Eigenflow

GUIDE · AGENTS IA

Harness² : le harness qui s'améliore tout seul

Ce qu'il change vraiment, ses chiffres avec leurs conditions, ce qu'il coûte, comment l'essayer sans risque, et sa méthode refaite à la main dans Claude Code.

PAR

Abyane Nabeebaccus

INSTAGRAM

@abyane_nb

ÉDITION

novembre 2026

Tu as commenté AGENT, voici le guide. Harness² est un outil gratuit et libre (licence Apache-2.0), publié sur GitHub dans l'organisation google-research par des chercheurs de Google Cloud AI Research et de Northwestern University. Il prend le harness d'un agent, c'est-à-dire tout ce qui entoure le modèle (consignes, outils, mémoire, skills), et le règle lui-même, tâche après tâche, sans toucher au modèle. Le dépôt : github.com/google-research/harness2.

Nous n'avons pas lancé Harness² : ce guide, écrit le 19 novembre 2026 et relu le 25 novembre, s'appuie sur l'article, la page officielle et le dépôt.

À savoir avant de lire

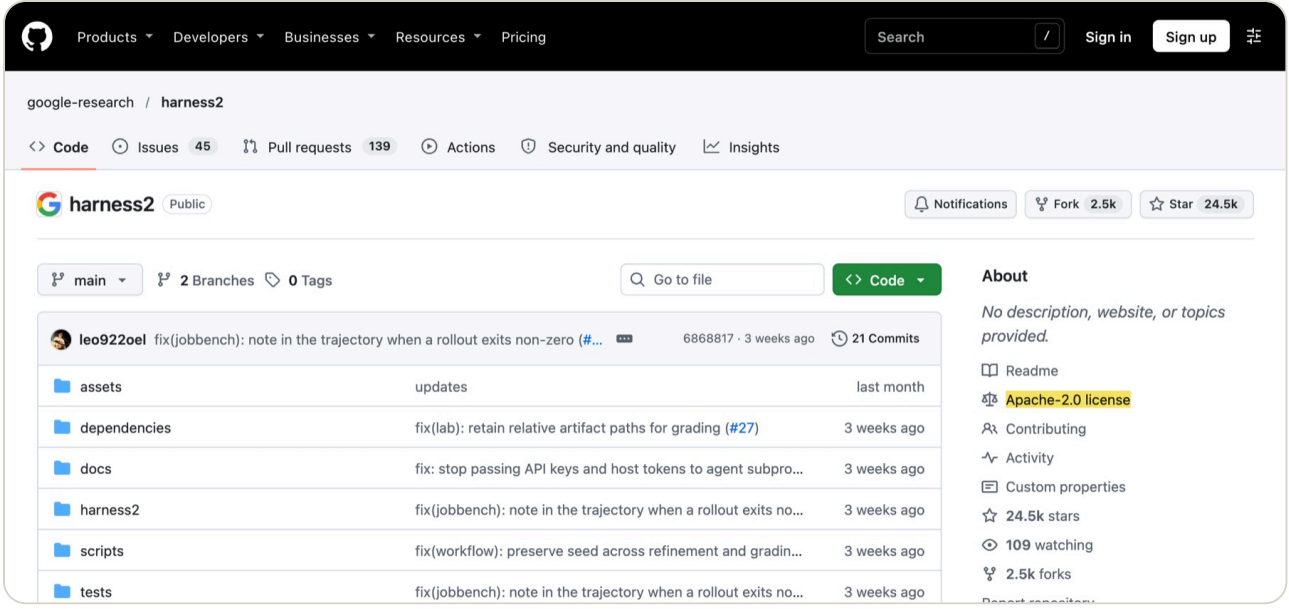
- Le modèle ne change pas. Harness² modifie des fichiers autour de lui. Les poids restent figés : aucun réentraînement.
- Il ne se branche pas sur Claude Code. Les agents qu'il règle tournent sous OpenCode ou Codex. Un modèle Claude peut servir d'améliorateur, mais par Google Cloud, jamais par ton abonnement Claude.
- Il faut Linux, Python 3.12 et uv. Pour les vrais tests, il faut en plus Node.js 20 ou plus, la CLI Google Cloud et un projet Vertex AI payé à l'usage.
- La démo locale est gratuite et sans clé, mais elle ne contient aucune IA : ses modifications sont écrites d'avance.
- Il ne tourne que sur ses trois bancs de test et sa démo. Pour tes propres tâches, il faut écrire un adaptateur en Python (`harness2/core/spec.py`).
- Le dépôt le dit lui-même : « This is not an officially supported Google product. »
- WorkBuddy-Bench, le banc de test du +11,5, n'est pas prévu pour un usage dans l'Union européenne (licence de Tencent).

