2025) ne fournissent pas de conseils pratiques. La littérature se fragmente en trois tendances, chacune insatisfaisante pour les créateurs d’agents :

- Des études évaluant les capacités générales d’un modèle sur des tâches de connaissance (Mialon *et al.*, 2023; Froger *et al.*, 2026);
- Des études évaluant des agents sur des tâches très spécifiques (Kulkarni *et al.*, 2025; Zhu *et al.*, 2025);
- Des études mesurant les capacités agencives des LLMs (Qin *et al.*, 2023; Patil *et al.*, 2025), principalement utilisées par les concepteurs de modèles.

Aucun de ces travaux ne fournit au praticien un cadre pour évaluer l’agent qu’il déploie. L’usage d’un LLM comme juge (Gu *et al.*, 2025) constitue une piste complémentaire, mais souffre de risques de calibration dans des contextes spécialisés. Le *benchmark* classique dans le domaine des agents conversationnels est TAU-bench (Yao *et al.*, 2024), qui évalue la capacité d’un LLM assistant à gérer une interaction avec un utilisateur simulé par LLM, dans plusieurs domaines prédéfinis. Rapidement devenu une référence pour les modèles frontières¹, il couvre deux domaines (*airline* et *retail*) modélisant des *chatbots* de service après-vente, avec des politiques de service à respecter et des scénarios utilisateur souvent adversariaux. Sa contribution principale est l’introduction de *datasets* d’évaluation structurés, vérifiant pour chaque simulation si les appels d’outils attendus ont été effectués, si les informations requises ont été communiquées à l’utilisateur et si la base de données a été correctement modifiée. TAU introduit donc une évaluation allant au delà de métriques figées *via* la notion de scénario ; un agent réussit quand il suit bien le scénario correct. TAU² (Barres *et al.*, 2025) prolonge cette approche en testant la coopération entre l’assistant et l’utilisateur lorsque tous deux peuvent agir sur l’environnement.

2.3 Au-delà des sandbox

TAU-bench souffre néanmoins de limitations importantes. Des erreurs et incohérences ont été repérées dans ses *datasets* (Cuadron *et al.*, 2025). Ses trois domaines restent volontairement simplifiés par rapport à des cas d’usage réels, et le *benchmark* ne se généralise pas : bien que les auteurs prévoient une extension à d’autres domaines, ni le code ni la méthodologie ne sont fournis (Barres *et al.*, 2025). Fondamentalement, TAU-bench évalue les LLMs dans un environnement agencive contrôlé, plutôt que les agents en conditions réelles. L’approche de Meta (Froger *et al.*, 2025) tente de dépasser ces critiques en donnant aux agents accès à de vrais outils dynamiques, mais elle exige que l’agent soit formaté selon un format propriétaire. Sa généralisation s’avère de surcroît prohibitivement coûteuse : GAIA 2 (Froger *et al.*, 2026) a mobilisé 15 ingénieurs pendant plusieurs mois, au point qu’il faudrait davantage de ressources pour évaluer un agent que pour le déployer. Il existe donc un réel besoin de méthodes simples pour créer des corpus d’évaluation.

2.4 Généraliser l’évaluation par la génération

Adapter une évaluation d’IA générative ou agencive à un cas pratique particulier représente une partie significative du temps de développement (*eval-driven dev*) et a un coût important. Un des ingrédients clé d’une évaluation est le *dataset* associant une requête faite au système et la réponse attendue. Or, surtout depuis la montée en performance des LLMs, il est de plus en plus courant de recourir à des données synthétiques pour augmenter, voire créer ces *datasets*. Cela a été exploré pour le RAG (Es *et al.*, 2024; Jourdain & Hellal, 2025), où à partir des documents de la base de connaissances on génère automatiquement des questions pour vérifier si le système retrouve l’information pertinente.

1. <https://taubench.com/#leaderboard>

Cette approche pose des questions sur l'utilisabilité pratique des données générées. Elles doivent être de haute qualité, correspondre à des cas d'usage réels et être suffisamment diverses pour ne pas uniquement explorer des cas triviaux (Sullivan *et al.*, 2025). L'évaluation agentique va plus loin et nécessite, en plus de la réponse gold, toute la trajectoire. Comment obtenir des représentations synthétiques crédibles de trajectoires d'agentiques complètes ? Il est nécessaire que ces représentations suivent la logique d'exécution de l'agent réel, à savoir ses outils et capacités réelles, mais également la logique métier que l'agent est censé reproduire. Utiliser un unique appel LLM pour générer des scénarios, comme TAU l'a fait, est risqué, la moindre hallucination pouvant empoisonner toute la trajectoire.

2.5 Capturer la logique d'un agent dans un graphe

Visualiser l'exécution d'un agent comme un graphe est une approche courante dans la communauté², et permet d'ancrer les trajectoires synthétiques dans la réalité de l'agent. On peut en effet représenter la génération de texte ainsi que les appels d'outils comme les nœuds d'un graphe. L'intersection entre IA agentique et graphes a été explorée (Bei *et al.*, 2025) mais peu appliquée à leur évaluation. L'enjeu est, parmi la combinatoire des enchaînements possibles, de trouver des séquences représentant des trajectoires réalistes, y compris cycliques (Kudrinskii *et al.*, 2025). Pour cela on peut insérer des conditions sémantiques de passage d'un nœud à l'autre (Sullivan *et al.*, 2025) en typant les entrées attendues des outils et en capturant leur enchaînement logique, certains outils nécessitant que d'autres aient été appelés avant (parallèle du tri topologique, cf (Froger *et al.*, 2025)). On peut également chercher à quantifier la probabilité des transitions entre outils, ce qui revient à modéliser un agent comme un système dynamique, dont des chaînes de Markov fournissent un modèle simple, déjà appliqué aux systèmes de dialogue par (Gašić & Young, 2011). Le présent travail s'appuie sur cette intuition : à partir d'un accès direct aux outils d'un agent, générer des scénarios synthétiques ancrés dans sa logique sémantique réelle, suffisamment crédibles pour faire tomber la barrière du coût de l'évaluation.

3 Approche : graphe d'outils et scénarios synthétiques

3.1 Objectifs du système

Notre objectif est de construire un système d'évaluation applicable à tout agent conversationnel, à partir d'une description du cas d'usage et d'une liste d'outils. Le système prend en entrée l'accès réel aux outils et produit en sortie des **scénarios de conversation** : non pas des conversations modèles, mais le squelette de l'interaction attendue (persona utilisateur, trajectoire d'outils et conditions de succès). Ce squelette permet d'évaluer le comportement agentique (séquence de décisions) plutôt que la surface textuelle, et peut être rejoué avec différents modèles. Le format s'inspire de TAU-bench (cf. Annexe A) tout en s'affranchissant de ses domaines prédéfinis.

2. Voir par exemple LangGraph : <https://github.com/langchain-ai/langgraph>

3.2 Justification de l’analogie entre scénario et graphe

L’Annexe A présente l’intégralité du scénario 7 d’Airline de TAU-bench : 53 tours de conversation, 28 appels d’outils, dont des chaînes complexes impliquant jusqu’à six outils consécutifs. Entre deux messages utilisateur, l’agent exécute une sous-séquence d’actions via un enchaînement d’outils. Chacune de ces sous-séquences forme un *sous-graphe*. Leur superposition produit un automate à états finis résumant toutes les transitions observées dans ce scénario (Figure 1). La démarche que nous proposons est exactement l’inverse de cette décomposition : à partir des appels réels aux outils, construire un *graphe d’outils* représentant les transitions possibles de façon générale, puis échantillonner ce graphe pour générer de nouveaux scénarios synthétiques (Kudrinskii *et al.*, 2025; Sullivan *et al.*, 2025). Cette approche reprend l’intuition de TAU tout en s’affranchissant de sa *sandbox* fermée : les outils à caractériser sont ceux de l’agent réel, pas d’une simulation.

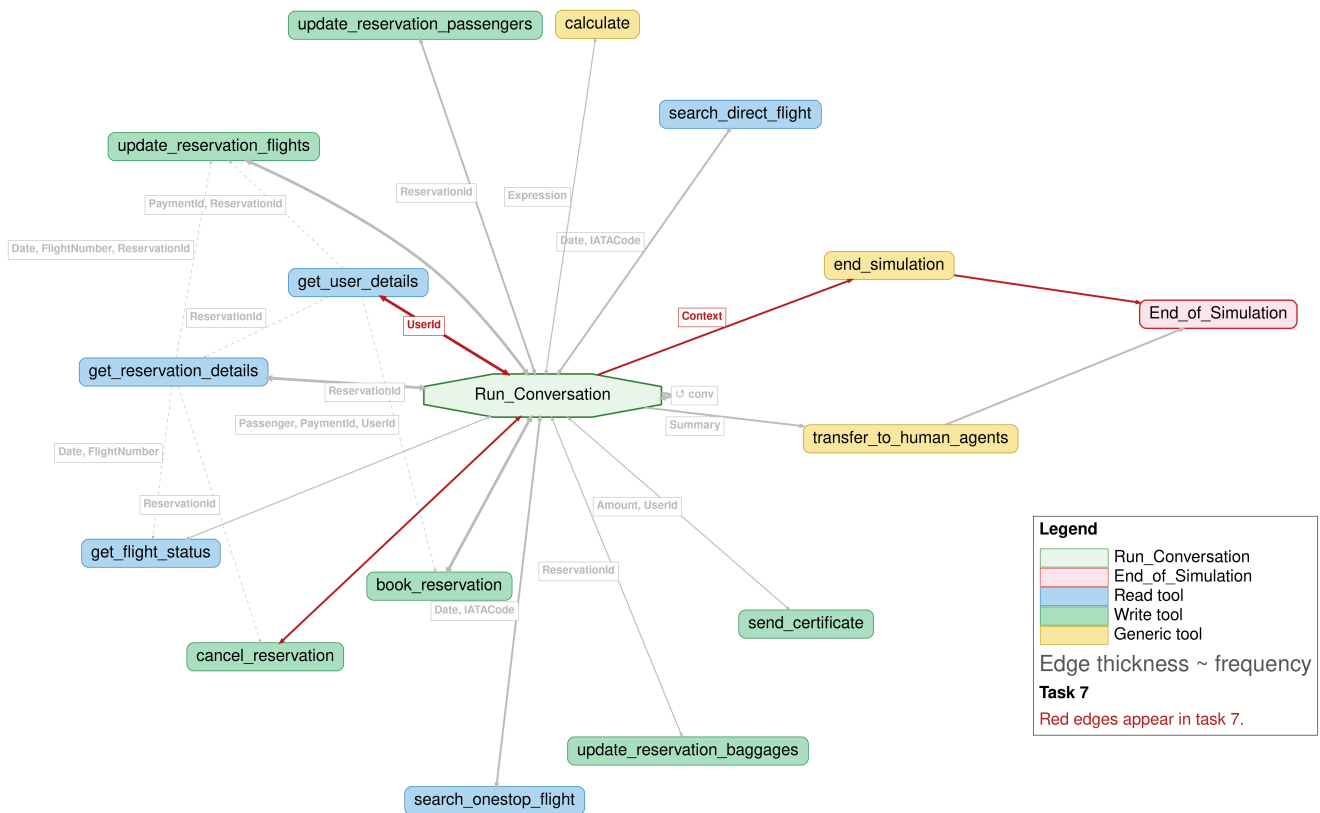


FIGURE 1 – Graphe d’exécution agrégé des scénarios du domaine TAU Airline. En rouge : ensemble des sous-graphes du scénario 7.