---

1. Harness² en deux phrases

Le README le définit ainsi : un harness est le système autour d'un modèle qui lui permet d'agir comme un agent. Harness² règle ce système pour la tâche en cours, puis garde ce qu'il a appris pour mieux régler les tâches suivantes.

Le dépôt a été créé le 6 octobre 2026 et l'article mis en ligne sur arXiv le 9 octobre. Le 25 novembre, il comptait 25 191 étoiles, 2 521 copies (forks) et toujours 21 commits : rien n'a bougé dans le code depuis le 2 novembre.



Le dépôt google-research/harness2 sur GitHub : 24 500 étoiles, 2 500 copies, licence Apache-2.0 (capture du 19 novembre)

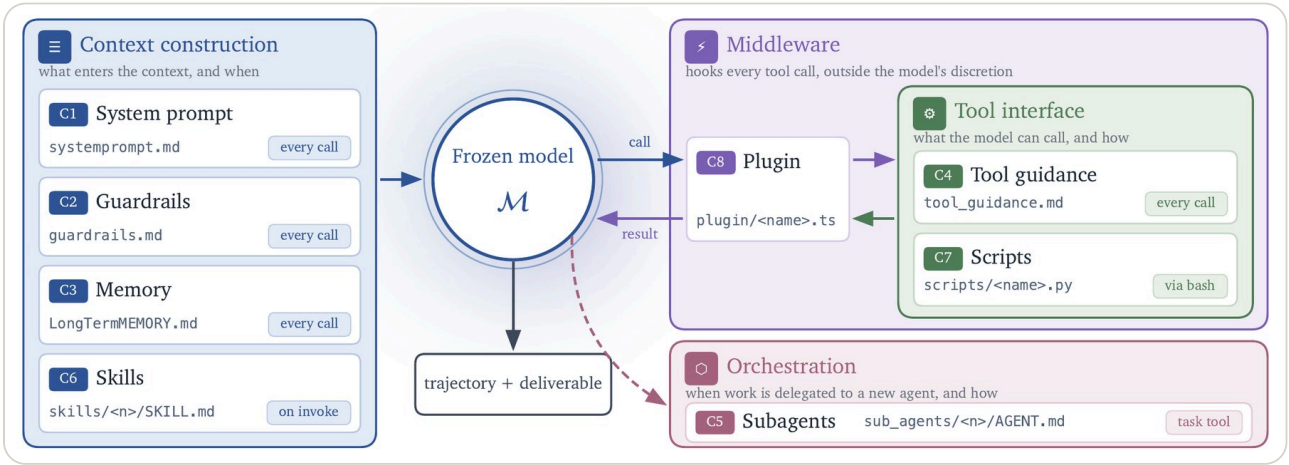
---

2. Ce qu'il change, et comment

Les huit pièces d'un harness

L'article découpe le harness en huit fichiers, tous modifiables :

- le prompt système (`systemprompt.md`) : la façon de travailler, lue à chaque appel ;
- les garde-fous (`guardrails.md`) : les règles qui tiennent quelle que soit la tâche ;
- la mémoire (`LongTermMEMORY.md`) : les faits durables d'un domaine ;
- les consignes d'outils (`tool_guidance.md`) : quand et comment se servir de chaque outil ;
- les sous-agents (`sub_agents/<nom>/AGENT.md`) : des agents délégués qui rendent seulement leur résultat ;
- les skills (`skills/<nom>/SKILL.md`) : une procédure chargée seulement quand l'agent en a besoin ;
- les scripts (`scripts/<nom>.py`) : un calcul confié à du code plutôt qu'au modèle ;
- le plugin (`plugin/<nom>.ts`) : du code qui s'intercale autour de chaque appel d'outil.



Les huit pièces du harness autour du modèle figé (image du dépôt Harness², © 2026 Google LLC, licence Apache-2.0)

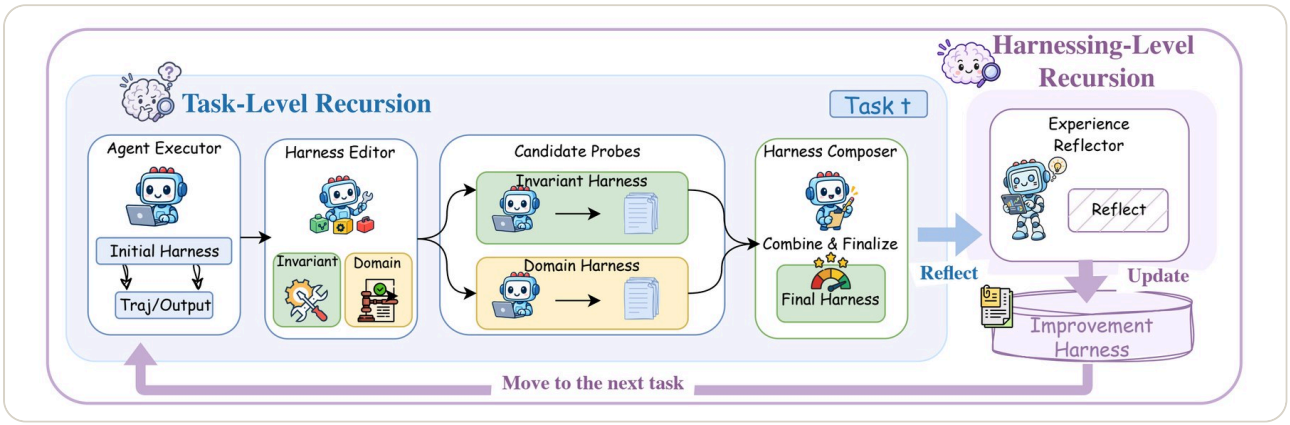
Proposer, tester, composer

Pour chaque tâche, Harness² fait d'abord tourner l'agent avec le harness de départ. Puis l'améliorateur (un modèle, parfois le même que celui de l'agent) lit ce qui s'est passé et propose deux versions modifiées :

- une version « invariante », qui vise les défauts qu'on retrouve partout : planifier, se servir des outils, gérer le contexte, déléguer ;
- une version « propre au domaine », qui ajoute les connaissances et les étapes du métier en cours.

Chaque version refait la même tâche, dans les mêmes conditions. L'améliorateur compare les trois exécutions et compose le harness final : il garde les changements dont l'effet se voit, en corrige d'autres ou laisse tout tel quel. Il ne connaît pas la bonne réponse : il juge sur ce que l'agent a fait. L'article le dit, les modifications sont guidées « without ground-truth verifiers ».

Un pas complet, c'est donc quatre exécutions de la même tâche (départ, deux essais, version finale), trois sessions de l'améliorateur, puis une de réflexion. C'est ce que le README appelle `--k 1`.