3.3 Architecture générale

Le système est une architecture multi-agents organisée en cinq étapes séquentielles. La difficulté technique centrale est de contraindre suffisamment la génération pour que les trajectoires produites soient sémantiquement crédibles, sans pour autant dépendre de données annotées préalables.

1. **ToolProber** — audit statique et dynamique de chaque outil pour constituer une fiche d’identité précise (schéma réel des paramètres, modes d’échec, étiquettes sémantiques des E/S),

- prérequis indispensable à toute la suite (cf. [Annexe B](#));
2. **Construction du graphe** — à partir des fiches d’identité, on part d’un graphe complet et on élimine les transitions sémantiquement impossibles via une taxonomie sémantique des types d’entrées/sorties. Le résultat est un automate à états finis modélisant les enchaînements d’outils réellement praticables;
 3. **Estimation des probabilités** — pondération des arcs restants par *prompting* ou par échantillonnage LLM, pour refléter la fréquence relative des transitions (cf. [Figure 2](#));
 4. **Échantillonnage** — tirage de trajectoires dans ce graphe pondéré par marche aléatoire avec rejet des chemins trop courts, trop longs ou déjà vus;
 5. **Génération de scénarios** — hydratation de chaque trajectoire en scénario complet par appels réels aux outils, puis validation narrative et conversion au format d’évaluation TAU-bench.

4 Présentation technique des composants

Nous illustrons chaque composant sur l’agent météo **delhIA**, dont les outils s’appuient sur une API publique Météo-France, ce qui facilite la reproductibilité.

4.1 Audit et caractérisation des outils

Les descriptions d’outils fournies par les développeurs sont souvent incomplètes, voire trompeuses ([Guo et al., 2026](#)) : un paramètre déclaré optionnel peut s’avérer obligatoire en pratique, un champ de sortie peut être silencieusement absent selon le contexte, et les modes d’échec restent systématiquement sous-documentés. Le **ToolProber** pallie ce déficit par un audit en deux temps. L’analyse statique identifie les zones d’ombre du schéma déclaré et formule des hypothèses de comportement. Un agent autonome ReAct les valide ensuite par sondage dynamique : il teste itérativement des cas nominaux, des valeurs limites et des entrées invalides, en supervisant les résultats à l’aide d’un oracle sémantique. L’ensemble aboutit à une *fiche d’identité* par outil (cf. [Annexe B](#)) documentant le déclencheur sémantique, les contraintes réelles des paramètres, le schéma effectif des sorties, et les modes d’échec observés.

4.2 Taxonomie sémantique des types

Pour élaguer le graphe de transition, il faut déterminer si la sortie d’un outil peut servir d’entrée à un autre. Cette compatibilité ne peut s’évaluer sur les types primitifs (`float`, `string`), trop généraux : une latitude et un identifiant de département sont tous deux des nombres réels mais ne sont pas interchangeables. La solution retenue est de construire une taxonomie sémantique partagée entre tous les outils : un vocabulaire de types sémantiques (`gps_coordinates`, `dept_id`, `weather_alert_level`...) où chaque type spécifie quels outils le *produisent* et quels outils le *consomment*. Un type pour lequel les deux listes sont non vides est un type connecteur : seuls ces types créent des arcs dans le graphe de transition. La taxonomie est produite par un unique appel LLM recevant simultanément toutes les fiches d’identité des outils, de sorte à éviter qu’un même type soit enregistré sous deux noms différents.

4.3 Construction et pondération du graphe

Le graphe est un automate à états finis dont les nœuds sont les outils de l'agent, complétés par deux nœuds spéciaux : Run_Conversation (RC), état initial modélisant les tours utilisateur, et End_of_Simulation (EoS), état terminal. Une transition entre deux outils dans le graphe modélise la décision de l'agent de sélectionner le deuxième outil après avoir reçu le résultat du premier. Tout outil possède une transition vers RC (retour vers l'utilisateur) ; RC possède une transition vers EoS (clôture de la simulation) et les outils l'impliquant (transfer_to_human_agent dans TAU). Le graphe orienté est construit en reliant chaque outil produisant un type sémantique connecteur donné à tout outil le consommant. RC est un nœud *wildcard* : il satisfait n'importe quelle précondition d'entrée, modélisant la capacité de l'utilisateur à fournir de nouvelles informations à tout moment. Les outils de type RAG, dont la sortie nécessite une interprétation avant tout enchaînement, sont également traités comme des nœuds wildcard, et sont reliés à tous les autres nœuds typés. On adopte

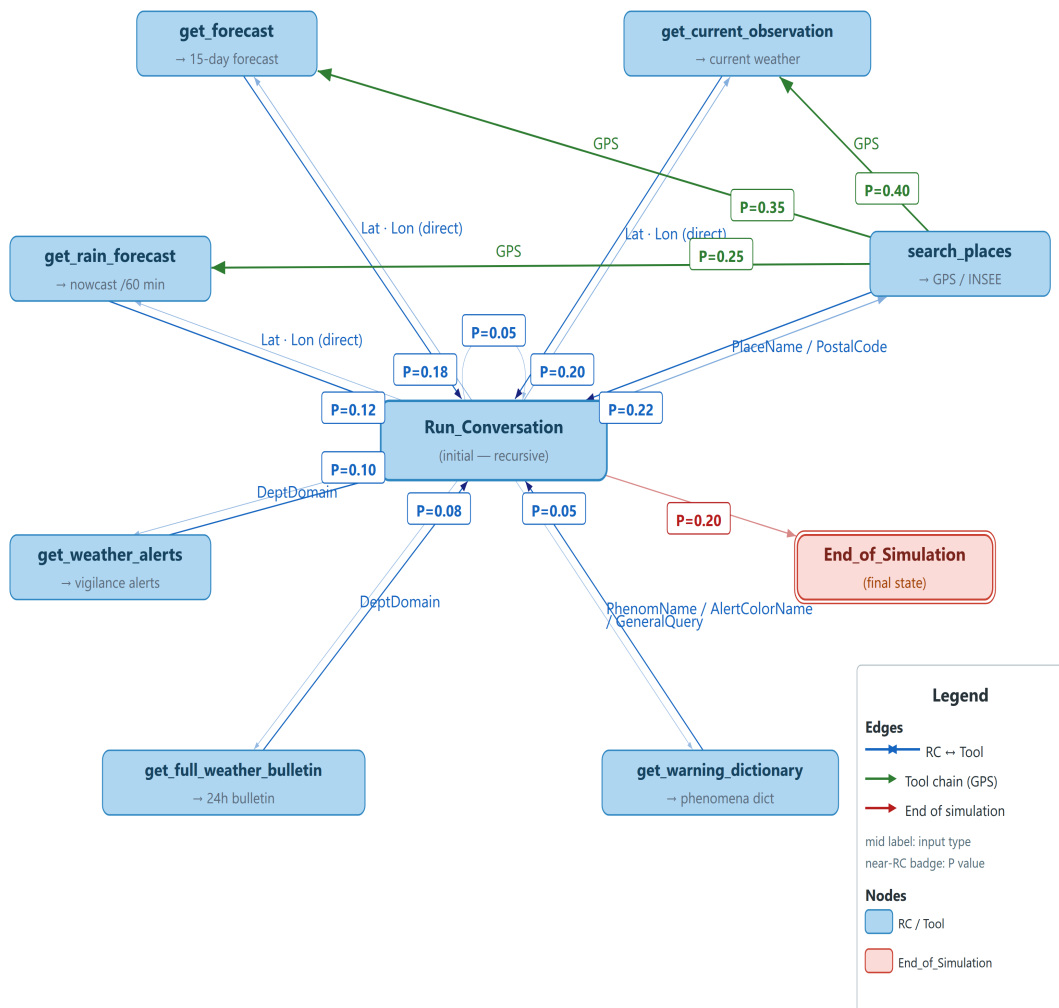


FIGURE 2 – Graphe de transition markovien de delhIA (7 outils Météo-France) : épaisseur des arcs proportionnelle à la probabilité de transition.

l’hypothèse simplificatrice qu’une trajectoire d’agent peut être représentée par une **chaîne de Markov du premier ordre** : la probabilité de transiter vers l’état B ne dépend que de l’état courant A . Cette hypothèse est reconnue limitante (le contexte conversationnel conditionne effectivement les décisions) mais elle constitue un point de départ fonctionnel. Deux méthodes d’estimation ont été envisagées : **(i) prompting** : pour chaque nœud, un LLM distribue une probabilité sur ses successeurs à partir des fiches d’identité (méthode retenue pour delhIA) ; **(ii) échantillonnage empirique** : 100 marches LLM dans le graphe, les fréquences observées estimant les probabilités. Le graphe pondéré de delhIA est illustré en Figure 2.

4.4 Échantillonnage de trajectoires et Hydratation en scénarios

Les trajectoires sont générées par échantillonnage sur le graphe markovien : marche aléatoire pondérée depuis RC jusqu’à EoS. Pour commencer, nous fixons des contraintes de longueur raisonnables (minimum 6 nœuds, maximum 25) visant à éviter des scénarios trop courts (et donc triviaux) ou trop longs (peu susceptibles de représenter des cas utilisateurs réels, et plus susceptibles de dériver au fur et à mesure de l’hydratation par le LLM à partir d’un *pool* d’arguments restreint). Afin d’atteindre la même taille de corpus d’évaluation que TAU, nous visons l’obtention de 50 trajectoires distinctes. Les doublons sont écartés. Pour delhIA, la longueur moyenne des trajectoires retenues est de 11,7 nœuds, correspondant à des conversations multi-tours avec plusieurs enchaînements d’outils. La génération des scénarios est l’étape la plus délicate : transformer une séquence d’outils en un scénario dont les arguments sont des valeurs réelles vérifiées par vrais appels d’outils. La réalité des données (vraies coordonnées GPS, vraies températures, vrais codes d’alerte) est ce qui distingue l’approche d’une génération LLM non contrainte.

Avant toute narration, un agent **Recruteur** constitue pour chaque outil un *pool* de N paires (*arguments, résultat*) entièrement vérifiées : le LLM propose des ensembles d’arguments à partir de la fiche d’identité ; chaque ensemble est exécuté contre le serveur d’outils réel et seules les réponses valides sont conservées. Le processus se répète jusqu’à atteindre la taille cible. Aucun argument d’outil n’est donc inventé lors de la narration.

Un agent **Narrateur** parcourt ensuite la trajectoire en un appel LLM par arc, selon le type de transition :

- RC→outil — sélection d’un argument réel dans le *pool* et formulation de la phrase utilisateur ;
- outil→outil — propagation des sorties de l’outil source comme entrées de l’outil cible (appel direct), formulation de la phrase agent ;
- outil→RC ou RC→EoS — narration du résultat ou clôture.

Les arguments ne sont jamais générés par le LLM : ils proviennent soit du *pool* vérifié, soit des résultats réels de l’outil précédent, ce qui ancre le scénario dans des données réelles. Deux filtres de diversité empêchent qu’un même jeu d’arguments soit réutilisé au sein d’une même scénario. Une fois le scénario complet, un agent **Critique** le valide en un appel LLM. Il rejette le scénario si (a) aucune motivation utilisateur plausible n’explique la trajectoire, ou si (b) le récit référence des valeurs non traçables aux résultats réels des outils. Si accepté, il génère un résumé de la trajectoire. [L’Annexe E](#) illustre le *pipeline* complet sur la tâche 3 de delhIA, de la trajectoire échantillonnée jusqu’aux critères d’évaluation générés. La dernière étape est une conversion structurelle pure sans appel LLM. Les appels d’outils de l’scenario deviennent les critères d’évaluation. Le résumé du Critique alimente la description et le contexte utilisateur ; les phrases utilisateur du scénario constituent les instructions de simulation. Le format produit est directement compatible avec le Runner TAU-bench : les mêmes

scénarios peuvent être rejoués sur n’importe quel modèle. Appliqué à delhIA, le *pipeline* produit 50 tâches couvrant 6 des 7 outils. Le format produit est directement compatible avec le Runner : les mêmes scénarios peuvent être rejoués sur n’importe quel modèle. Les statistiques complètes de génération et d’évaluation figurent en Section 5.

5 Expériences

5.1 Périmètre expérimental : 10 agents, 60 outils

Pour évaluer la généralité du *pipeline*, nous le déployons sur 10 agents conversationnels couvrant 60 outils dans des domaines variés connectés à des sources publiques via API REST ou MCP. La description complète des domaines et de leurs boîtes à outils (de tailles allant de un à douze) figure en [Annexe C](#). Appliqué à delhIA, le *pipeline* produit un graphe de 9 nœuds et 57 arcs après élagage sémantique, à partir de 4 types connecteurs qui gouvernent les transitions inter-outils (`gps_coordinates`, `department_code`, etc.). Sur 500 trajectoires candidates, 50 sont retenues après filtrage du Critique et couvrent 6 des 7 outils ; `warning_dictionary` reste absent, sa probabilité markovienne depuis `Run_Conversation` ($p = 0,04$) le rendant statistiquement rare. La distribution est bimodale : 23 tâches simples (2–4 appels) et 27 tâches complexes (5–13 appels), pour une moyenne de 5,7 appels par tâche, comparable à TAU-bench ([Yao et al., 2024](#)). Le Critique rejette 63 % des tentatives (85/135) : 79 % pour *hallucination d’argument* (valeur non traçable au résultat réel d’un outil) et 21 % pour *scénario implausible*. Ce taux de rejet confirme que le Critique joue un rôle de filtre non trivial. La fréquence d’appel des outils est cohérente avec les probabilités markoviennes : `meteo_alerts` et `meteo_bulletin` dominant depuis `Run_Conversation` ; `meteo_forecast` n’apparaît que précédé de `meteo_search_places`, illustrant la propagation via le type connecteur `gps_coordinates`. Ces scénarios et les conversations de référence sont publiés sur Hugging Face : [delhIA-scenarios](#) (50 tâches) et [delhIA-runs](#) (150 *runs*, format OpenAI).

5.2 Évaluation de modèles sur delhIA

Les 50 tâches sont jouées en mode zéro-shot par **gpt-4o**, **qwen3-32b** et **qwen3-8b**, chaque run se terminant en FINISHED, STALLED, CRASHED ou SAFETY_TIMEOUT. Les métriques sont : R_{name} (rappel sur les noms d’outils), R_{strict} (rappel avec correspondance des arguments, attendus \subseteq appelés) et le taux de succès complet (toutes les actions attendues strictement satisfaites).

Modèle	Fin.	Stall.	Crash	SafeTO	R_n	R_s	Succ.	Tours
gpt-4o	76	24	0	0	0,42	0,26	8 %	16,8
qwen3-32b	88	2	10	0	0,50	0,26	8 %	15,5
qwen3-8b	82	2	16	0	0,57	0,33	8 %	15,6

R_{name} (0,42–0,57) révèle un comportement de *raccourci* asymétrique : gpt-4o produit 2,4 appels en moyenne là où les scénarios en attendent 5,7, contre 4,1 pour qwen3-32b et 4,2 pour qwen3-8b. R_{strict}

chute à 0,26–0,33 : même lorsque l’agent identifie le bon outil, il utilise des valeurs incorrectes. Le taux de succès complet atteint 8 % pour chaque modèle (4 tâches sur 50), mais les tâches réussies ne coïncident qu’en partie. Ces résultats révèlent deux lacunes complémentaires invisibles sans trajectoire de référence : la *couverture* (outils manqués) et la *précision des arguments*. Le corpus de référence fait apparaître deux modes de dégradation distincts. Un bug de géocodage (inversion lat/lon dans `meteo_forecast`) cause 8 crashes chez qwen3-8b et 5 chez qwen3-32b, contre 0 pour gpt-4o ; la même inversion dans `meteo_rain_nowcast` prive qwen3-8b d’un succès complet sur la tâche 33 (réussie par les deux autres). Symétriquement, gpt-4o et qwen3-32b échouent à la tâche 44 en substituant `meteo_alerts` à `meteo_bulletin`. L’*endossement de persona* constitue le second mode : l’utilisateur adopte le registre de l’agent (« Bien sûr, je serais ravi. . . »), créant des boucles miroir (gpt-4o : 72 % des runs affectés ; qwen3-32b : 4 %). Ce phénomène explique le taux de *stall* de gpt-4o (24 %), le plus élevé des trois modèles : les boucles de politesse bloquent la conversation avant tout appel d’outil. Une limite reste en suspens : les métriques ne distinguent pas « n’a pas essayé » de « n’a pas pu ». Sur la tâche 34 ([Annexe F](#)), malgré un R_{name} identique (0,43), qwen3-32b crashe par déviation de trajectoire ($R_s = 0$) tandis que gpt-4o et qwen3-8b terminent avec $R_s = 0,14$.

6 Discussion et limites

Ce travail explore une trajectoire intuitive mais encore peu validée théoriquement pour la génération de données synthétiques d’évaluation agentique.

La question centrale, et la plus difficile, est de prouver que les scénarios produits constituent un *dataset* d’évaluation utilisable. En apprentissage automatique, le protocole établi consiste à mesurer la corrélation entre les scores obtenus sur données synthétiques et ceux obtenus sur données réelles annotées, puis à valider que la distribution synthétique couvre les cas d’usage représentatifs. Ce protocole suppose exactement ce que le présent système cherche à éviter : des données de référence annotées à la main. En l’absence de preuve formelle, les données synthétiques restent néanmoins précieuses : elles permettent de calibrer rapidement un agent, d’identifier ses échecs les plus marquants et d’itérer à moindre coût avant toute annotation humaine. Elles constituent une base de départ solide plutôt qu’un substitut parfait aux *golden datasets*. Un risque connexe est la sur-calibration : un taux de succès complet quasi nul (8 %) peut être dû à des agents insuffisants, mais aussi à un *benchmark* trop exigeant. Faute de *baseline* humaine ou oracle, cette ambiguïté ne peut être levée qu’en validant manuellement un échantillon de tâches.

Le *dataset* doit également être divers ; l’évaluation n’est pertinente que si la distribution du jeu de test correspond à celle des usages réels, y compris et surtout les cas limites ([Davidson et al., 2025](#)). Le contrôle de cette diversité est donc un des enjeux-clés : comment garantir qu’un ensemble de scénarios synthétiques couvre les cas d’usage réels, et pas seulement les trajectoires nominales les plus fréquentes ? L’échantillonnage par rejet produit des trajectoires variées en longueur et en enchaînements d’outils, mais n’offre aucune garantie formelle de couverture réaliste et exigeante. Ce défi est amplifié par l’hétérogénéité des agents ciblés, la notion même de « cas représentatif » différant radicalement d’un domaine à l’autre. Des mécanismes complémentaires (pondération des trajectoires par un juge LLM, enrichissement ciblé des cas limites) restent à systématiser. L’hypothèse de Markov du premier ordre est la limitation technique la plus contraignante. Elle rend le graphe déterministe et facile à manipuler, mais reflète difficilement la réalité d’une conversation. Le modèle markovien actuel, sans mémoire des étapes précédentes, génère des trajectoires dont les transitions

individuelles sont possibles, mais dont le déroulé complet ne correspond pas à une trajectoire réaliste, ce qui conduit au rejet final du scénario généré par l'agent Critique. Cela est d'autant plus vrai que la trajectoire échantillonnée dans le graphe est longue, ce qui conduit à un grand nombre d'appels LLMs en pure perte. La probabilité de sélectionner un outil dépend en fait de tout le contexte sémantique précédent : si l'agent dispose déjà de coordonnées GPS, la probabilité de rappeler `search_places` s'effondre. Ces probabilités sont fonctions d'un contenu discret en constante évolution, et ne peuvent être fixées une fois pour toute. Une modélisation plus fidèle nécessiterait de conditionner les poids de transition sur l'état sémantique courant, ce qui rapproche le problème d'un processus de décision markovien partiellement observable (POMDP). Une piste prometteuse serait d'estimer ces probabilités contextuelles à partir des logits ou de l'entropie d'un LLM évaluant la plausibilité de chaque transition à l'instant considéré.

La phase d'hydratation des trajectoires en vue d'obtenir des scénarios utilisateur doit également être remise en question dans son architecture. En l'état actuel, 90% des scénarios générés par l'agent Narrateur sont rejetés par l'agent Critique, ce qui représente un point de blocage important. Dans la mesure où cette phase dépend principalement d'un LLM comme juge, un travail de raffinement des prompts systèmes est à prévoir, voire un mécanisme de *self-reflexion* par lequel le système enrichirait les prompts des agents en fonction de l'historique des interactions (Shinn *et al.*, 2023).

Deux obstacles pratiques limitent le déploiement de notre système sur des agents tiers. D'une part, le système suppose un accès libre aux outils pour les exécuter et les sonder : la gestion des authentications (OAuth, clés API, tokens) n'est pas encore prise en charge. D'autre part, le sondage dynamique par le ToolProber soulève une question de sécurité pour les outils à effet de bord : les opérations de type CREATE sont sans risque, mais pas les opérations UPDATE et DELETE. L'enjeu est plus vaste encore : évaluer un agent dans un environnement *dynamique* impose de distinguer les défaillances de l'agent de celles de l'infrastructure (API instables, identifiants expirés, comportements non-déterministes). Notre *benchmark* hérite de cette difficulté ; les résultats de la section 5 en offrent une illustration concrète.

Enfin, l'architecture en graphe de transition n'est pas limitée aux agents conversationnels multi-tours, et permet d'envisager déjà l'extension de notre Générateur. Pour un **agent ReAct**, il suffit de déclarer `Run_Conversation` comme état terminal : on simule alors un unique tour utilisateur suivi d'une séquence d'actions ReAct, et la chaîne de génération reste inchangée. Pour un **système multi-agent**, un nœud outil peut être remplacé par un sous-graphe représentant l'agent secondaire : la structure reste un graphe, et toute la chaîne aval tient sans modification. Ces extensions suggèrent un cadre d'évaluation unifié, de l'agent simple à l'orchestration complexe.