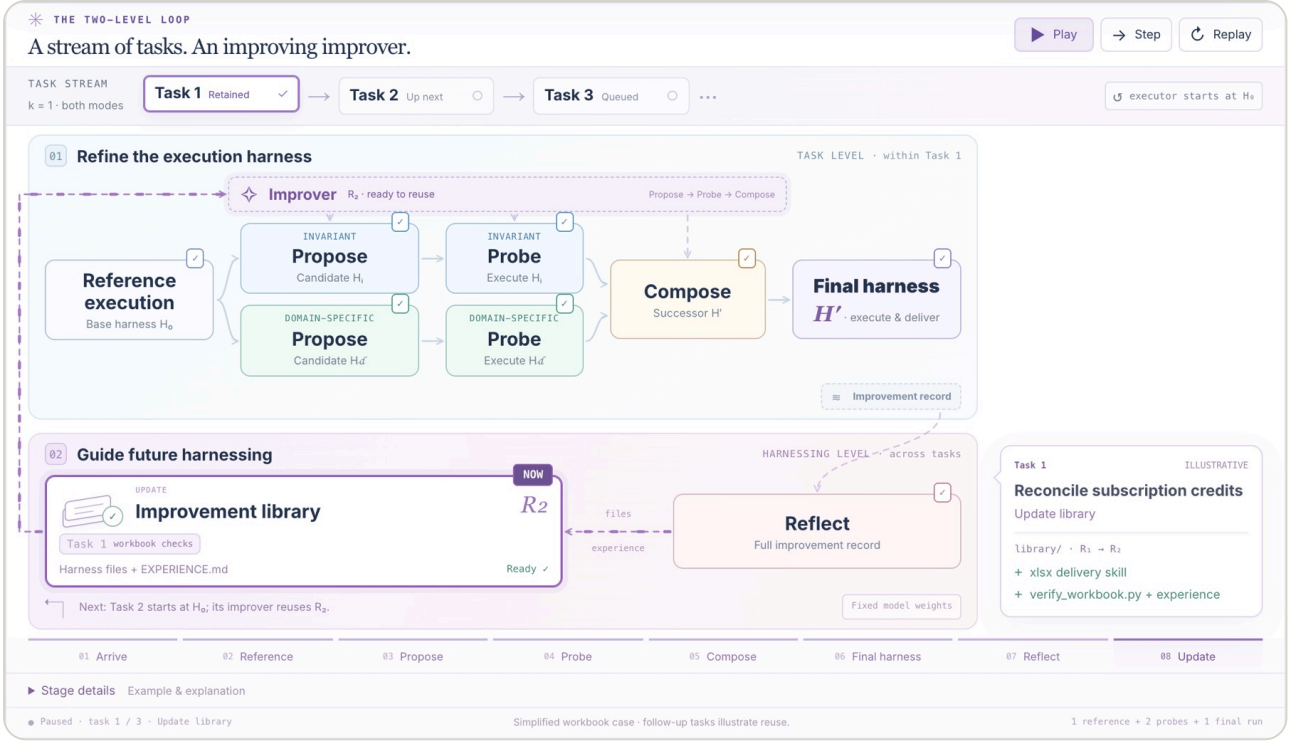


La boucle d'une tâche (agent, éditeur, deux essais, compositeur), puis la réflexion qui met à jour la bibliothèque (image du dépôt Harness², © 2026 Google LLC, licence Apache-2.0)

Ce qui est gardé d'une tâche à l'autre

Après chaque tâche, l'améliorateur relit tout ce qui s'est passé. Si les essais montrent clairement ce qui a aidé, il range dans sa bibliothèque le harness retenu et un fichier `EXPERIENCE.md` en quatre lignes : la situation, la leçon, la preuve, la confiance. Si la preuve manque, il n'écrit rien.

À la tâche suivante, l'agent repart du harness de départ et n'hérite de rien d'office. C'est l'améliorateur qui relit la bibliothèque, et il peut y reprendre une pièce quand il propose ses modifications. L'article parle d'un « improving improver » : l'outil apprend à mieux améliorer. La bibliothèque est rangée par domaine, et séparée par banc de test, exécuteur, mode et budget : elle ne te suit pas d'un projet à l'autre.



La page officielle, étape « Update » : la bibliothèque reçoit les fichiers et EXPERIENCE.md, la tâche 2 repart de H_0 et son améliorateur relit la bibliothèque (capture du 19 novembre)

---

3. Les chiffres, avec leurs conditions

Les auteurs ont testé deux modèles, GLM-5 et Gemini 3.5 Flash, sous OpenCode et sous Codex, sur trois bancs de test publics. Chaque chiffre est une moyenne de trois séries.

[Submitted on 9 Oct 2026]

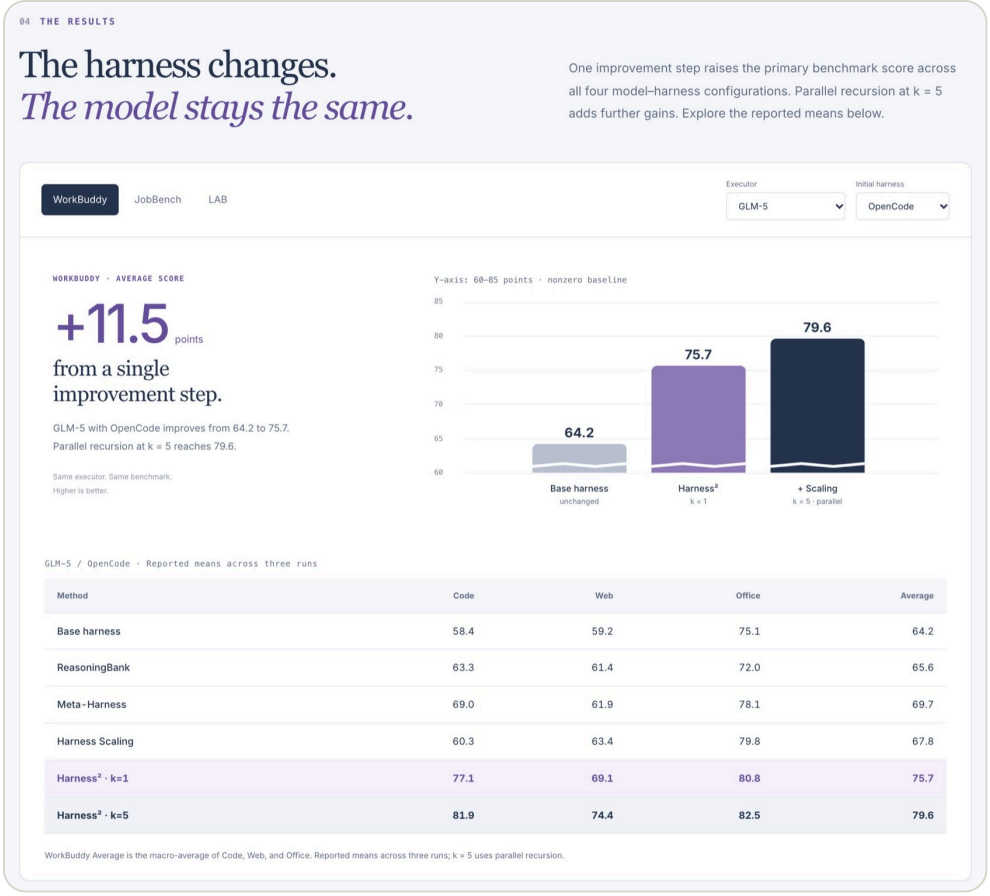
Harness^2: Recursive Agent Harnessing for an Open World

Ruiyao Xu, Yanfei Chen, Zhongying CuiZhu, Bhavana Dalvi Mishra, Yifei Ming, Han Yu, Rujun Han, Chen-Yu Lee, Tomas Pfister

LLM agents' capabilities depend not only on their underlying models but also on their execution harnesses. Recursive self-improvement (RSI) offers a promising approach to automatically evolve these harnesses. However, deploying RSI in open-world settings still faces two critical challenges. First, open-world tasks are typically unseen during the harness design phase and novel at test time, leaving no ground-truth verifiers to reliably guide online harness edits. Second, the RSI process remains highly task-specific and fails to generalize to novel scenarios. We introduce Harness^2, a plug-and-play, recursive agent harnessing framework that addresses these challenges through two connected levels of recursion. Task-level recursion refines the current task's harness using a contrastive Propose-Probe-Compose step that reliably guides edits without ground-truth verifiers, and can be scaled further via parallel or sequential recursion. Harnessing-level recursion then extracts experience from these edits to update a continual improvement harness, enabling the agent to better adapt to future tasks. Across three professional benchmarks and four model-harness configurations, a single step consistently improves GLM-5 and Gemini 3.5 Flash over their base harnesses, yielding gains of up to 11.5 points on WorkBuddy-Bench, 4.5 points on JobBench, and 11.8 percentage points on LAB, with further gains as the task-level recursion budget increases.

Le résumé de l'article sur arXiv, publié le 9 octobre 2026 : jusqu'à 11,5 points sur WorkBuddy-Bench, 4,5 sur JobBench, 11,8 points de pourcentage sur LAB

- +11,5 points : WorkBuddy-Bench (tâches de bureau, web et code), GLM-5 avec OpenCode, de 64,2 à 75,7, en un pas.
- +4,5 points : JobBench (tâches de métier longues), Gemini 3.5 Flash avec OpenCode, de 31,5 à 36,0.
- +11,8 points de pourcentage : LAB (tâches juridiques de Harvey), GLM-5 avec OpenCode, de 59,6 % à 71,4 % de critères réussis.



La page officielle, onglet WorkBuddy, exécuteur GLM-5 et OpenCode : 64,2 au départ, 75,7 après un pas, 79,6 à k = 5 (capture du 19 novembre)

Ce que ces chiffres ne disent pas tout seuls :

- Le modèle le plus faible gagne le plus. Avec Gemini 3.5 Flash, un pas rapporte entre 1,4 et 5,1 points selon le banc et l'exécuteur.
- Le gain se mesure sur la tâche même où le harness a été réglé, après quatre exécutions.
- Sur LAB avec Gemini et Codex, une autre méthode, Meta-Harness, fait mieux (76,5 % contre 75,6 %).
- Dans les tests principaux, l'améliorateur était Claude Opus 4.8, sous OpenCode. Les auteurs ont aussi confié les deux rôles au même modèle, avec deux versions de chaque sorte : les cinq modèles essayés, de GLM-5 à Claude Opus 5, progressent sur les trois bancs.