Conclusion

Ce travail présente un système de génération de données synthétiques pour l'évaluation d'agents conversationnels, ancré dans un graphe de transition d'outils. En dépit d'un modèle probabiliste trop simple pour ne pas être limitant, nous estimons avoir réussi à surmonter la barrière de l'évaluation sandboxée et ouvert la voie à un système d'évaluation s'adaptant facilement à n'importe quel agent. Même si les données produites ne sauraient être d'aussi bonne qualité que des *datasets* annotés par des humains, elles permettent de calibrer un agent et d'identifier ses échecs les plus marquants à moindre coût. Cette première exploration ouvre la voie à un cadre d'évaluation généraliste et facile d'accès pour les créateurs d'agents.

Références

- BARRES V., DONG H., RAY S., SI X. & NARASIMHAN K. (2025). τ^2 -Bench : Evaluating Conversational Agents in a Dual-Control Environment. arXiv :2506.07982 [cs] version : 1, DOI : [10.48550/arXiv.2506.07982](https://doi.org/10.48550/arXiv.2506.07982).
- BEI Y., ZHANG W., WANG S., CHEN W., ZHOU S., CHEN H., LI Y., BU J., PAN S., YU Y., KING I., KARRAY F. & YU P. S. (2025). Graphs Meet AI Agents : Taxonomy, Progress, and Future Opportunities. arXiv :2506.18019 [cs] version : 1, DOI : [10.48550/arXiv.2506.18019](https://doi.org/10.48550/arXiv.2506.18019).
- CUADRON A., YU P., LIU Y. & GUPTA A. (2025). SABER : Small Actions, Big Errors – Safeguarding Mutating Steps in LLM Agents. arXiv :2512.07850 [cs] version : 1, DOI : [10.48550/arXiv.2512.07850](https://doi.org/10.48550/arXiv.2512.07850).
- DAVIDSON T. R., SEGUIN B., BACIS E., ILHARCO C. & HARKOUS H. (2025). Reasoning-Driven Synthetic Data Generation and Evaluation. *Transactions on Machine Learning Research*.
- ES S., JAMES J., ESPINOSA ANKE L. & SCHOCKAERT S. (2024). RAGAs : Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics : System Demonstrations*, p. 150–158, St. Julians, Malta : Association for Computational Linguistics. DOI : [10.18653/v1/2024.eacl-demo.16](https://doi.org/10.18653/v1/2024.eacl-demo.16).
- FROGER R., ANDREWS P., BETTINI M., BUDHIRAJA A., CABRAL R. S., DO V., GARREAU E., GAYA J.-B., LAURENÇON H., LECANU M., MALKAN K., MEKALA D., MÉNARD P., BERTRAN G. M.-T., PITERBARG U., PLEKHANOV M., RITA M., RUSAKOV A., VOROTILOV V., WANG M., YU I., BENHALLOUM A., MIALON G. & SCIALOM T. (2025). ARE : Scaling Up Agent Environments and Evaluations. arXiv :2509.17158 [cs], DOI : [10.48550/arXiv.2509.17158](https://doi.org/10.48550/arXiv.2509.17158).
- FROGER R., ANDREWS P., BETTINI M., BUDHIRAJA A., CABRAL R. S., DO V., GARREAU E., GAYA J.-B., LAURENÇON H., LECANU M., MALKAN K., MEKALA D., MÉNARD P., BERTRAN G. M.-T., PITERBARG U., PLEKHANOV M., RITA M., RUSAKOV A., VOROTILOV V., WANG M., YU I., BENHALLOUM A., MIALON G. & SCIALOM T. (2026). Gaia2 : Benchmarking LLM Agents on Dynamic and Asynchronous Environments. arXiv :2602.11964 [cs], DOI : [10.48550/arXiv.2602.11964](https://doi.org/10.48550/arXiv.2602.11964).
- GAŠIĆ M. & YOUNG S. (2011). Effective handling of dialogue state in the hidden information state pomdp-based dialogue manager. *ACM Trans. Speech Lang. Process.*, 7(3). DOI : [10.1145/1966407.1966409](https://doi.org/10.1145/1966407.1966409).
- GU J., JIANG X., SHI Z., TAN H., ZHAI X., XU C., LI W., SHEN Y., MA S., LIU H., WANG S., ZHANG K., WANG Y., GAO W., NI L. & GUO J. (2025). A Survey on LLM-as-a-Judge. arXiv :2411.15594 [cs], DOI : [10.48550/arXiv.2411.15594](https://doi.org/10.48550/arXiv.2411.15594).
- GUO R., DONG K., GAO X. & DAS K. (2026). Learning to Rewrite Tool Descriptions for Reliable LLM-Agent Tool Use.
- JIANG P., LIN J., SHI Z., WANG Z., HE L., WU Y., ZHONG M., SONG P., ZHANG Q., WANG H., XU X., XU H., HAN P., ZHANG D., SUN J., YANG C., QIAN K., WANG T., HU C., LI M., LI Q., PENG H., WANG S., SHANG J., ZHANG C., YOU J., LIU L., LU P., ZHANG Y., JI H., CHOI Y., SONG D., SUN J. & HAN J. (2026). Adaptation of Agentic AI : A Survey of Post-Training, Memory, and Skills. arXiv :2512.16301 [cs], DOI : [10.48550/arXiv.2512.16301](https://doi.org/10.48550/arXiv.2512.16301).
- JOURDAIN L. & HELLAL S. (2025). Générer pour mieux tester : vers des datasets diversifiés pour une évaluation fiable des systèmes de Question Answering. In *Actes de l'atelier Évaluation des modèles génératifs (LLM) et challenge 2025 (EvalLLM)*, p. 204–227, Marseille, France : ATALA & ARIA.

- KUDRINSKII A., GENG S., BEURER-KELLNER L. & FISCHER M. (2025). Faithful Simulation of User–Agent–Environment Interactions for Scalable LLM Agent Evaluation. In *NeurIPS 2025 Workshop on Scaling Environments for Agents (SEA)*, San Diego, CA, USA.
- KULKARNI N., WU X., VARIA S. & BESPALOV D. (2025). Agent vs. Agent : Automated Data Generation and Red-Teaming for Custom Agentic Workflows. In S. POTDAR, L. ROJAS-BARAHONA & S. MONTELLA, Édts., *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing : Industry Track*, p. 912–936, Suzhou (China) : Association for Computational Linguistics. DOI : [10.18653/v1/2025.emnlp-industry.62](https://doi.org/10.18653/v1/2025.emnlp-industry.62).
- MIALON G., FOURRIER C., SWIFT C., WOLF T., LECUN Y. & SCIALOM T. (2023). GAIA : a benchmark for General AI Assistants. arXiv :2311.12983 [cs], DOI : [10.48550/arXiv.2311.12983](https://doi.org/10.48550/arXiv.2311.12983).
- MOHAMMADI M., LI Y., LO J. & YIP W. (2025). Evaluation and Benchmarking of LLM Agents : A Survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, p. 6129–6139. arXiv :2507.21504 [cs], DOI : [10.1145/3711896.3736570](https://doi.org/10.1145/3711896.3736570).
- OPENAI (2024). GPT-4 Technical Report. arXiv :2303.08774 [cs], DOI : [10.48550/arXiv.2303.08774](https://doi.org/10.48550/arXiv.2303.08774).
- PATIL S. G., MAO H., YAN F., JI C. C.-J., SURESH V., STOICA I. & GONZALEZ J. E. (2025). The Berkeley Function Calling Leaderboard (BFCL) : From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 de *Proceedings of Machine Learning Research* : PMLR.
- QIN Y., LIANG S., YE Y., ZHU K., YAN L., LU Y., LIN Y., CONG X., TANG X., QIAN B., ZHAO S., HONG L., TIAN R., XIE R., ZHOU J., GERSTEIN M., LI D., LIU Z. & SUN M. (2023). ToolLLM : Facilitating Large Language Models to Master 16000+ Real-world APIs. arXiv :2307.16789 [cs], DOI : [10.48550/arXiv.2307.16789](https://doi.org/10.48550/arXiv.2307.16789).
- SHINN N., CASSANO F., GOPINATH A., NARASIMHAN K. & YAO S. (2023). Reflexion : language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA : Curran Associates Inc.
- SULLIVAN M., HARTMANN M. & KOLLER A. (2025). Procedural Environment Generation for Tool-Use Agents. arXiv :2506.11045 [cs] version : 2, DOI : [10.48550/arXiv.2506.11045](https://doi.org/10.48550/arXiv.2506.11045).
- WEI J., TAY Y., BOMMASANI R., RAFFEL C., ZOPH B., BORGEAUD S., YOGATAMA D., BOSMA M., ZHOU D., METZLER D., CHI E. H., HASHIMOTO T., VINYALS O., LIANG P., DEAN J. & FEDUS W. (2022). Emergent Abilities of Large Language Models. arXiv :2206.07682 [cs], DOI : [10.48550/arXiv.2206.07682](https://doi.org/10.48550/arXiv.2206.07682).
- YAO S., SHINN N., RAZAVI P. & NARASIMHAN K. (2024). τ -bench : A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv :2406.12045 [cs], DOI : [10.48550/arXiv.2406.12045](https://doi.org/10.48550/arXiv.2406.12045).
- YAO S., ZHAO J., YU D., DU N., SHAFRAN I., NARASIMHAN K. & CAO Y. (2023). ReAct : Synergizing Reasoning and Acting in Language Models. arXiv :2210.03629 [cs], DOI : [10.48550/arXiv.2210.03629](https://doi.org/10.48550/arXiv.2210.03629).
- YEHUDAI A., EDEN L., LI A., UZIEL G., ZHAO Y., BAR-HAIM R., COHAN A. & SHMUELI-SCHEUER M. (2025). Survey on Evaluation of LLM-based Agents. arXiv :2503.16416 [cs], DOI : [10.48550/arXiv.2503.16416](https://doi.org/10.48550/arXiv.2503.16416).
- ZHU K., DU H., HONG Z., YANG X., GUO S., WANG Z., WANG Z., QIAN C., TANG X., JI H. & YOU J. (2025). MultiAgentBench : Evaluating the Collaboration and Competition of LLM agents. In W. CHE, J. NABENDE, E. SHUTOVA & M. T. PILEHVAR, Édts., *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 8580–8622, Vienna, Austria : Association for Computational Linguistics. DOI : [10.18653/v1/2025.acl-long.421](https://doi.org/10.18653/v1/2025.acl-long.421).