---

4. Ce qu'il te faut, et ce que ça coûte

- Une machine Linux. Le README ne parle ni de macOS ni de Windows : sur Mac, prends une machine virtuelle Linux ou un serveur sans rien de sensible dessus.
- Python 3.12 et uv (le paquet accepte aussi 3.13).
- Pour un banc de test : Node.js 20 ou plus, la CLI Google Cloud, un projet Vertex AI qui a accès aux modèles choisis. LAB demande Pandoc et Poppler, WorkBuddy demande Docker.

Le prix de Gemini 3.5 Flash sur Vertex AI, le 25 novembre (région globale) : 1,50 $ le million de jetons en entrée, 0,15 $ en cache, 9 $ en sortie. Au cours de la BCE du jour (1 $ = 0,91449 €), environ 1,37 €, 0,14 € et 8,23 € hors taxes.

Sur la figure 6 de l'article (lecture à l'œil, échelle logarithmique), une tâche JobBench consomme environ 5 millions de jetons sans Harness[2] et autour de 27 millions avec un pas ; une tâche WorkBuddy, environ 0,3 million contre un peu plus de 2. Calcul à nous : 27 millions de jetons au prix d'entrée plein, c'est environ 37 € hors taxes. Le cache coûte dix fois moins, la sortie six fois plus, et l'article ne donne pas la répartition : c'est un ordre de grandeur. Rien n'est publié pour LAB.

L'exécuteur Codex, en option, ne passe pas par ton compte ChatGPT : il appelle le pont fourni (`harness2-bridge`, vers Gemini sur Vertex AI) ou une autre API avec sa clé.

Les bancs de test ont leur propre licence. LAB est sous licence MIT, JobBench sous Apache-2.0. WorkBuddy-Bench est sous une licence de Tencent qui écarte l'Union européenne :

```
Tencent is pleased to support the community by making WorkBuddy Bench available.

Copyright (C) 2026 Tencent. All rights reserved. WorkBuddy Bench IS NOT INTENDED FOR USE WITHIN THE EUROPEAN UNION.

WorkBuddy Bench is licensed under the License Terms of WorkBuddy Bench except for the third-party components listed below, which is licensed under different terms. WorkBuddy Bench does not impose any additional limitations beyond what is outlined in the respective licenses of these third-party components. Users must comply with all terms and conditions of original licenses of these third-party components and must ensure that the usage of the third party components adheres to all relevant laws and regulations.

For avoidance of doubts, WorkBuddy Bench refers to the inference enabling code, parameters and weights made publicly available by Tencent in accordance with the License Terms of WorkBuddy Bench in this repository.

Terms of the License Terms of WorkBuddy Bench:
--------------------------------------------------------------------

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
0. Additional Territorial Limitation

*WorkBuddy Bench IS NOT INTENDED FOR USE WITHIN THE EUROPEAN UNION.*
IN THE EVENT OF ANY CONFLICT, THIS CLAUSE SHALL PREVAIL.
```

La licence de WorkBuddy-Bench : « not intended for use within the European Union », clause qui prime sur le reste (capture du 19 novembre)

---

5. L'essayer pas à pas

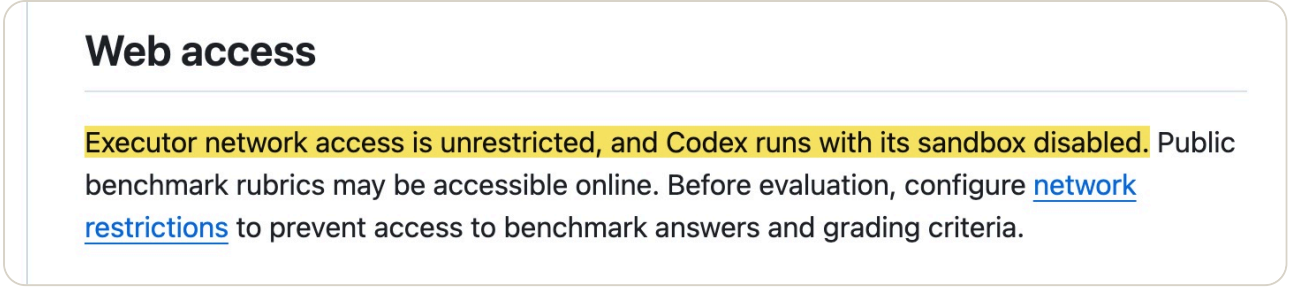
Lire avant de lancer

Ouvre Claude Code dans un dossier vide, en mode Manual :

À COLLER

```
Lis le README, docs/SETUP.md et docs/EXPERIMENTS.md de
https://github.com/google-research/harness2. Dis-moi ce que harness2-setup et
harness2 téléchargent, installent et lancent sur une machine, quels services
Google Cloud ils appellent, et ce qui se paie. Ne lance rien.
```

Tu dois y retrouver ceci : l'installation clone des dépôts dans `dependencies/` et télécharge des données depuis Hugging Face, et les agents lancés ont un accès réseau complet.



Le README : l'accès réseau des exécuteurs n'est pas limité, et Codex tourne sans bac à sable (capture du 19 novembre)

Pendant un test des auteurs, un agent Codex a téléchargé la grille de notation de sa tâche JobBench sans qu'on le lui demande. Fais tourner tout ça sur une machine où rien d'autre ne traîne.

La démo locale, sans clé

Sur la machine Linux, les commandes du README :

```
À COLLER

git clone https://github.com/google-research/harness2
cd harness2
uv venv --python 3.12 .venv
source .venv/bin/activate
uv pip install -e '.[render]'
harness2-demo --output runs/demo --k 2
```

La démo traite deux petites tâches de comptabilité : additionner les montants d'un fichier. Le script de départ oublie les montants négatifs, l'éditeur scripté retire ce filtre, et la bibliothèque reçoit un `EXPERIENCE.md`. Aucun modèle n'est appelé : tu y vois l'arborescence de `runs/demo` et le format d'une leçon, rien sur la qualité de l'outil.

Une vraie tâche, une seule

Le README prend LAB en exemple. La première commande crée `.env` : remplis-le avec ton projet Google Cloud et les modèles auxquels il a accès, avant la suite.

```
À COLLER

cp .env.example .env
gcloud auth application-default login
harness2-setup lab --dry-run
harness2-setup lab
harness2-check --bench lab
harness2 --bench lab --k 1 --mode parallel --only-domains antitrust-competition
--max-tasks 1 --dry-run
```

`--dry-run` montre ce qui sera lancé sans appeler de modèle. Si tout est bon, relance la dernière commande sans `--dry-run`, puis note le résultat :

À COLLER

```
harness2-judge --bench lab --k 1 --mode parallel
harness2-report --bench lab --k 1 --mode parallel
```

Les résultats arrivent dans `runs/lab/` : les livrables, le harness retenu, les notes de réflexion, la note du juge. La bibliothèque est dans `runs/lab/library/`, un `INDEX.md` par domaine et un `EXPERIENCE.md` par tâche retenue. Avec une seule tâche, elle reçoit au plus une entrée. Elle ne sert qu'à partir de la deuxième tâche du même domaine (`--max-tasks 2`).

Avant de lancer, crée un budget dans Google Cloud. Google prévient qu'un budget avec alertes seules ne coupe rien : il t'envoie un e-mail. Le plafond de dépense n'existe que pour certains services.

---

6. La méthode refaite à la main dans Claude Code

Claude Code a les mêmes pièces sous d'autres noms. Ton `CLAUDE.md` joue le prompt système, les garde-fous, la mémoire et les consignes d'outils. Tes skills sont dans `.claude/skills/<nom>/SKILL.md`, tes sous-agents dans `.claude/agents/`, et les hooks font le travail du plugin, autour de chaque appel d'outil. Tu peux donc refaire la boucle toi-même, sur ton abonnement, sans Google Cloud.

Prends une tâche que tu refais souvent (un devis à partir d'un brief, un export comptable à rapprocher), garde ses fichiers d'entrée et le dernier résultat.