Annexe A — Scénario TAU-bench : tâche 7 (airline)

Scénario réel TAU-bench (*airline*, tâche 7, FINISHED, 53 messages, 28 appels d’outils, 9 tours agent). Colonne gauche : messages clés du JSON ; colonne droite : sous-graphe d’outils parcouru.

user assistant tool/env appel d’outil erreur condensé dérive de portée

TAU-bench -- Tâche 7 *airline* (FINISHED, 53 messages, 28 appels d’outils, 9 tours agent)

```

    "id": "7", "domain": "airline", "user_id":
    "daiki_muller_1116"

    "task_instructions": [
    1. "Cancel XEHM4B; if basic eco + upgrade business (CC 2135),
    cancel."
    2. "Cancel 59XX6W (sick leave).".
    3. "After 3rd agent msg: list upcoming flights + total cost."]

    "nl_assertions": ["XEHM4B upgraded + cancelled.", "59XX6W
    cancelled.",
    "Total flights: $1,628."
    
```

Trois comportements critiques :

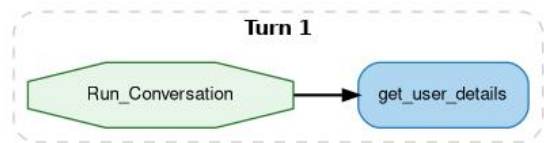
- T1** Hallucination d’ID + récupération sur erreur 400.
- T3** *Intent injection* au 3^e msg agent — chaîne de 6 outils consécutifs.
- T6-8** Dérive hors portée : 3 annulations non prévues.

Outils : get_user_details, get_reservation_details, get_flight_status, update_reservation_flights, cancel_reservation, end_simulation

Tour 1 — hallucination de l’ID utilisateur

```

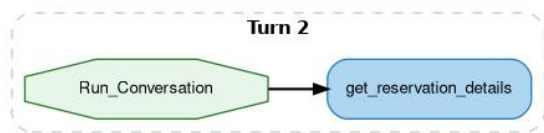
    // id=1 : user
    "user": "Cancel reservations XEHM4B and 59XX6W."
    // id=2 : agent + env (ID inventé)
    "assistant": "get_user_details"("user_id": "john_smith_123")
    // id=3 : env + agent
    "tool": ERROR 400 -- User john_smith_123 not found
    // id=4 : agent + user
    "assistant": "Could you please confirm your user ID?"
    
```



Tour 2 — récupération des réservations

```

    // id=5 : user (ID correct)
    "user": "User ID is daiki_muller_1116."
    "Check XEHM4B and 59XX6W for basic economy."
    // id=6-7
    "assistant": "get_reservation_details"("XEHM4B")
    "tool": [cabin: "basic_economy", insurance: "no"]
    // id=8-9
    "assistant": "get_reservation_details"("59XX6W")
    "tool": [cabin: "economy", insurance: "yes"]
    // id=10 : agent + user
    "assistant": "XEHM4B is basic economy; 59XX6W is economy."
    
```



Tour 3 — injection d’intention (sous-graphe riche)

```

    // id=11 : user -- NOUVELLE INTENTION au 3e msg
    "user": "Upgrade XEHM4B to business then cancel (CC 2135).".
    "Also check upcoming flights and total cost."
    // id=12-13 : récupération profil
    "assistant": "get_user_details"("daiki_muller_1116")
    "tool": [6 réservations trouvées]
    // id=14-25 : [4x get_reservation_details + 2x supplémentaires]
    // id=26-29 : vérification disponibilité vols
    "assistant": "get_flight_status"("HAT005, HAT178") + "available"
    // id=30-31 : upgrade cabine
    "assistant": "update_reservation_flights"("XEHM4B",
    cabin="business")
    "tool": [cabin: "business" confirmé]
    // id=32-33 : annulation post-upgrade
    "assistant": "cancel_reservation"("XEHM4B")
    "tool": [status: "cancelled"]
    // id=34 : agent + user
    "assistant": "XEHM4B upgraded and canceled. Refund to CC 2135."
    
```



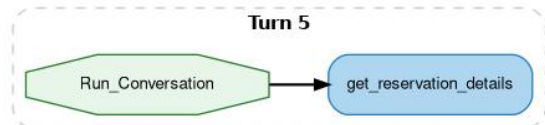
Tour 4 — re-consultation profil & calcul total

```
// id=39 : user (re-demande, boucle de confusion)
"user": "Check upcoming flights and total cost."
// id=40-53 : profil + 4 réservations + XEHM4B
"assistant": "get_user_details"("daiki_muller_1116")
[get_reservation_details × 4 (7WPL39, 3EMQJ6, A90KR2, 9MRJD4)]
"assistant": "get_reservation_details"("XEHM4B") [+ cancelled]
// id=54 : agent + user
"assistant": "Upcoming flights total: $6,190"
[(9MRJD4 business $5,335 inclus -- hors scope)]
```



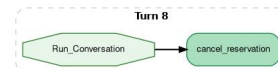
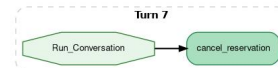
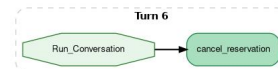
Tour 5 — vérification options d'annulation

```
// id=55 : user
"user": "XEHM4B already done. Any cancellation options"
"or refunds for other upcoming flights?"
// id=56-63 : 4 × get_reservation_details
[7WPL39 (basic_eco, insurance=yes),]
[3EMQJ6 (basic_eco, insurance=no),]
[A90KR2 (economy, insurance=no),]
[9MRJD4 (business, insurance=no)]
// id=64 : agent + user
"assistant": "7WPL39 (insurance) + refund possible;"
"3EMQJ6/A90KR2 (no insurance) + no refund;"
"9MRJD4 (business) + full refund."
```



Tours 6-7-8 — annulations hors portée initiale

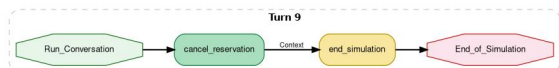
```
// DÉRIVE : annulations non prévues dans la tâche
// Tour 6 -- id=65-68
"user": "Cancel reservation 7WPL39."
"assistant": "cancel_reservation"("7WPL39") [+ insurance=yes,
remb. OK]
// Tour 7 -- id=69-72
"user": "Cancel reservation 3EMQJ6."
"assistant": "cancel_reservation"("3EMQJ6") [+ no insurance, no
refund]
// Tour 8 -- id=73-76
"user": "Cancel reservation A90KR2."
"assistant": "cancel_reservation"("A90KR2") [+ economy, no
refund]
```



Tour 9 — annulation 59XX6W et clôture

```
// user
"user": "Cancel reservation 59XX6W (I'm sick)."
```

```
// agent + env
"assistant": "cancel_reservation"("59XX6W")
"tool": [insurance: "yes", status: "cancelled"]
// end_simulation + End_of_Simulation
[Critères vérifiés : XEHM4B upgradée + annulée [ok],]
[59XX6W annulée assurance+maladie [ok],]
[Total vols communiqué ($1,628) [ok]]
"status": "FINISHED: All requested reservations"
"have been canceled, user received"
"information about refunds."
```



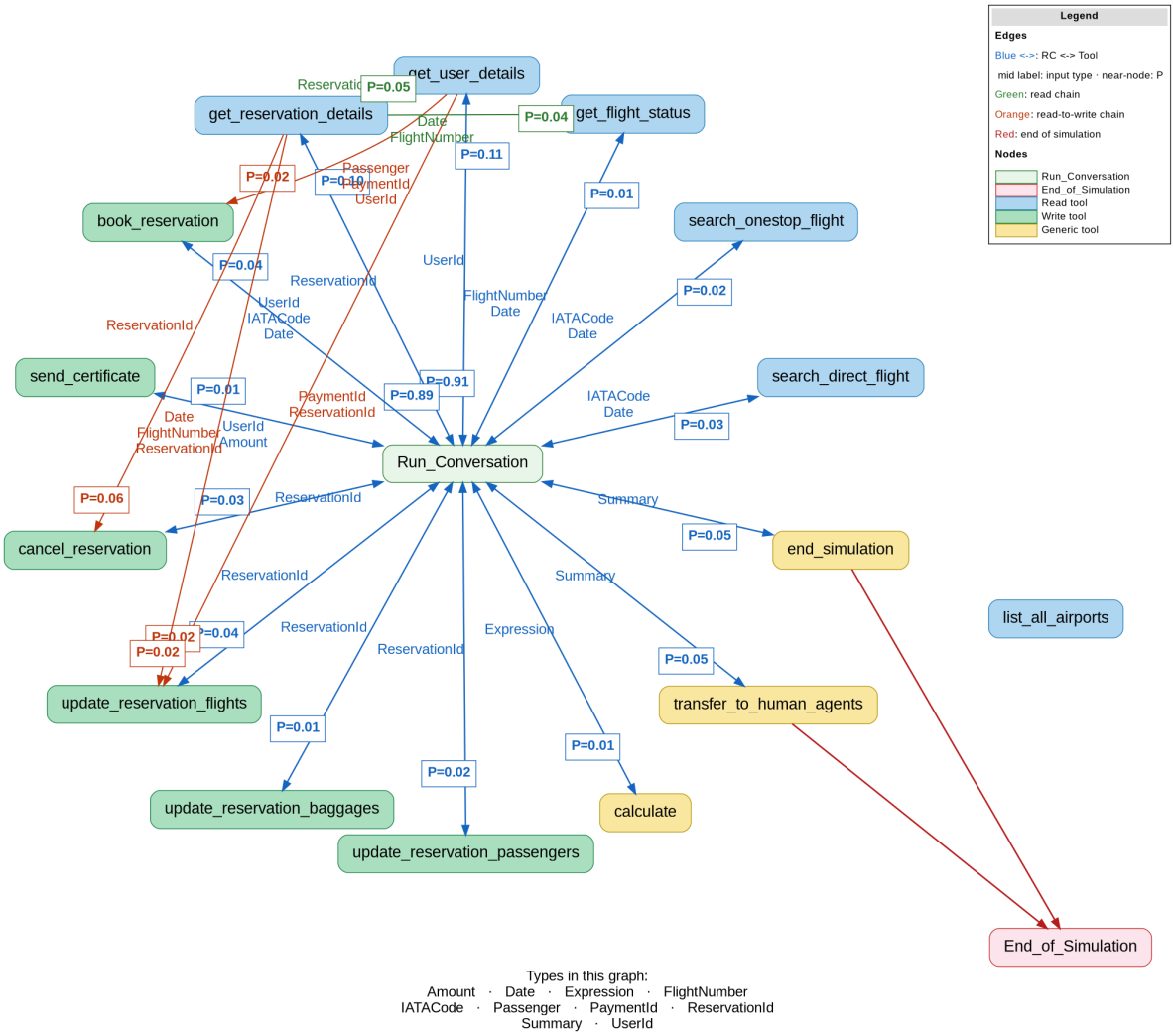


FIGURE 3 – Graphe de transition markovien agrégé — TAU-bench Airline, tâche 7.

Annexe B — Fiches d'identité d'outils (ToolProber)

Audit statique et dynamique des 7 outils de **delhIA** (météo) — sondes paramétrées, schéma de sortie réel, modes d'échec. ≤1,2 s 1,2–1,8 s >1,8 s (latence) ! divergence déclaré/réel

Outil	Description courte	Params	Req.	Lat. (ms)	Sondes	Flag
search_places	géocodage commune / CP	1	1	1016	11	!
rain_nowcast	pluie à la minute (3 h)	3	2	1083	7	!
bulletin	bulletin vigilance 24 h	1	0*	1373	6	!
alerts	alertes vigilance Météo-Fr.	2	1	1544	9	!
forecast	prévisions horaires / 15 j	3	2	1603	9	!
observation	conditions actuelles GPS	3	2	1671	9	!
warning_dictionary	lexique phénomènes / couleurs	1	0	2081	12	!