Première demande, proposer deux modifications :

À COLLER

```
Lis mon CLAUDE.md, puis le résultat de la tâche [nom de la tâche] dans
[dossier]. Sans rien modifier, propose deux changements séparés. A : une règle
générale qui aurait évité ce que tu vois (planifier, vérifier, relire ce que tu
as écrit). B : une consigne propre à ce métier (une étape, une skill ou un
script). Écris-les dans essais/A.md et essais/B.md, quinze lignes au plus
chacun, avec la raison.
```

Puis teste-les sur la même tâche. Copie ton projet deux fois :

À COLLER

```
cp -R mon-projet essai-A
cp -R mon-projet essai-B
```

Ajoute A au `CLAUDE.md` de `essai-A`, B à celui de `essai-B`. Ouvre une nouvelle session de Claude Code dans chaque dossier, pour qu'il relise son `CLAUDE.md`, et colle exactement la même demande que la première fois.

Deuxième demande, comparer :

À COLLER

```
Compare trois résultats de la même tâche : sans changement (mon-projet), avec A (essai-A), avec B (essai-B). Pour chaque différence, dis ce qui a changé dans la façon de travailler et si ça vient de A ou de B. Décris ce que tu observes, sans décider qui a raison.
```

Toi, tu connais la bonne réponse : juge lequel des trois résultats est juste.

Troisième demande, garder et noter la leçon :

À COLLER

```
Dans mon CLAUDE.md, garde seulement les lignes de A et de B dont l'effet se voit dans les essais. Puis ajoute une entrée à EXPERIENCE.md en quatre lignes : quand (la situation), leçon, preuve (ce que tu as vu dans les essais), confiance. Si rien n'est prouvé, n'écris rien dans EXPERIENCE.md.
```

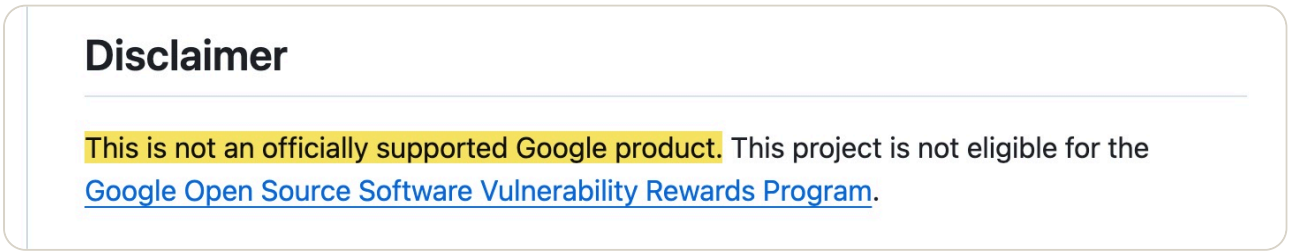
Ajoute enfin une ligne à ton `CLAUDE.md` : « Avant de proposer une modification de ce fichier, lis EXPERIENCE.md. » C'est la bibliothèque d'amélioration, en version d'un seul fichier.

Une exécution par version, c'est peu : deux essais de la même tâche peuvent différer sans aucun changement. Avant de garder une règle qui compte, refais le test une seconde fois.

---

7. Limites

- Les chiffres viennent des auteurs, sur des bancs de test publics. Nous ne les avons pas refaits.
- Sans la bonne réponse, l'outil juge sur des comparaisons de comportement. L'article le dit : ces signaux « do not guarantee correctness ».
- Tes tâches partent chez Google Cloud et chez le fournisseur du modèle choisi. N'y mets pas les documents d'un client sans son accord.
- Le projet a sept semaines (version 0.1.0, dernier commit le 2 novembre) ; au 25 novembre, 48 tickets et 139 propositions de modification ouverts.



Le README : « This is not an officially supported Google product » (capture du 19 novembre)

Sources

- Harness² : le dépôt, le guide d'installation, le guide des expériences, les bancs de test, la page officielle, les résultats en données.
- L'article : Ruiyao Xu, Yanfei Chen, Zhongying CuiZhu, Bhavana Dalvi Mishra, Yifei Ming, Han Yu, Rujun Han, Chen-Yu Lee et Tomas Pfister, Harness² sur arXiv.
- Bancs de test : LAB, JobBench, la licence de WorkBuddy-Bench.
- Coûts : prix de Vertex AI, budgets Google Cloud, cours BCE par Frankfurter.
- Claude Code : CLAUDE.md et mémoire, skills, sous-agents, hooks.

Abyane · @abyane_nb · eigenflow.fr

Eigenflow