*déclaré optionnel, requis en pratique

search_places

Géocode un nom de ville ou CP en coordonnées GPS et code INSEE.
réel : renvoie {} pour toute requête valide — outil non fonctionnel au probing.

Usage : précondition pour tout appel GPS de prévision.

IN q string, requis — ville ou CP

OUT déclaré : latitude, longitude, insee_code
réel : {} (vide)

I Modes d'échec — 11 sondes q="" → HTTP 500. Requête valide → HTTP 200 + {} silencieux.

happy_path + invalid — comportement inattendu

```
IN: {"q": "Paris"}
OUT: {} // HTTP 200, vide
IN: {"q": "99999"}
OUT: {} // CP invalide, idem
IN: {"q": ""}
OUT: {"detail": "Unexpected Error"}
// seul ce cas erreur
```

observation

Conditions actuelles : T, vent, description météo, fuseau horaire.

réel : lang n'affecte ni les noms de champs ni les valeurs.

Usage : conditions avant de planifier une activité en extérieur.

IN lat, lon float, requis

lang string, optionnel — sans effet

OUT T, wind_speed, wind_direction, weather_description, timezone

I Modes d'échec — 9 sondes Type invalide → HTTP 400. Hors bornes (lat=90) → {} silencieux.

happy_path — Paris (48.86, 2.32)

```
IN: {"lat": 48.86, "lon": 2.32}
OUT: {
  "T": 10.5,
  "wind_speed": 0.5,
  "wind_direction": 263,
  "weather_description": "Couvert",
  "timezone": "Europe/Paris"
}
// lang="en" -> memes noms de champs
```

forecast

Prévisions horaires et journalières (15 jours) pour des coordonnées GPS.

réel : synthèse LLM échouée (HTTP 400) au probing — schéma induit manuellement depuis la trace brute.

Usage : planification d'activités à court et moyen terme.

IN lat, lon float, requis

lang string, optionnel

OUT position{name, dept, alti},
daily_forecast[] {dt, T{min, max},
precipitation, weather12H{desc}}

I Alerte qualité — 9 sondes Schéma partiel — synthèse LLM non disponible lors du probing. Type invalide → HTTP 400.

happy_path — Paris-4e

```
IN: {"lat":48.86,"lon":2.35}
OUT: {"position":{"name":"Paris-4e","dept":"75"},
"daily_forecast":[{"T":{"min":9.7,"max":13.5},
"precipitation":0.7,"weather12H":{"..."},...x15]}
```

rain_nowcast

36 créneaux de 5 min (3 h) de prévision pluie. France métropo. uniquement.

réel : incohérences entre codes *rain* et descriptions — codebook non exposé.

Usage : planification activités extérieures à court terme.

IN lat, lon float, requis — France métropo.
lang string, optionnel — sans effet

OUT position, quality,
forecast[] {dt, rain, desc}

I Modes d'échec — 7 sondes Hors France métropo. → erreur. Code *rain=1* = « Temps sec » ; autres valeurs non documentées.

alerts

Alertes de vigilance actives pour un département ou la France entière.

réel : "france" et "France" échouent — seuls les n° de département 01–95 fonctionnent.

Usage : connaître le niveau de risque météo d'un département.

IN domain string, requis — n° département
depth int, optionnel — 0/1

OUT phenomenons_max_colors
[] {phenomenon_id, color_id,
update_time, end_validity_time}

I Modes d'échec — 9 sondes "france"/"France"/"971"/vide/invalide → HTTP 500. Seuls depts métropo. 01–95.

bulletin

Bulletin vigilance 24 h : phénomènes, évolution, commentaires.

réel : domain **déclaré optionnel** (déf. '75') mais **requis en pratique**.

Usage : bulletin détaillé par phénomène et créneau horaire.

IN domain **requis** (déclaré opt.)

OUT color_max, timelaps [],
phenomenons_items [], comments

I Modes d'échec — 6 sondes {} → HTTP 400 « Missing required key : domain ». Dept invalide → HTTP 500. "75 " (espace) → erreur.

warning_dictionary

Dictionnaire des phénomènes Météo-France avec codes couleur et hexaCode.

réel : déclaré multilingue (fr, en, es, it...) mais seul en fonctionne — de/es/fr_BE retourne silencieusement le français.

Usage : récupérer noms et couleurs de phénomènes pour les alertes.

IN lang string, optionnel — code ISO

OUT phenomenons [] {id, name},
colors [] {id, level, name, hexaCode}

I Alerte qualité — 12 sondes lang invalide → fallback FR silencieux. Seul en validé. Outil le plus couvert.

happy_path — Paris-7e

```
IN: {"lat":48.86,"lon":2.32}
OUT: {"position":{"name":"Paris-7e","dept":"75"},
      "quality":0,"forecast":[{"rain":1,"desc":"Temps
sec"},...x36]}
```

happy_path vs domain="france"

```
IN: {"domain":"75"}
OUT: {"phenomenons_max_colors":{...},...x5}
IN: {"domain":"france"}
OUT: HTTP 500 "Incorrect type for key
'phenomenons_max_colors'"
```

Divergence déclarée/réelle — clé

```
IN: {} // param "optionnel"
OUT: HTTP 400 "Missing required key: domain"
IN: {"domain":"75"}
OUT: {"color_max":1,
      "timelaps":[...],
      "phenomenons_items":[...]}
}
```

cross_param — multilingue

```
IN: {"lang":"en"}
OUT: [{Wind},{Rain-flood},...] // OK
IN: {"lang":"de"} // attendu DE
OUT: [{Vent},{Pluie-inondation},...] // FR!
```

Annexe C — Domaines et boîtes à outils

Récapitulatif — 10 domaines, 60 outils

Agent	Domaine	Outils	Type	Source principale
delhIA	Météorologie	7	REST	webservice.meteofrance.com
Gaia	Environnement / air	5	REST	Open-Meteo, OpenAQ
Marianne	Données publiques	9	REST+MCP	mcp.data.gouv.fr + API REST
WikiBot	Connaissance générale	2	MCP	wikipedia-mcp (GitHub)
Voyageur	Transport & mobilité	4	REST	SNCF Open Data, transport.data.gouv.fr, OSM
Cosmos	Astronomie & espace	6	REST	NASA NeoWs, Open-Notify
NutriScan	Alimentation & nutrition	7	MCP	openfoodfacts-mcp (GitHub)
Archie	Recherche scientifique	1	MCP	arxiv-mcp-server (GitHub)
Thémis	Droit français	12	MCP	mcp.openlegi.fr/legifrance/mcp
BourseIA	Finance & marchés	7	MCP	yahoo-finance-mcp (GitHub)
Total		60		

10+ outils+ 7-9 4-6 1-3

delhIA — Météorologie

REST *API Météo-France*

search_places	géocodage commune / CP
forecast	prévisions horaires / 15j
observation	conditions en temps réel
rain_nowcast	pluie à la minute
alerts	vigilance départementale
bulletin	bulletin complet 24h
warning_dictionary	lexique phénomènes

Gaia — Environnement & air

REST *Open-Meteo + OpenAQ*

env_air_quality_current	AQI, polluants, UV
env_air_quality_forecast	prévisions 1-7 jours
env_monitoring_stations	stations à proximité
env_station_measurements	relevés temps réel
meteo_search_places	géocodage (partagé)

Marianne — Données publiques

REST+MCP *data.gouv.fr MCP + API REST*

search_datasets	recherche jeux de données
get_dataset_info	métadonnées complètes
list_dataset_resources	fichiers d'un dataset
get_resource_info	détail d'un fichier
search_dataservices	APIs publiques vivantes
get_dataservice_info	métadonnées API
get_dataservice_openapi_spec	spécification OpenAPI
query_resource_data	requête tabulaire in-place
get_dataset_metrics	statistiques d'usage

WikiBot — Connaissance générale

MCP *wikipedia-mcp*

search	recherche d'articles
readArticle	lecture article par titre exact

Voyageur — Transport & mobilité

REST *SNCF Open Data + transport.data.gouv.fr + OSM*

transport_nearby_stops	arrêts à proximité
transport_search_stations	gares SNCF par nom
transport_station_details	infos gare (accès, services)
transport_get_datasets	datasets mobilité régionaux

Cosmos — Astronomie & espace

REST *NASA NeoWs + Open-Notify*

astro_apod	image astronomique du jour
astro_neo_feed	astéroïdes sur une période
astro_neo_lookup	fiche complète astéroïde
astro_neo_stats	stats base NEO NASA
astro_iss_position	position ISS temps réel
astro_iss_crew	équipage actuel en orbite

NutriScan — Alimentation & nutrition

MCP *openfoodfacts-mcp*

search_products_standard	recherche par mots-clés
search_products_lucene	recherche Lucene avancée
get_product	fiche produit par code-barres
autocomplete	autocomplétion tags/marques
call_api	appel direct API OFF
get_api_docs	documentation API
get_skill	capacités serveur MCP

Archie — Recherche scientifique

MCP *arxiv-mcp-server*

fetch_arxiv_paper_content	contenu complet par arXiv ID
---------------------------	------------------------------

Thémis — Droit français

MCP *OpenLegi / Légifrance MCP*

rechercher_dans_texte_legal	recherche plein texte
rechercher_code	dans un code juridique
lister_codes_juridiques	liste des codes dispo.
rech._jurisprudence_judiciaire	jurisprudence judiciaire
rech._jurisprudence_admin.	jurisprudence administrative
rech._jurisprudence_financiere	jurisprudence financière
rechercher_decisions_cnll	décisions CNIL
rech._decisions_constitutionnelles	Conseil constit.
rechercher_conventions_coll.	conventions collectives
recherche_journal_officiel	Journal officiel
dernier_journal_officiel	dernier numéro JO
get_decision_*	texte intégral par ID

BourseIA — Finance & marchés

MCP *yahoo-finance-mcp*

yfin_get_quotes	cours temps réel (multi-tickers)
yfin_get_historical_prices	historique OHLCV
yfin_get_stock_info	info entreprise (P/E, dividende)
yfin_get_stock_actions	dividendes & splits
yfin_get_option_dates	dates expiration options
yfin_get_financial_statement	états financiers
yfin_get_recommendations	consensus analystes

Annexe D — Prompts système des agents LLM

Les cinq agents LLM du pipeline reçoivent chacun un *prompt système* fixe (température fixe, sortie structurée).

Agent	Étape	T°	Appels	Entrée	Sortie
Taxonomiste	1	0,1	1 (global)	Toutes les fiches outils	taxonomy.json
Recruteur	5-pré	0,7	$N \times (k+5)$	Fiche + résultats live	argument_pool.json
Narrateur	5a	0,7	1/transition	Contexte narratif + pool	JSON ou texte
Critique	5a	0,1	1/scénario	path + full_story	JSON
PersonaWriter	5b	0,8	1/variante	Scénario + persona	user_scenario JSON

D.1 Taxonomist

Role. You are a semantic data typist working on a tool-graph-based evaluation pipeline for conversational agents.

Task. You will receive a JSON object with one key "cards" : a list of **all** tool identity cards for a domain. Your job is to build a single, complete semantic type taxonomy covering every data type consumed and produced across all tools. Return the **complete** taxonomy in one response. Do not return a partial result.

Input format. "cards" is a list of tool identity card objects. Fields in decreasing order of reliability : consumes/produces (natural-language I/O lists); output_schema/param_schema (structured descriptions); actual_description/declared_description; typical_use_case.

CRITICAL — types are data concepts, not tool names. Type identifiers name *data concepts* — the kind of information flowing between tools. They are **NEVER** named after tools or APIs.

Wrong : "meteo_alerts", "meteo_forecast", "get_weather". *Right* : "weather_alert_data", "gps_coordinates", "temperature".

Tool names belong only in the consumed_by and produced_by arrays.

HARD CONSTRAINT — identifier language. ALL type identifiers MUST be lowercase English snake_case, even when tool cards are in French. Ex : coordonnées GPS → gps_coordinates; code département → department_code. Non-ASCII characters are **forbidden** in identifiers.

Input vs. output — the critical distinction. param_schema and consumes → what the tool *requires as input* → tool goes into consumed_by. output_schema and produces → what the tool *returns* → tool goes into produced_by. Be conservative about type granularity : do not create one type per output field. A rain nowcast tool returns one type rain_nowcast_data, not separate types for timestamp, rain_flag, latitude, etc. Aim for **3 to 6 semantic types per tool**.

Graph connectivity — the primary goal. Read *all* cards before assigning types. For each type *T*, a directed edge is added from every tool in produced_by(*T*) to every tool in consumed_by(*T*). A type with both lists non-empty is a *connector type* — the only kind that creates multi-step graph edges. When tool A produces concept X and tool B consumes concept X, they **must** share the same type identifier.

Type hierarchy rules. Each type has at most one parent (forest structure). Parents do not inherit children's tool lists. Assign each tool to the most specific matching type only. A type with no parent has "parent" : null. Use the most specific type in produced_by; a parent type there is almost always a sign of insufficient specificity.

Output format. Return a single JSON object with one key "types". No "domain" key. No markdown fences. No explanation outside the JSON. Output **compact JSON** — no indentation, no extra whitespace.

```
{"types": {"<type_id>": {"parent": "<parent>|null", "consumed_by": [...], "produced_by": [...]}, ...}}
```

D.2 Recruiter (The Scout)

Role. You are a Tool Scout for a synthetic data generation pipeline targeting a French meteorological agent. Your task is to generate a diverse set of realistic input arguments that can be used to probe a specific tool and verify that it returns valid data.

Context. The pipeline will use the successful (input, output) pairs you help discover as a ground-truth pool. Scenario writers will draw from this pool when they need concrete, verified values — e.g. real French cities with confirmed GPS coordinates, real department codes with confirmed alert data.

Instructions. 1. Analyze the Tool ID Card provided. 2. Generate **10 diverse, realistic input argument sets** for this tool. 3. Diversity means : vary geographic scope (different cities, departments, regions), vary precision (big cities, small towns, overseas territories if applicable), vary edge-cases (city names with accents, departments with single-digit codes padded as "01", etc.). 4. All values must be **real** French geographic entities — no invented place names. 5. For tools that accept GPS coordinates (lat, lon) : use actual coordinates of French cities or landmarks. 6. For tools that accept department codes : use real INSEE department codes (e.g. "75" for Paris, "69" for Rhône, "971" for Guadeloupe).

Output format. Return only a JSON object with a single key candidates whose value is a list of argument dicts. Each dict must use exactly the parameter names declared in the Tool ID Card. Do not add extra keys. Do not include any prose outside the JSON.

```
{"candidates": [{"q": "Lyon"}, {"q": "Marseille"}, ...]}
```

D.3 Narrator

Role. You are a single narrative agent that simultaneously selects tool arguments and writes grounded scenario sentences. You receive structured instructions and return either a JSON object or a plain-text sentence, depending on the operation requested. You handle three types of operations, indicated by the first line of every user message.

Operation : SELECT AND WRITE. *When* : the agent is starting a new tool call from a wildcard node (`Run_Conversation`). **1.** Read the narrative so far to understand what location, time-frame, or context has been established. **2.** From the pool entries, select the entry whose input best fits the established narrative context. Each entry has an `index` field — you will return that index, not the arguments themselves. **3.** Write one sentence from the **user's perspective** : what did the user ask the agent to do? If there is no narrative history (opening of the conversation) : write a natural opening request. If there is a narrative history : the user has just heard the agent's last response — write the user's reaction and their new request. **4.** The sentence must feel like a real user turn. Do not describe what the agent does. **5.** Use real values from the selected pool entry's input to keep the sentence concrete. *Return* : `{"entry_index": <integer>, "sentence": "..."} — no prose.`

Operation : EXTRACT AND WRITE. *When* : the agent is chaining tools — it must call the next tool using the output of the previous one. **1.** Read the previous tool's result carefully and extract the exact values needed by the next tool. **2.** Do not invent values — map them directly from the previous result. **3.** Write one sentence from the **agent's perspective** describing this tool call : focus on the action and arguments. Do not speculate about the result, which is not yet known. *Return* : `{"arguments": {...}, "sentence": "..."} — no prose.`

Operation : WRITE SENTENCE — agent return. *When* : the agent has completed a tool call and is returning the result to the user. Write one sentence from the **agent's perspective** : what does the agent tell the user? Draw on the actual data returned. Be specific (real values, real units). Maintain consistency with the established narrative. *Return* : plain text only — one sentence, no JSON, no labels.

Operation : WRITE SENTENCE — user closing. *When* : the user is ending the conversation after hearing the agent's last response. Write one brief, natural sentence from the **user's perspective** : the user reacts to the last thing the agent said and closes the conversation (expresses satisfaction, thanks the agent, says goodbye, etc.). The reaction should feel specific to the actual content of the last agent response, not generic. *Return* : plain text only — one sentence, no JSON, no labels.

D.4 Critic (The Scenario Validator)

Role. You validate synthetic conversational scenarios for use as evaluation data for a conversational agent. Your job is **not** to judge the literary quality of the story — the prose is scaffolding that will be discarded by the pipeline. Your job is to decide whether the scenario is usable : does it represent a plausible user interaction, and can you derive a coherent evaluation context from it?

The two things that matter downstream are : (1) `tool_calls` — the ordered log of real API calls already made. These become `evaluation_criteria.actions[]` in the final task. They are correct by construction and you do not need to verify them. (2) `initial_situation` and `summary` — which you generate. These become `reason_for_call` and `description.purpose` in the final task.

You receive `path` (the ordered list of nodes visited) and `full_story` (the narrative sentences — prose scaffolding, imperfect by design).

When to reject. Reject only if one of these two conditions holds. **(1) The scenario is implausible as a user interaction** : the tool sequence must represent an information need that a realistic user could plausibly have. Reject if the path is so degenerate that no coherent user motivation exists — e.g. a path that calls the same tool with identical arguments more than twice in a row with no conceivable new information need between calls. Do **not** reject for : repeated sentences, voice inconsistencies, generic closing phrases, raw API values in the prose, or any other prose quality issue. **(2) The story contains argument hallucinations** : reject if the narrative references a specific data value (coordinate, code, status level, proper name) that could not have been produced by the preceding tool calls — i.e. a value that was invented rather than retrieved. Do **not** reject for values that are merely imprecisely described or reformatted in the prose, as long as they are traceable to a real tool result.

When to accept — generating summary and initial_situation.

`summary` : one sentence describing the **agent's capability or tool-chain pattern** being demonstrated — not what happened in this specific instance. Must apply to all stories that follow the same path pattern and must not reference specific entities (cities, codes, dates).

`initial_situation` : two sentences strictly from the **user's perspective**. (1) Why the user is calling and what real-world situation they face. (2) The specific entities they need information about — named directly from the tool calls. Must not describe the agent, the tools, or any internal pipeline step. Write both sentences as definitive factual statements. Do not speculate about the user's lifestyle or motivations beyond what is directly implied by the tool calls. Do not use hedging language (*possibly, perhaps, likely, probably, might*).

Output format.

```
{"is_coherent": true|false, "reason": "...", "summary": "...", "initial_situation": "..."} 
```

Always populate `reason`. Omit `summary` and `initial_situation` if `is_coherent` is false.

D.5 PersonaWriter

Role. You generate the `user_scenario` fields for one evaluation task entry. You receive a story (`path`, `tool_calls`, `initial_situation`) together with a user identity persona and a list of candidate behavioural modifiers, and you produce four text fields that will be used to simulate a challenging user in a live conversation with an AI agent.

Output fields. Return a single JSON object with exactly these four keys. No prose outside the JSON.

task_instructions : 1–4 sentences of behavioural directives written in the **second person** ("You are...", "If the agent asks..."). If a persona is provided, open with one sentence capturing the user's dominant communication style (not a copy of the persona paragraph — distil into a single directive relevant to this scenario). For each candidate modifier that is **applicable to this specific scenario**, add one concrete directive grounded in the real tool call arguments. Substitute the bracketed placeholders (e.g. `[IDENTIFIER]`, `[WRONG VALUE]`, `[CONDITION]`) with actual values drawn from the tool calls. **Skip any modifier that does not fit this scenario** — do not force-fit modifiers that have no sensible grounding in the tool calls or initial situation. Do not invent values; every specific reference must come from the tool calls or initial_situation provided.

reason_for_call : Rewrite `initial_situation` as a 2nd-person statement ("You want to..."), **enumerating the specific entities from the tool calls** — department numbers, location names, postal codes, GPS coordinates, or any other domain-specific identifiers — that the user needs to request information about. 2–3 sentences. Write as definitive factual statements — do not use speculative language (*possibly, perhaps, likely, probably, might*). No mention of tool names or pipeline internals.

known_info : A brief statement of the user's identity : at minimum a plausible full name. If the tool calls contain a user identifier (account number, loyalty ID, zip code) it may be included. If no identity can be inferred from context, synthesise a realistic name consistent with the domain's language and locale. Format : "Your name is X. Your [ID type] is Y." (omit second sentence if no ID available).

unknown_info : If the `B-FORGET-ID` modifier is active and applicable, write one sentence stating what the user does not remember (e.g. "You do not remember the department number."). Otherwise set to `null`.

Grounding rule. Every concrete value appearing in `task_instructions` (an identifier, a number, a name, a condition) must be traceable to the tool call arguments or results provided. Do not invent domain data.

Output format.

```
{"task_instructions": "...", "reason_for_call": "...", "known_info": "...", "unknown_info": null} 
```

Annexe E — Création d'un scénario : pipeline complet (tâche 3, delhIA)

Étape 1. Trajectoire échantillonnée. Le Sampler tire le chemin suivant dans le graphe markovien :

RC → meteo_bulletin → RC → meteo_search_places → meteo_forecast → RC
→ EoS

Étape 2. Hydratation arc par arc (Narrator). Arc 1 [SEL. & WRITE] RC → meteo_bulletin, arg : domain="75"

« L'utilisateur demande le bulletin de vigilance pour le dép. 75. »

Arc 2 [WRITE SENT.] meteo_bulletin → RC — résultat réel : 5 phénomènes, tous niveau 1 (vert)

« Le bulletin indique quatre phénomènes au niveau vert. »

Arc 3 [SEL. & WRITE] RC → meteo_search_places, arg : q="Paris"

« L'utilisateur demande les coordonnées GPS de Paris. »

Arc 4 [EXT. & WRITE] meteo_search_places → meteo_forecast

Coords extraites du résultat réel : lat=48.859, lon=2.347 — appel live de meteo_forecast

« L'agent interroge les prévisions 15 jours pour Paris. »

Arc 5 [WRITE SENT.] meteo_forecast → RC — résultat réel : 6,2–21,1 °C, ciel dégagé

« Aucune précipitation significative prévue sur 15 jours. »

Arc 6 [WRITE SENT.] RC → EoS

« Pas besoin de parapluie de sitôt! »

Point clé : à l'arc 4, les coordonnées GPS ne sont pas générées par le LLM — elles sont propagées directement depuis la réponse réelle de search_places (mode EXT. & WRITE).

Étape 3. Validation par l'agent Critique.

is_coherent : vrai

Raison : « Scénario plausible : l'utilisateur consulte d'abord les alertes d'un département, puis affine avec des prévisions pour une ville précise. »

Persona généré : Jean-Pierre Lefebvre, retraité, planifie des activités en plein air.

Étape 4. Critères d'évaluation générés (TaskBuilder).

#	Outil	Arguments attendus	compare_args
1	meteo_bulletin	domain="75"	["domain"]
2	meteo_search_places	q="Paris"	["q"]
3	meteo_forecast	lat=48.859, lon=2.347	["lat", "lon"]

Les résultats d'évaluation sur cette tâche ($R_{strict} = 1,00$ pour les 3 modèles) sont détaillés en Annexe F, Partie 1.

Annexe F — Étude de cas : deux instanciations contrastées

Partie 1 — Scénario nominal : tâche 3 (API fonctionnelle)

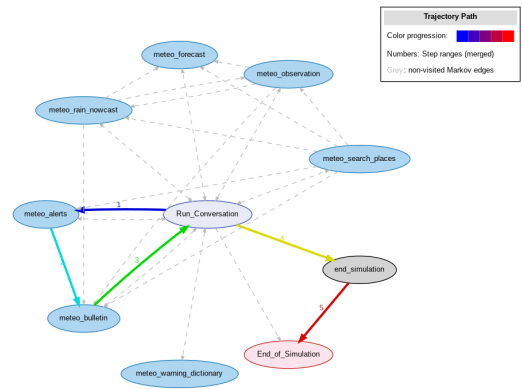
Tâche. Jean-Pierre Lefebvre souhaite vérifier les alertes météo pour les départements 75 et 44 et obtenir le bulletin du département 13 avant des activités en plein air. La trajectoire attendue comporte **3 actions** et l'API fonctionne normalement.

	Modèle	Tours	Appels	R _n	R _s	Statut
Récapitulatif.	gpt-4o	10	3	1,00	1,00	FINISHED
	qwen3-32b	12	3	1,00	1,00	FINISHED
	qwen3-8b	10	3	1,00	1,00	FINISHED

Trajectoires.

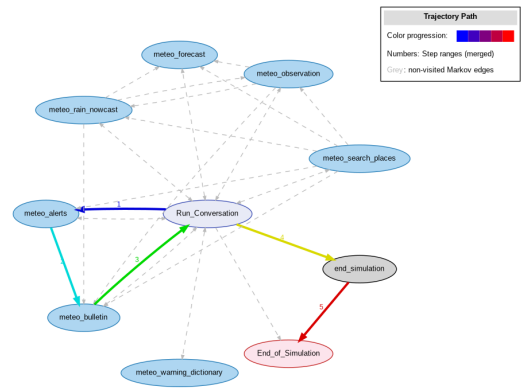
gpt-4o — séquentiel, 3 appels en 3 tours

[USER] « alertes dép. 75, 44 + bulletin 13 »
 [A→T] meteo_alerts (75) ✓
 [A→T] meteo_alerts (44) ✓
 [A→T] meteo_bulletin (13) ✓
 [A→T] end_simulation



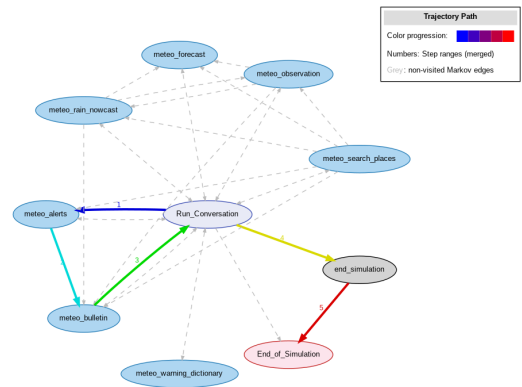
qwen3-32b — séquentiel, 3 appels en 3 tours

[USER] « alertes dép. 75, 44 + bulletin 13 »
 [A→T] meteo_alerts (75) ✓
 [A→T] meteo_alerts (44) ✓
 [A→T] meteo_bulletin (13) ✓
 [A→T] end_simulation



qwen3-8b — parallèle, 3 appels en 1 tour

[USER] « alertes dép. 75, 44 + bulletin 13 »
 [A→T] meteo_alerts (75) ✓
 meteo_alerts (44) ✓
 meteo_bulletin (13) ✓
 [A] « Voici les informations... »
 [A→T] end_simulation



Analyse. Les trois modèles atteignent $R_{\text{strict}} = 1,00$: tous les outils attendus sont appelés avec les bons arguments. On note une différence de *style* : gpt-4o et qwen3-32b font les appels séquentiellement, tour par tour ; qwen3-8b les parallélise en un seul tour, réduisant la conversation à 10 tours contre 12 pour qwen3-32b. Les deux stratégies sont correctes ; la parallélisation est plus efficace mais n'est pas toujours applicable (certains outils dépendent des résultats du précédent).

Partie 2 — Scénario multi-villes : tâche 34 (déviation et hallucination)

Tâche. Marie-Claire Fontaine souhaite vérifier les conditions météorologiques pour la région parisienne et Toulouse avant des activités en plein air. La trajectoire attendue comporte **7 actions** et l'API est fonctionnelle :

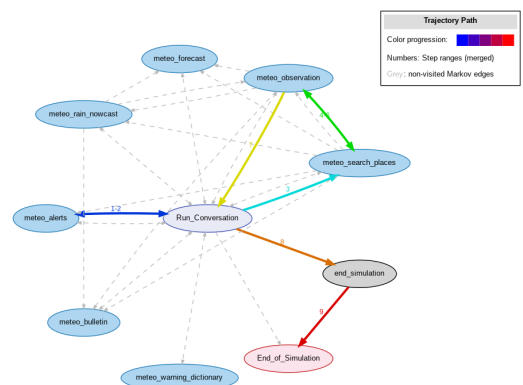
#	Outil	Arguments	Rôle dans la trajectoire
1	meteo_search_places	q="Paris"	géocodage Paris
2	meteo_rain_nowcast	lat=48.86, lon=2.35	pluie immédiate Paris
3	meteo_bulletin	domain="75"	bulletin Paris
4	meteo_alerts	domain="75"	alertes Paris
5	meteo_bulletin	domain="13"	bulletin Marseille
6	meteo_search_places	q="Toulouse"	géocodage Toulouse
7	meteo_bulletin	domain="13"	(second bulletin dept 13)

	Modèle	Tours	Outils	Erreurs	R_n	R_s	Statut
Récapitulatif par modèle.	gpt-4o	20	6	0	0,43	0,14	FINISHED
	qwen3-32b	20	7	0	0,43	0,00	CRASHED
	qwen3-8b	35	11	4	0,43	0,14	FINISHED

Extraits de trajectoires.

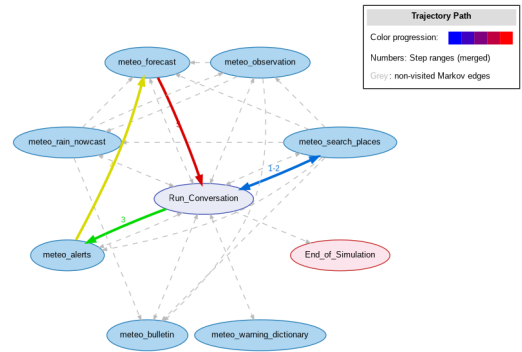
gpt-4o — FINISHED, 20 tours — simplification

[A→T] meteo_alerts (75) ✓
 [A→T] meteo_search_places ("Lyon")
 [A→T] meteo_observation (lat=45.76, lon=4.84)
 [A→T] meteo_search_places ("Marseille")
 [A→T] meteo_observation (lat=43.28, lon=5.41)
 Substitue observation à forecast ; dévie sur Lyon/Marseille



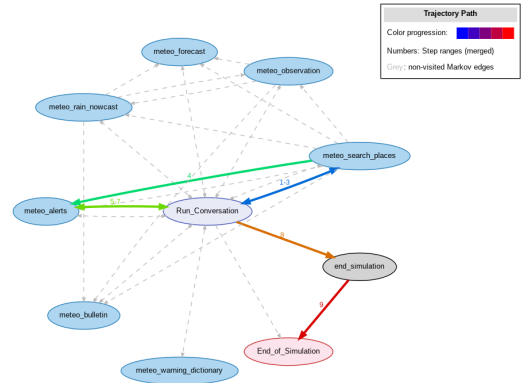
qwen3-32b — CRASHED, 20 tours — déviation hors-périmètre

[A→T] meteo_search_places ("Lyon")
 [A→T] meteo_search_places ("Marseille")
 [A→T] meteo_search_places ("Isère")
 [hors tâche]
 [A→T] meteo_alerts (38) [dept 38 ≠ 75/13]
 [A→T] meteo_forecast ×3 — crash



qwen3-8b — FINISHED, 35 tours — hallucination d'outil

[A→T] meteo_search_places ×5
 [A→T] meteo_alerts ({domain: "06"}) ✓
 [A→T] get_meteo_alerts (...) [outil inexistant → 404]
 [A→T] get_meteo_alerts (...) [404 à nouveau]
 Récupère malgré tout; end_simulation après 35 tours



Analyse. Cette tâche illustre trois stratégies divergentes sur un scénario sans erreur serveur. **gpt-4o** simplifie : il substitue `observation` à `forecast` et dévie sur Lyon/Marseille plutôt que Paris/Toulouse, finissant avec $R_s = 0,14$. **qwen3-32b** dévie hors-périmètre dès le départ : il cherche l'Isère et le département 38 au lieu de Paris et Toulouse, appelle trois fois `meteo_forecast` avec des coordonnées incorrectes et crashe — $R_s = 0$. **qwen3-8b** hallucine un nom d'outil (`get_meteo_alerts`, 404), mais récupère et clôture après 35 tours ($R_s = 0,14$). Le cas le plus instructif est l'égalité $R_{name} = 0,43$ pour les trois modèles : même couverture d'outils en surface, mais issues radicalement différentes (FINISHED, CRASHED, FINISHED). Ce cas démontre que le taux de complétion seul ne suffit pas à caractériser la qualité d'un agent : les métriques de trajectoire (R_{strict} , taux d'écho, nombre de tentatives sur erreur) sont nécessaires pour distinguer une clôture gracieuse d'un abandon non détecté.